UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

LÍVIA ALMADA CRUZ

CONSULTAS *K*NN EM REDES DEPENDENTES DO TEMPO

FORTALEZA, CEARÁ

2013

LÍVIA ALMADA CRUZ

CONSULTAS *K*NN EM REDES DEPENDENTES DO TEMPO

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Banco de Dados

Orientador: Prof. Dr. José Antônio F. de Macedo

Co-Orientador: Prof. Dr. Mário A. Nascimento

FORTALEZA, CEARÁ

2013

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca de Ciências e Tecnologia

962c Cruz, Lívia Almada.
Consultas KNN em redes dependentes do tempo. / Lívia Almada Cruz. – 2013.
75f. : il. color., enc. ; 30 cm.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Departamento de Computação, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2013.
Área de Concentração: Banco de Dados.
Orientação: Prof. Dr. José Antônio Fernandes Macêdo.
Coorientação: Prof. Dr. Mário Antônio Nascimento.

1. Sistema de localização automática de veículos a motor. 2. Sistemas de posicionamento global. 3. Trânsito – controle eletrônico. I. Título.

CDD 005

LÍVIA ALMADA CRUZ

CONSULTAS *K*NN EM REDES DEPENDENTES DO TEMPO

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação. Área de concentração: Banco de Dados

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. José Antônio F. de Macedo
Universidade Federal do Ceará - UFC
Orientador

Prof. Dr. Mário A. Nascimento
University of Alberta
Co-orientador

Prof. Dr. Marco Antonio Casanova
Pontifícia Universidade Católica do Rio de Janeiro - PUC Rio

Prof. Dr. Javam de Castro Machado
Universidade Federal do Ceará - UFC

À memória da minha avó Maria José,
que em vida torceu pelo meu sucesso
e foi parte essencial da minha história.

AGRADECIMENTOS

A Deus por todas as conquistas, oportunidades e desafios que Ele têm me dado a oportunidade de vivenciar.

À minha mãe, Silvaelena, por sua dedicação, apoio e suporte que nunca me faltaram.

À minha irmã, Gizele, por sua amizade, carinho e paciência.

Ao meu noivo, Rafael, pelo companheirismo e apoio total em todos os momentos.

A todos os meus familiares, pela torcida e apoio, em especial à minha tia, Aparecida, por ser uma segunda mãe em minha vida e à minha prima e irmã, Alyne.

Aos amigos Arthur Rodrigues, Carlos Vinícius, Danusa Ribeiro, Márcio Costa, Phablo Moura e Ticiana Linhares, por todos os momentos compartilhados na graduação e no mestrado.

Ao meu orientador, Professor José Antônio Macedo, pelo suporte na pesquisa.

Ao meu co-orientador, Professor Mário Nascimento, pelo suporte na pesquisa, receptividade e acolhimento na Universidade de Alberta.

Aos professores Javam de Castro Machado e Marco Antonio Casanova pela disponibilidade em participar da banca avaliadora e contribuir com seus sólidos conhecimentos para este e para futuros trabalhos.

A todos os colegas e professores do grupo de pesquisa ARIDa.

À CAPES pelo auxílio financeiro que me permitiu dedicação integral ao mestrado e à pesquisa.

"A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original."

(Albert Einstein)

RESUMO

Nesta dissertação foi estudado o problema de processar consultas *k*NN em redes de rodovias considerando o histórico das condições de tráfego, em particular o caso onde a velocidade dos objetos móveis depende do tempo. Dado que um usuário está em uma dada localização e em um determinado instante de tempo, a consulta retorna os k pontos de interesse (por exemplo, postos de gasolina) que podem ser alcançados em uma quantidade de tempo mínima considerando condições históricas de tráfego. Soluções anteriores para consultas *k*NN e outras consultas comuns em redes de rodovia estáticas não funcionam quando o custo das arestas (tempo de viagem) é dependente do tempo. A construção de estratégias e algoritmos eficientes e corretos, e métodos de armazenamento e acesso para o processamento destas consultas é um desafio desde que algumas das propriedades de grafos comumente supostas em estratégias para redes estáticas não se mantêm para redes dependentes do tempo. O método proposto aplica uma busca A^* à medida que vai, de maneira incremental, explorando a rede. O objetivo do método é reduzir o percentual da rede avaliado na busca. Para dar suporte à execução do algoritmo, foi também proposto um método para armazenamento e acesso para redes dependentes do tempo. A construção e a corretude do algoritmo são discutidas e são apresentados resultados experimentais com dados reais e sintéticos que mostram a eficiência da solução.

Palavras-chave: Consultas kNN. Processamento de consultas espaciais. Redes dependentes do tempo.

ABSTRACT

In this dissertation we study the problem of processing k-nearest neighbours (kNN) queries in road networks considering the history of traffic conditions, in particular the case where the speed of moving objects is time-dependent. For instance, given that the user is at a given location at a certain time, the query returns the k points of interest (e.g., gas stations) that can be reached in the minimum amount of time. Previous solutions to answer kNN queries and others common queries in road networks do not work when the moving speed in each road is not constant. Building efficient and correct approaches and algorithms and storage and access schemes for processing these queries is a challenge because graph properties considered in static networks do not hold in the time dependent case. Our approach uses the well-known A* search algorithm by applying incremental network expansion and pruning unpromising vertices. The goal is reduce the percentage of network assessed in the search. To support the algorithm execution, we propose a storage and access method for time-dependent networks. We discuss the design and correctness of our algorithm and present experimental results that show the efficiency and effectiveness of our solution.

Keywords: kNN Queries. Spatial querying process. Time-dependent networks.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo está organizado da seguinte forma. A motivação para este trabalho é apresentada na Seção 1.1. A Seção 1.2 apresenta uma classificação para redes dependentes do tempo. Na Seção 1.3 são apresentados os objetivos gerais e específicos. Por fim, a Seção 1.4 apresenta as contribuições e a Seção 1.5 conclui o capítulo.

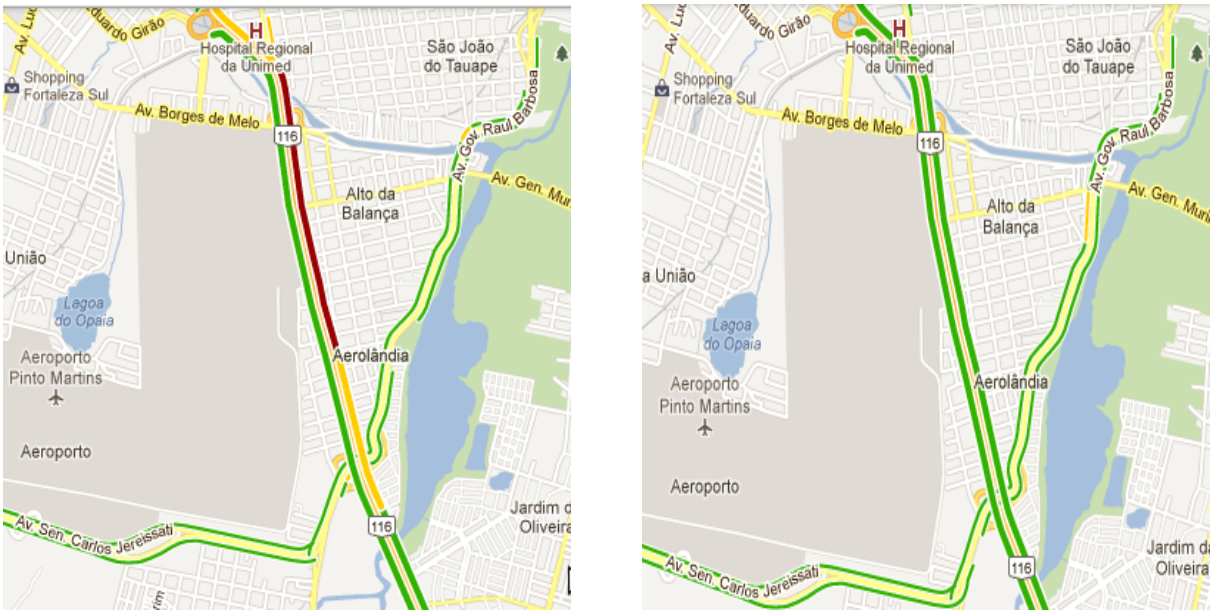
1.1 Motivação

A alta disponibilidade de dispositivos de rastreamento, tais como dispositivos habilitados com GPS, propiciou a expansão dos chamados serviços baseados em localização. Serviços baseados em localização são quaisquer serviços que levem em consideração a posição geográfica de uma entidade (JUNGLAS; WATSON, 2008). Esta mesma disponibilidade possibilita também coletar grande quantidade de dados de mobilidade de veículos, pessoas ou quaisquer outros objetos móveis. Uma série de diferentes e importantes análises pode ser feitas sobre dados de mobilidade, que permitem descobrir, por exemplo, o comportamento e as preferências de indivíduos. No contexto de trânsito, é possível coletar uma grande quantidade de dados de trajetórias de veículos e, a partir dos mesmos, criar uma visão das condições de tráfego no tempo e espaço. A avaliação das condições de tráfego é a chave para propiciar sistemas de transporte inteligentes.

Diversas aplicações, como planejamento de rotas, detecção de menores caminhos, consultas de vizinhos mais próximos e outras; utilizam uma estrutura de rede de rodovias subjacente. A natureza de tais redes é espaço-temporal. Espacial, pois precisam representar o espaço e as restrições espaciais às quais os objetos móveis são submetidos. E temporal, considerando que o tempo de viagem em redes rodoviárias depende diretamente do tráfego e que os padrões de tráfego dependem, por sua vez, não apenas da região em questão, mas também do período do dia. Portanto, o tempo que um objeto móvel leva para atravessar um segmento da via depende tipicamente do instante de tempo de partida. Assim, o tempo gasto para percorrer uma rota em uma rede de rodovias depende do tempo inicial em que se iniciou o percurso. Essa variação temporal do tempo de viagem é fruto das características dinâmicas do sistema causadas, por exemplo, por engarrafamentos em alguns momentos do dia, e de pouco ou nenhum tráfego em certas regiões e horários. A Figura 1.1 (obtida do Google Maps [1]) trás um exemplo real de como o tempo pode influenciar no tráfego, portanto, no tempo gasto no percurso de uma viagem. Ela ilustra dois momentos diferentes de uma mesma via na cidade de Fortaleza. Às 7h30min da manhã (Figura 1.1(a)), o tráfego é mais intenso e, portanto, o tempo para percorrer a via é maior que ao meio-dia (Figura 1.1(b)), quando não há congestionamentos.

Pesquisas recentes têm incluído a dependência temporal para solucionar problemas convencionais de consultas espaciais (GOLDBERG; HARRELSON, 2005) (NANNICINI et al., 2012) (DEMIRYUREK et al., 2011). Entretanto, a maioria dos trabalhos existentes que propõem soluções para consultas espaciais não consideram a variação temporal. Neles, a dis-

[1] https://maps.google.com/



(a) Tráfego às 7h30min. (b) Tráfego às 12h30min.

Figura 1.1: BR-116 em dois diferentes momentos. Do lado esquerdo, às 7h30min da manhã. As cores vermelha e amarela da via representam tráfego intenso. Do lado direito, às 12h30min, a cor verde representa trânsito livre na mesma via.

tância entre dois pontos na rede é fixa e independente do tempo, o que não reflete as condições de tráfego previstas em redes reais.

Neste trabalho, o modelo de rede utilizado é tal que a distância entre dois pontos é dada por uma função do tempo. Estas funções dão o tempo de viagem de uma via da rede em função do instante de tempo em que se inicia um percurso. Desta forma, além das informações espaciais, é possível incorporar à rede as informações sobre os padrões de tráfego em cada via. Construir estratégias e algoritmos para o processamento correto e eficiente de consultas em redes dependentes do tempo é um desafio, desde que as propriedades comuns de grafos podem não ser satisfeitas no caso dependente do tempo (GEORGE; KIM; SHEKHAR, 2007).

Na Seção 1.2, é apresentada uma classificação das redes de transporte (e mobilidade em geral) segundo descrito em (GEORGE; KIM; SHEKHAR, 2007). Esta classificação é importante para que se possa reconhecer exatamente a categoria de rede considerada neste trabalho e suas principais características.

1.2 Classificação de redes

Segundo a classificação de (GEORGE; KIM; SHEKHAR, 2007), as redes são categorizadas de acordo com a natureza da variação temporal dos custos de suas arestas. Esta classificação permite avaliar as diversas aplicações e situações que cada categoria pode modelar. A Figura 1.2 apresenta um diagrama que ilustra esta classificação. Podem-se reconhecer sete categorias distintas (nem todas mutuamente exclusivas): Redes de futuro imprevisível, Re-

des de futuro previsível, Redes estáticas, Redes estacionárias, Redes não-estacionárias, Redes FIFO e Redes não-FIFO. As redes consideradas neste trabalho se enquadram na categoria FIFO. Cada uma destas categorias é explicada e exemplificada a seguir.

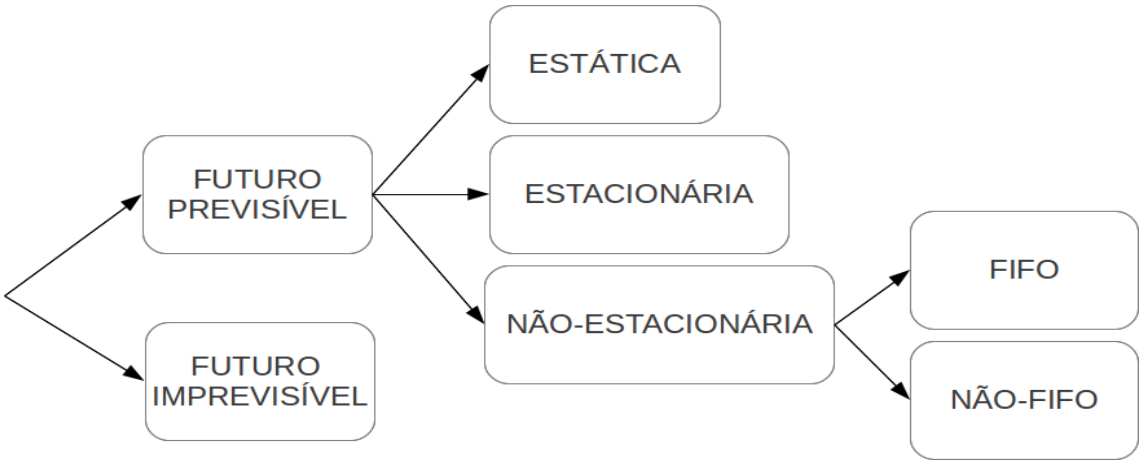


Figura 1.2: Categorias de redes rodoviárias. Fonte (GEORGE; KIM; SHEKHAR, 2007).

Uma rede de *futuro imprevisível* representa ocasiões que, como o próprio nome diz, acontecem sem qualquer previsibilidade. Por exemplo, acidentes em uma determinada via e mudanças no tráfego em decorrência destes acidentes são situações que não podem ser preditas. Assumindo a inexistência de eventos imprevisíveis, as mudanças na rede podem ser descritas por padrões. Estes cenários são ditos de *futuro previsível*. Na verdade, em uma rede que se enquadra nesta categoria, assume-se que as mudanças nos parâmetros e topologia são conhecidas previamente. Um exemplo de uso desta categoria é no planejamento de transporte público. Na ausência de eventos randômicos, estes sistemas podem ser reescalonados desde que as variações ao longo do dia são conhecidas previamente.

As redes de futuro previsível podem ser de três tipos: estáticas, estacionárias ou não-estacionárias. Uma rede *estática* representa uma rede cujos parâmetros e topologia não se modificam com o tempo. Um grafo estático poderia representar uma rede de transporte que não apresenta mudanças significantes. Em uma *rede estacionária* as rotas preferenciais não variam com o tempo, apesar das mudanças nos parâmetros da rede. Uma rede é estacionária, por exemplo, no caso de uma rede em que dirigir em uma estrada de uma origem a um determinado destino é sempre mais rápido que dirigir pelas vias locais. Neste caso, o tempo de viagem pode variar com o tempo, mas as rotas preferenciais não mudam, mesmo com as variações nos parâmetros da rede.

Uma rede tem características *não-estacionárias* quando as rotas mais rápidas mudam com o tempo. Essa mudança ocorre possivelmente devido a diferentes níveis de congestionamento em diferentes partes da rede. Outro cenário possível em redes *não-estacionárias* é o de encontrar o melhor instante de tempo para iniciar uma viagem, este instante seria aquele em que o menor caminho entre dois pontos é o que leva menos tempo durante todo o tempo possível. Com a disponibilidade de dados de tráfego dependente do tempo, proveniente do aumento de redes de sensores para monitorar o tráfego, ou com a disponibilidade de dados de trajetórias

que podem auxiliar a inferir dados de tráfego dependentes do tempo, torna-se importante o desenvolvimento de novos modelos e algoritmos para esta categoria.

As redes *não-estacionárias* dividem-se ainda em *FIFO* e *não-FIFO*. Uma aresta da rede obedece a propriedade FIFO quando um objeto A que começa a percorrer uma aresta antes de um objeto B também finaliza esse percurso (chega à extremidade final da aresta) antes de B. Se todas as arestas da rede obedecem à propriedade FIFO, então a rede é dita FIFO, caso contrário, a rede é não-FIFO. Uma característica das redes FIFO é que a espera em um determinado nó não permite antecipar o tempo de chegada de um veículo. O problema do menor caminho dependente do tempo tem solução polinomial em redes FIFO, entretanto é NP-difícil em redes não-FIFO (ORDA; ROM, 1990).

Neste trabalho, assume-se que as redes utilizadas se enquadram da categoria FIFO. Portanto, são também de futuro previsível e não-estacionárias, o que significa que eventos não previsíveis não são considerados e que o custo para ir de um ponto a outro da rede modifica-se de acordo com o tempo de partida. Nestas condições, as próximas seções descrevem os objetivos e as contribuições deste trabalho.

1.3 Objetivos

1.3.1 Objetivo Geral

Diante do cenário de motivação apresentado, o objetivo geral deste trabalho consiste em analisar e propor estratégias para processamento de consultas de k vizinhos mais próximos (kNN do inglês k nearest neighbours) em redes dependentes do tempo do tipo FIFO.

1.3.2 Objetivos Específicos

Para realizar o objetivo geral estabelecido anteriormente, foram estabelecidos os seguintes objetivos específicos.

- Definir um algoritmo para processar consultas kNN visando reduzir o percentual da rede avaliado para solucionar a consulta;
- Avaliar o algoritmo proposto fazendo uso de dados reais e sintéticos;
- Definir um método de armazenamento e acesso a redes dependentes do tempo para ser utilizado durante a execução do algoritmo de processamento;
- Definir um método para geração de redes dependentes do tempo a partir de dados de trajetórias.

1.4 Contribuições

As contribuições deste trabalho são:

- Definição de um algoritmo baseado na estratégia de busca A* para processamento de consultas *k*NN;
- Definição de um método de acesso que pode ser utilizado tanto pelo algoritmo proposto como por outros algoritmos para acessar informações sobre adjacência e histórico de tráfego em redes dependentes do tempo;
- Proposta de método de geração para construção de redes dependentes do tempo.

1.4.1 Produção científica

Artigo publicado no periódico **Journal of Information and Data Management, 2012** com o título **k-Nearest Neighbors Queries in Time-Dependent Road Networks** (CRUZ; NASCIMENTO; MACÊDO, 2012).

1.5 Conclusão

Neste capítulo foi apresentada a motivação deste trabalho. Mais especificamente, foram discutidas as razões para agregar dependência temporal a consultas comuns em redes estáticas. Também foi apresentada uma classificação das redes de rodovias de acordo com algumas características do tempo de viagem em cada via. Os objetivos gerais e específicos e as contribuições foram descritos. Na sequência, o próximo capítulo define o modelo matemático utilizado para representar as redes dependentes do tempo.

2 MODELO CONCEITUAL PARA REDES DEPENDENTES DO TEMPO

Este capítulo descreve os conceitos e modelo utilizados para a representação das redes dependentes do tempo e algumas de suas propriedades. A Seção 2.1 descreve o modelo de grafo, chamado de grafo dependente do tempo, bem como alguns conceitos necessários para a formulação do problema estudado. A Seção 6.2 descreve o conceito de pontos de interesse e de pontos de consulta e como tais pontos podem ser representados, e uma proposta para incorporar os pontos de interesse à representação da rede. Por fim, a Seção 2.3 descreve uma proposta de um método para a construção das funções de tempo de viagem (ou de tráfego) com base em um conjunto de trajetórias sobre a rede e a Seção 2.4 conclui o capítulo.

2.1 Descrição do modelo de TDG

A estrutura de uma rede pode ser modelada por um grafo onde os vértices representam as junções, pontos iniciais e finais de um segmento de rodovia (por exemplo, uma rua ou uma avenida) e as arestas conectam os vértices (dependendo da aplicação, pontos adicionais podem representar mudanças na curvatura ou na velocidade máxima de um segmento). Neste trabalho, o tempo de viagem é modelado por um grafo dependente do tempo, TDG (do inglês *Time-Dependent Graph*), onde o custo (tempo) para percorrer uma aresta é dado por uma função do instante de tempo de partida. Nesta seção, o conceito de TDG é formalizado. As definições de *travel-time*, *time-dependent fastest path* e *time-dependent distance* também são apresentadas.

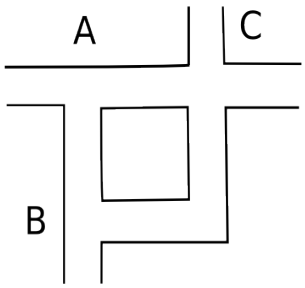
Definição 1. *Um TDG $G = (V,E,C)$ é um grafo onde: (i) $V = \{v_1, \dots v_n\}$ é um conjunto de vértices; (ii) $E = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ é um conjunto de arestas; (iii) $C = \{c_{(v_i,v_j)}(\cdot) \mid (v_i, v_j) \in E\}$, onde $c_{(v_i,v_j)} : [0,T] \to R^+$ é uma função que atribui um peso positivo para (v_i, v_j) dependendo de um instante de tempo $t \in [0,T]$ e onde T é o tamanho do domínio temporal.*

Em termos gerais, um TDG é um grafo onde o custo das arestas varia com o tempo. Para cada aresta (u,v), uma função $c_{(u,v)}(t)$ dá o custo de percorrer a aresta quando o percurso é iniciado no instante de tempo t. O domínio das funções em C é representado como $[0,T]$. Por exemplo, quando $T = 24h$, qualquer função $c_{(u,v)}(t)$ está definida para um dia completo.

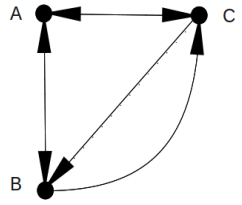
A Definição 1 não restringe ao TDG a ser não-direcionado, de tal forma que a existência de uma aresta (u,v) não implica na existência de (v,u). Além disso, ela permite que arestas opostas, (u,v) e (v,u), possam ser tais que $c_{(u,v)}(t) \neq c_{(v,u)}(t)$.

Para exemplificar, considere a Figura 2.1(a) onde uma rede rodoviária é parcialmente apresentada. A rede pode ser modelada pelo grafo na Figura 2.1(b). O tempo de percurso de cada aresta é dado pelas funções da Figura 2.1(c). Os pares de arestas opostas (B,A) e (A,B), e (A,C) e (C,A), tem o mesmo custo. Entretanto, (B,C) e (C,B), apesar de opostas, têm custos distintos.

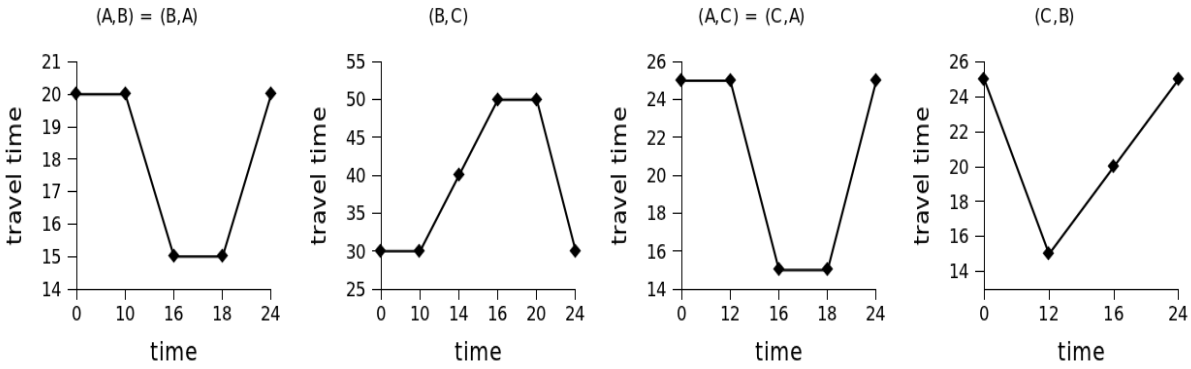
O custo temporal para executar um caminho a partir de um instante de tempo específico é chamado de *travel-time*. O *travel-time* é calculado assumindo que paradas não são



(a) Uma rede rodoviária.



(b) A representação em grafo da rede em (a).



(c) Funções de custo, ou de *tempo de viagem* nas arestas.

Figura 2.1: Exemplo de uma rede rodoviária, o grafo que modela a rede e as funções de *tempo de viagem.*

permitidas, para seu cálculo deve-se calcular o *arrival-time* em cada aresta. Estes conceitos são definidos formalmente a seguir.

Definição 2. *Dado um TDG* $G = (V,E,C)$*, o* ***arrival-time*** *do percurso de uma aresta* $(v_i, v_j) \in E$ *considerando o instante de partida* $t \in [0,T]$ *é dado por* $AT(v_i, v_j, t) = (t + c_{(v_i,v_j)}(t))$ *mod* T*.*

Dado que um veículo percorre um segmento de rodovia e este percurso inicia em um determinado instante de tempo t, o *arrival-time* calcula o instante de tempo no qual o veículo deve chegar ao outro extremo da via, segundo o modelo fornecido pelo TDG. Considerando a Figura 2.1(c), por exemplo, ao percorrer a via representada por (B,C) às 10h00min horas da manhã, o veículo chega às 10h30min, ou seja, $AT(B,C,10:00)$ é calculado para t = 10:00, $c_{(B,C)}(10:00)$ =0:30 e T = 24:00. Note que a operação de resto (mod) existe para que o cálculo seja circular. Para exemplificar, considere o caso em que t = 24:00, logo o custo é dado por $c_{(B,C)}(24:00)$ =0:30, pois o veículo deverá chegar às 0h30min.

Definição 3. *Dado um TDG* $G = (V,E,C)$*, um caminho sobre G,* $p = \langle v_{p_1}, \ldots, v_{p_i}, v_{p_{i+1}}, \ldots, v_{p_k} \rangle$ *em G e um instante de partida* $t \in [0,T]$*, o* ***travel-time*** *de p é o custo dependente do tempo para percorrer o caminho p, dado por* $TT(p,t) = \sum_{i=1}^{k-1} c_{(v_{p_i}, v_{p_{i+1}})}(t_i)$*, onde* $t_1 = t$ *e* $t_{i+1} = AT(v_{p_i}, v_{p_{i+1}}, t_i)$*.*

A definição acima trata de mostrar como o custo de uma rota, *travel-time*, deve ser calculado. A ideia é simples, dada a sequência de arestas do grafo que representam a rota iniciada no instante t, o custo da primeira é calculado com respeito ao instante t. O custo das próximas arestas é dependente do *arrival-time* da aresta anterior. Note que essa definição de custo não leva em consideração paradas nos nós do grafo, ou seja, o percurso da próxima aresta do caminho inicia exatamente no mesmo instante de tempo em que esta foi alcançada.

Definição 4. *Dado um TDG $G = (V,E,C)$, $u,v \in V$ e um instante de partida t, seja $P(v,u)$ o conjunto dos possíveis caminhos entre u e v em G. Um caminho $p \in P(v,u)$ é um* ***time-dependent fastest path*** *de u até v se p é tal que $TT(p,t) \leq TT(p',t)$, para todo $p' \in P(v,u)$. Denota-se por $TDFP(u,v,t)$ a função que retorna um* ***time-dependent fastest path*** *de u até v.*

Definição 5. *Dado um TDG $G = (V,E,C)$, $u,v \in V$ e um instante de partida t, uma* ***time-dependent distance*** *entre u e v no instante de partida t é dado por $TDD(u,v,t) = TT(p,t)$, onde p é um* ***time-dependent fastest path*** *de u até v.*

Retornando ao exemplo das Figuras 2.1(b) e 2.1(c), considere dois caminhos para ir de B até C. Alguém poderia preferir o caminho $\langle B,C\rangle$, que passa por B e vai diretamente à C. Outra possibilidade é o caminho $\langle B,A,C\rangle$, que passa por A. O caminho mais rápido de B a C depende do instante de partida t_s. Para t_s = 10:00, o *travel-time* do caminho $\langle B,C\rangle$ é 30 minutos. O caminho $\langle B,A,C\rangle$, inciado ás 10h00min tem o *travel-time* calculado da seguinte forma. O custo de percorrer às 10:00 é $c_{(B,A)}(10{:}00) = 0{:}20$, o *arrival-time* é 10h20min. O custo de percorrer a próxima aresta (A,C) é dado por $c_{(A,C)}(10{:}20) =$0:45. Portanto, o custo total é de 45 minutos. Similarmente, se t_s = 16:00, então o *travel-time* de $\langle B,C\rangle$ é 50 minutos e de $\langle B,A,C\rangle$ é 30 minutos. Dessa forma, o *time-dependent fastest path* às 10h00min é $TDFP(B,C,10:00) = \langle B,C\rangle$ e a *time-dependent distance* de B até C é igual a $TDD(B,C,10:00)$ = 20 minutos. No entanto, às 16 horas o caminho mais rápido $TDFP(B,C,16:00) = \langle B,A,C\rangle$ e $TDD(B,C,10:00)$=30 minutos.

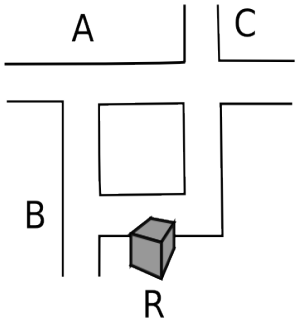
2.2 Representação de Localizações e de Pontos de Interesse na Rede

No modelo utilizado, uma localização como pontos de interesse (ou POIs, do inglês *points of interest*) ou pontos de consulta são representados por um par $p = \langle (u,v), \tau_{(u,v)}\rangle$, tal que (u,v) é uma aresta em G onde p está sobre, $\tau_{(u,v)} = \frac{d(u,p)}{d(u,v)}$ é a distância proporcional de u a p com respeito ao segmento inteiro e (p_i,p_j) é a distância Euclideana entre p_i e p_j, se p_i,p_j estão sobre a mesma aresta, ou indefinido no caso contrário. Note que a utilização da distância Euclideana na definição leva à suposição implícita de que cada mudança de curvatura nos segmentos da rede é mapeada em um novo vértice do TDG. Esta suposição implica também em um potencial aumento no número de vértices para modelar a rede. Entretanto, esta condição não representa uma desvantagem para execução do algoritmo proposto, pois os vértices são recuperados de maneira eficiente pelo método de acesso utilizado.

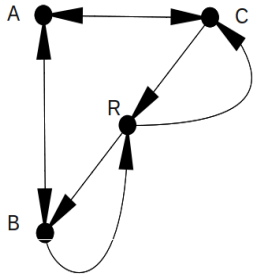
No caso em que a aresta (u,v) tem uma aresta oposta (v,u), uma mesma localização pode ter diferentes representações. Por exemplo, $p = \langle (u,v), \tau_p\rangle$ e $p = \langle (v,u), 1-\tau_p\rangle$, uma

para cada direção. No fundo, se a localização é de um objeto que está se locomovendo, cada representação dá uma diferente direção de locomoção. Foi considerado que apenas uma das representações é dada, desde que uma pode ser obtida a partir da outra.

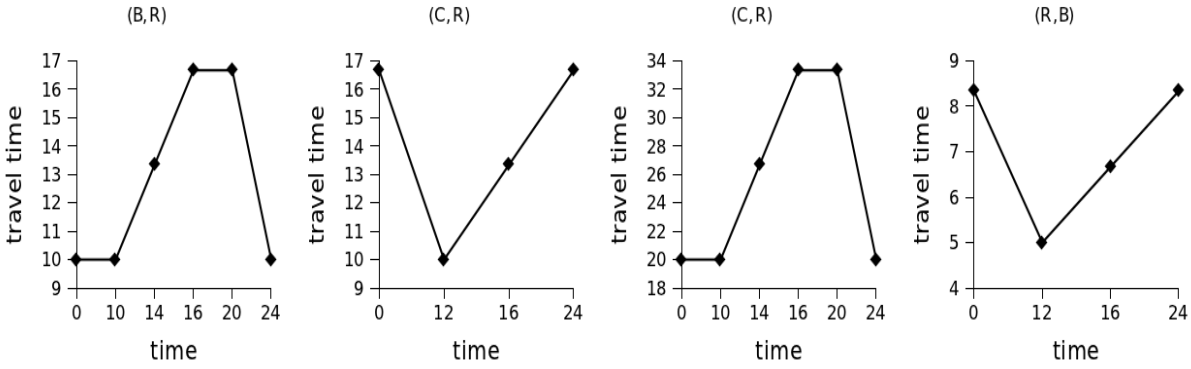
No processo de geração da rede, G pode ser gerado tal que cada POI de um conjunto de POIs S é incluído como um vértice de G. Como na rede original os POIs não são necessariamente vértices, é apresentado um processo para incluir um POI como vértice da rede e calcular as funções de tempo de viagem para sua adjacência. Seja $G = (V,E,C)$ um TDG e S um conjunto de POIs, um dado POI pode ser incluído como um vértice de G. O Algoritmo 2.2 é um procedimento que recebe como entrada um TDG G e um POI $p = \langle (u,v), \tau_p \rangle$, gera um vértice em G equivalente a p e calcula a função de custo para as novas arestas de G que apareceram com a inclusão de p. A Figura 2.2 apresenta um exemplo de inclusão de POI como vértice no TDG da Figura 2.1(b). A Figura 2.2(c) apresenta como a função de tempo de viagem de um mesmo TDG muda de acordo com um novo vértice incluído. Neste exemplo, $R = \langle (B,C), \frac{1}{3} \rangle$ é um ponto de interesse do domínio da aplicação. Note que o tempo de viagem das funções das arestas (A,B) e (C,B), e suas arestas opostas, definidos na Figura 2.1(c) permanecem os mesmos.



(a) Rede e um ponto de interesse R.



(b) TDG considerando um ponto de interesse R.



(c) Função de custo das novas arestas.

Figura 2.2: TDG representando uma rede e seus pontos de interesse.

O algoritmo INCLUDEPOI funciona da seguinte forma. Primeiro, deve-se incluir um vértice R como um vértice no conjunto V. Como R é um ponto sobre (B,C), (B,C) é removido de E e são criadas duas arestas (B,R) e (R,C). As funções de tempo de viagem para

Algorithm 1: IncludePOI

Input: A TDG $G = (V, E, C)$, a POI $p = \langle (u, v), \tau_p \rangle$
Output: $G = (V^{'} = V \cup \{p\}, E^{'}, C^{'})$

$V^{'} \leftarrow V \cup \{p\}$;
$E^{'} \leftarrow E \setminus \{(u,v)\} \cup \{(u,v_p),(v_p,v)\}$;
$c_{(u,p)}(t) \leftarrow \tau_p \times c_{(u,v)}(t)$;
$c_{(p,v)}(t) \leftarrow (1-\tau_p) \times c_{(u,v)}(t)$;
$C^{'} \leftarrow C \setminus \{c_{(u,v)}(t)\} \cup \{c_{(u,p)}(t), c_{(p,v)}(t)\}$;
if $(v,u) \in E$ **then**
    $E^{'} \leftarrow E^{'} \setminus \{(v,u)\}$;
    $c_{(v,p)}(t) \leftarrow (1-\tau_p) \times c_{(v,u)}(t)$;
    $c_{(p,u)}(t) \leftarrow \tau_p \times c_{(v,u)}(t)$;
    $C^{'} \leftarrow C^{'} \setminus \{c_{(v,u)}(t)\} \cup \{c_{(v,p)}(t), c_{(p,u)}(t)\}$;
end
Return $G = (V^{'}, E^{'}, C^{'})$;

(B,R) e (R,C) são $c_{(B,R)} = \frac{1}{3}c_{(B,C)}$ e $c_{(R,C)} = \frac{2}{3}c_{(B,C)}$, respectivamente, e a função de tempo de viagem $c_{(B,C)}$ é removida de C. Agora, tem-se que checar se (C,B) é uma aresta em E. Neste caso, deve-se remover (C,B) de E, $c_{(C,B)}$ de C, as arestas opostas as estas, ((C,R) e (R,B)) e calcular as funções de custo para outra direção. Para calcular as novas funções, considera-se $(1-p_R)$ e $c_{(C,B)}$ ao invés de p_R e $c_{(B,C)}$. As novas funções de custo são $c_{(C,R)} = \frac{2}{3}c_{(C,B)}$ e $c_{(R,B)} = \frac{1}{3}c_{(C,B)}$. Este processo assume que a velocidade de um objeto não muda a partir do momento que ele inicia o percurso de uma aresta até finalizar. Ou seja, a velocidade de um objeto específico é constante em uma aresta.

2.3 Construção das Redes Dependentes do Tempo

Em alguns casos, as informações sobre o tráfego, necessárias para construir as funções de custo da rede, não estão disponíveis. Uma alternativa possível para estimar o tempo de viagem ponto a ponto na rede é inferir este tempo a partir de um conjunto de trajetórias sobre a rede. Construir uma visão do tráfego de toda a rede a partir de um conjunto de trajetórias é um desafio, pois a qualidade, volume e distribuição no tempo e espaço destes dados influenciam fortemente nos resultados obtidos. Neste trabalho, foi desenvolvido um método para estimar a informação de tráfego a partir de um conjunto de trajetórias.

Dados de trajetórias coletados por dispositivos GPS têm dois problemas: (i) a taxa da captura dos dados pode ser baixa e (ii) existe erro nas observações do GPS, que implica que uma informação mais detalhada do movimento exato do objeto é perdida, adicionando incerteza nas rotas. Esta incerteza afeta a efetividade e eficiência de métodos que utilizam estes dados. Uma estratégia comum para reduzir a incerteza é inferir o segmento correto, ou até mesmo local exato das observações. Este problema é conhecido na literatura como *Map-Matching Problem* (CHEN et al., 2011) (LOU et al., 2009) (YUAN et al., 2010) (ZHENG et al., 2012).

O método de construção de rede proposto consiste em, primeiramente, computar

a velocidade média de um segmento da rodovia em um determinado intervalo de tempo, usando os dados de trajetórias. A velocidade média é computada como a média das velocidades das observações das trajetórias que passaram sobre o segmento durante o intervalo de tempo $[t_i, t_{i+1}]$. Considerando que as observações provenientes dos dispositivos GPS são afetadas por erros, utilizar os dados brutos, da maneira que foram coletados, impossibilita saber sobre qual segmento está a posição real das observações.

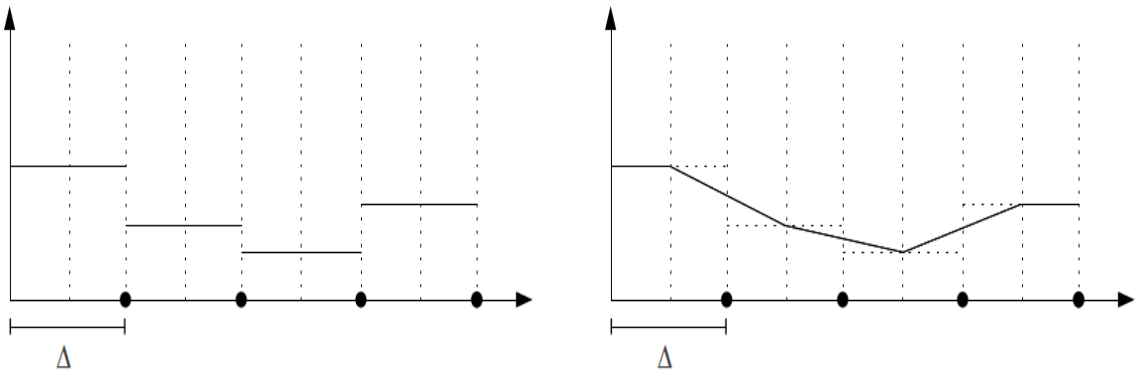
Portanto, para a computação das velocidades, é necessário que seja estabelecida uma estratégia de *map-matching*. O primeiro passo do método proposto consiste em aplicar o algoritmo baseado no modelo gravitacional GMatch (do inglês Gravitational Matching), para estimar a localização das observações nos segmentos de rodovia. No GMatch (CINTRA P. ; TRASARTI, 2013), um ponto ou observação de GPS é atribuído a um subconjunto de segmentos. A estratégia é denominada GMatch, pois define um peso para cada segmento que funciona como uma força de atração que o ponto exerce sobre o segmento. O segmento que é "atraído" com maior força é o mais provável de ser o segmento sobre o qual o ponto de trajetória estava. O GMatch tem como saída, para cada ponto, o segmento atribuído ao ponto (o que gerou maior força de atração) e o peso (força de atração) entre o segmento e o ponto. A partir deste conjunto de atribuições, é possível estimar a velocidade média dos segmentos de rodovia como definido a seguir.

Dado um conjunto de observações de pontos de trajetória $O = \{o_1 \dots o_t\}$, sendo $o_i = (p_i, d_i, s_i)$, onde p_i é a posição espacial, d_i é a direção e s_i a velocidade coletados, um conjunto de segmentos de rodovia $R = \{r_1 \dots r_m\}$ e uma função $\sigma(o_i, R) = (w_{(o_i,rj)}, r_j)$ atribuindo a observação o_i ao segmento $r_j \in R$ com peso $w_{(o_i,rj)}$. Seja O_j o conjunto de observações atribuídas ao segmento rj, é possível estimar a velocidade média do segmento r_j como: $Speed(O, R, r_j) = \frac{\sum_{o_i \in O_j} w_{(o_i,rj)} \cdot o_i.s}{\sum_{o_i \in O_j} w_{(o_i,rj)}}$.

Na segunda etapa do processo, o objetivo é utilizar a estimativa de velocidades para dar suporte à construção das funções do conjunto C de um TDG. Mais especificamente, a velocidade média por segmento dentro de cada intervalo de tempo, na sequência de intervalos que compõem o domínio temporal, será convertida em uma função linear por partes. Uma função linear por partes pode ser vista como uma sequência de seções de linha reta do tipo $ax + b$.

O algoritmo GENERATETRAVELTIME usa o modelo gravitacional para construir um conjunto de trajetórias, as funções necessárias para definir um TDG. Ele recebe um conjunto de pontos de trajetórias P_S, um segmento S, um instante inicial e final, t_0 e t_f respectivamente, e o número de partições temporais n. O procedimento constrói uma função linear por partes para representar o tempo de viagem de acordo com o instante de partida.

Na primeira etapa, o algoritmo divide o espaço temporal em $\frac{n}{2}$ partições. A variável Δ (linha 3) dá o tamanho inicial das partições. Então, para cada partição, o tempo médio de viagem é computado usando o modelo gravitacional utilizado para estimar as velocidades. O resultado é armazenado no vetor *ttAgv* (linha 6). A Figura 2.3(a) mostra um exemplo de formato de função construída nesta etapa.



(a) (b)

Figura 2.3: Do lado esquerdo, um exemplo de uma função construída com o tempo médio de viagem obtido das estimativas de velocidade média. Do lado direito, um resultado da transformação aplicada na segunda etapa para suavizar função.

A próxima etapa suaviza a função construída na primeira etapa. Uma interpolação linear é aplicada do ponto médio de uma partição para o ponto médio da próxima partição. Os valores dos coeficientes angular e linear, a e b, são calculados nas linhas 17 e 18. Cada função linear $tt_i(x) = ax + b$ é uma estimativa de tempo de viagem em s na i-ésima partição temporal. A função de tempo de viagem é formada pela concatenação de todas as funções lineares de um mesma aresta. A Figura 2.3(b) mostra o resultado da transformação aplicada na segunda etapa.

No caso de uma função linear por partes, é bastante simples identificar se a função é ou não FIFO. O Algoritmo 3, CHECKFIFOPROPERTY faz esta verificação. Para checar se a função de tempo de viagem satisfaz a propriedade FIFO, verifica-se se a desigualdade $t_i + f(t_i) \leq t_i + f(t_{i+1})$ é válida quando $t_i \leq t_{i+1}$. CHECKFIFOPROPERTY retorna se $f(.)$ é ou não uma função FIFO, considerando que $f(.)$ é uma função linear por partes. Como cada função que compõe a função de tempo de viagem é uma seção de linha reta do tipo $ax+b$, basta checar a desigualdade considerando t_i como início do intervalo e t_{i+1} como o fim do intervalo da seção.

2.4 Conclusão

Neste capítulo foi apresentada a definição de TDG, utilizada para modelar uma rede dependente do tempo e alguns conceitos que podem ser definidos com base neste modelo. Um algoritmo para inclusão de pontos de interesse como vértices da rede foi apresentado. Além disso, um método proposto para a geração do *travel-time* das arestas do TDG a partir de dados de observações de trajetórias foi apresentado. O próximo capítulo descreve o problema de interesse deste trabalho, bem como o algoritmo proposto como solução.

Algorithm 2: GENERATETRAVELTIME

Input:
A set of trajectory points P_S;
A segment $S_{(v_i,v_j)}$;
The initial global time t_0;
The final global time t_f;
The number of temporal partitions $n+1$
Output:
The travel-time function $c_{(v_i,v_j)}$

/*First Step*/
$t \leftarrow t_0$;
$\Delta \leftarrow \frac{t_f - t_0}{n}$;
$i \leftarrow 0$;
while $t + \Delta < t_f$ **do**
    $ttAvg(i) \leftarrow \frac{|S_{(v_i,v_j)}|}{SpeedEstimation(S,P_S,t,t+\Delta)}$;
    $departime(i) \leftarrow t$;
    $i \leftarrow i$;
    $t \leftarrow t + \Delta$;
end
/*Second Step*/
$i \leftarrow 0$;
$tt(i).a \leftarrow ttAvg(i)$;
$tt_i.b \leftarrow 0$;
$tt_i.time \leftarrow [t_0, t_0 + \frac{\Delta}{2})$;
while $i < n$ **do**
    $tt_i.a \leftarrow \frac{departime(i) - \frac{\Delta}{2} - departime(i)}{ttAvg(i-1) - ttAvg(i)}$;
    $tt_i.b \leftarrow ttAvg(i) - \frac{departime(i) - \frac{\Delta}{2} - departime(i-1)}{ttAvg(i+2) - ttAvg(i)}$;
    $tt_i.intTime \leftarrow [departime(i) - \frac{\Delta}{2}, departime(i) + \frac{\Delta}{2})$;
    $i \leftarrow i + 1$;
end
$tt_i.a \leftarrow ttAvg(i-1)$;
$tt_i.b \leftarrow 0$;
$tt_i.time \leftarrow [departime(i-1) + \frac{\Delta}{2}, departime(i-1) + \Delta]$;
return tt;

Algorithm 3: CHECKFIFOPROPERTY

Input:
A piecewise linear function $tt(.)$;
The number of temporal partitions n
Output:
$true$ if $tt(.)$ is a FIFO function and $false$, otherwise

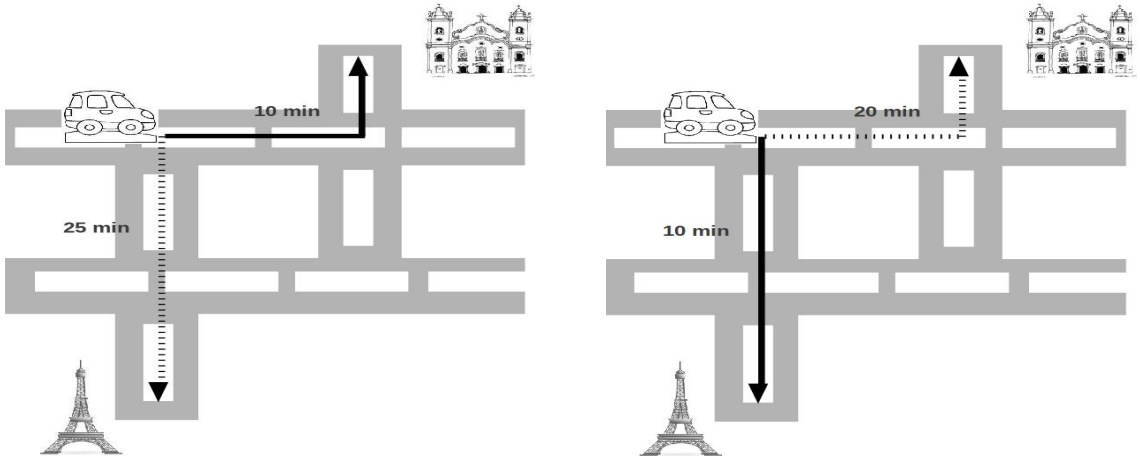
```
1 i ← 0;
2 while i < n do
3     [t_start, t_end] ← tt_i.time;
4     if tt_i.a × t_start + tt_i.b > tt_{i+1}.a × t_end + tt_{i+1}.b then
5         return false;
6     end
7 end
8 return true;
```

3 CONSULTAS *K*NN EM REDES DEPENDENTES DO TEMPO

Este capítulo descreve o problema de interesse deste trabalho e a solução proposta para esse problema. O capítulo está organizado da seguinte forma. O problema do processamento de consultas *k*NN em redes dependentes do tempo é descrito na Seção 3.1. O algoritmo proposto é apresentado na Seção 3.2, para isso são apresentas algumas definições necessárias para a formalização do método proposto. Em seguida, o algoritmo TD-NE-A* é apresentado formalmente, além de uma discussão sobre sua corretude. Por fim, a Seção 3.3 apresenta um exemplo de execução do algoritmo e a Seção 3.4 conclui o capítulo.

3.1 Consultas TD-*k*NN

Neste trabalho, foi considerada a dependência temporal para responder a consultas *k*NN. Uma consulta *k*NN recupera o conjunto de *k* pontos de interesse que são mais próximos a um ponto de consulta. Nas redes dependentes do tempo, uma consulta *k*NN retorna o conjunto de *k* pontos de interesse com o mínimo tempo de viagem do ponto de consulta, considerando um instante de partida. Como exemplo, o cenário ilustrado na Figura 3.1. Imagine um turista em Paris, interessado em visitar alguma atração turística próxima a ele. Considere duas atrações turísticas na cidade, a Torre Eiffel e a Catedral de Notre Dame. O turista gostaria de saber qual destes pontos turísticos tem um caminho do ponto onde ele está em determinado instante, que é o mais rápido entre os possíveis caminhos considerando as condições de tráfego durante o percurso. Por exemplo, caso a consulta seja processada às 10 horas da manhã ele levaria 20 minutos para ir para catedral, sendo esta a atração mais próxima. Se ao invés disso, a consulta for submetida às 22 horas, no mesmo ponto de partida, a atração mais próxima seria a Torre Eiffel.



(a) Ponto de interesse mais próximo às 10*h*. (b) Ponto de interesse mais próximo às 22*h*.

Figura 3.1: Um exemplo de aplicação para consultas *k*NN dependentes do tempo. O caminho mais rápido do ponto de consulta para o vizinho mais próximo é marcado pelas linhas sólidas. O caminho mais rápido para o outro ponto de interesse está em linhas pontilhadas.

Soluções anteriores para o problema de menor caminho, para o processamento de

consultas *k*NN e outras consultas comuns em redes de rodovia estáticas não funcionam quando o custo das arestas (tempo de viagem) é dependente do tempo. A construção de estratégias e algoritmos eficientes e corretos para o processamento destas consultas é um desafio desde que algumas das propriedades de grafos comumente supostas em estratégias para redes estáticas, não mantêm-se para redes dependentes do tempo.

O problema é descrito formalmente a seguir.

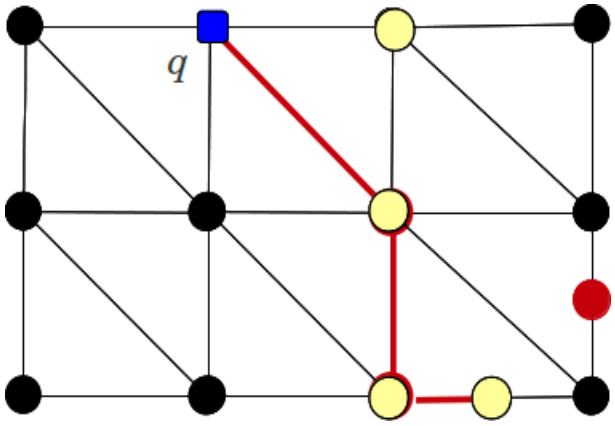
Definição 6. *Seja $G = (V,E,C)$ um TDG e $POI \subseteq V$ um sub-conjunto de vértices de G identificados como pontos de interesse de G. Dado um ponto de consulta q, um instante de tempo de partida t e um valor inteiro positivo $k > 0$, uma consulta kNN dependente do tempo (ou TD-kNN, do inglês Time-Dependent kNN) retorna um conjunto $R = \{v_{r_1}, \ldots v_{r_k}\} \subseteq POI$ tal que $\forall v \in POI \setminus R, TDD(q, v_{r_i}, t) \leq TDD(q, v, t)$.*

3.2 Algoritmo TD-NE-A*

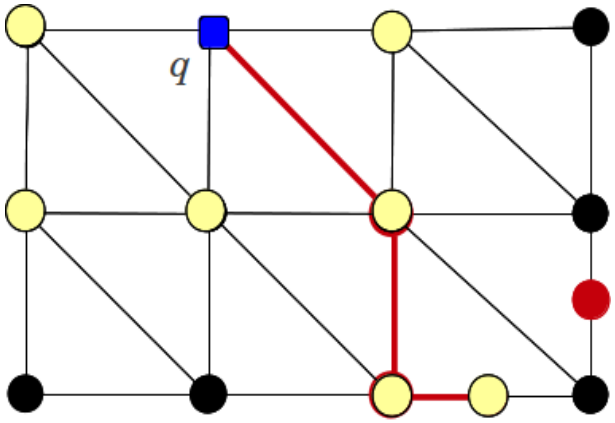
Nesta seção, é descrito o algoritmo para processar consultas TD-*k*NN proposto neste trabalho. Tal algoritmo é baseado no algoritmo de expansão incremental da rede (INE, do inglês Incremental Network Expansion para processar consultas *k*NN em redes estáticas, proposto originalmente em (PAPADIAS et al., 2003). O INE é um algoritmo baseado no algoritmo de Dijkstra (DIJKSTRA, 1959) que, iniciando de um ponto de consulta q, visita todos os vértices alcançáveis de q em ordem de sua proximidade, até que todos os k pontos de interesse mais próximos sejam localizados.

A solução proposta incorpora uma busca A* (GOLDBERG; HARRELSON, 2005) diretamente na expansão realizada pelo INE (deve-se a isso o nome TD-NE-A*). Para solucionar o problema de menor caminho, por exemplo, a busca A* é incorporada na execução do algoritmo de Dijkstra, de tal forma que a execução do algoritmo é guiada, sendo possível descartar a verificação de caminhos que não levam à solução. Para isso utiliza-se a distância $d(v_i, v_k)$ mais uma função heurística (ou de potencial) $h(v_k)$ para determinar a ordem na qual os vértices são expandidos na árvore de busca. Assim, os vértices que têm o maior potencial para levar à solução têm preferência para serem avaliados. A distância atual, $d(v_i, v_k)$, mais a função heurística de um vértice v_k, $h(v_k)$, é uma estimativa de custo do caminho partindo de v_i até o objetivo, passando por v_k. Para que a busca A* possa ser utilizada, a $h(v_k)$ deve ser uma heurística admissível; isto é, que não superestima a distância de v_k até o objetivo. Quando $h(v_k)$ satisfaz a condição $h(v_k) \leq d(v_k, v_j) + h(v_j)$, para toda aresta (v_k, v_j) do grafo, $h(v_k)$ é dita monotônica, ou consistente. Em tais casos, a busca A* pode ser implementada mais eficientemente porque soluciona o problema de maneira incremental construindo caminhos sem retornar a etapas anteriores.

Na solução proposta, a busca A* é incorporada diretamente na expansão incremental da rede, ao invés de ser utilizada para calcular cada distância (dependente do tempo) do ponto de consulta q para cada ponto de interesse candidato. Portanto, a busca executada no algoritmo proposto, diferentemente das soluções utilizadas no cálculo do menor caminho, tem múltiplos e desconhecidos objetivos. A estratégia incremental evita re-computar custos de cam-



(a) Expansão da rede com busca A*



(b) Expansão incremental.

Figura 3.2: Exemplo da diferença nos vértices avaliados pelo algoritmo proposto e a expansão incremental.

inhos previamente computados e visita apenas as arestas necessárias uma única vez. No algoritmo TDNE-A*, a função heurística $h(.)$ adiciona a cada vértice uma estimativa do potencial de fazer parte do caminho que leva o ponto de consulta q a um ponto de interesse mais próximo. A ideia dessa solução é evitar expandir vértices em um caminho que é rápido, entretanto não leva a um ponto de interesse próximo. A motivação por detrás disso é o fato de um vértice u poder ser alcançado rapidamente a partir do ponto de consulta q, mas não necessariamente o caminho que passa por u é o menor caminho que leva ao próximo ponto de interesse mais próximo.

A Figura 3.2 exemplifica a diferença entre a execução de um algoritmo de expansão com busca A* e o algoritmo de expansão incremental que realiza uma busca cega. Os vértices em vermelho são os pontos de interesse, o quadrado em azul é o ponto de consulta, em amarelo estão os vértices que foram acessados pela busca. A Figura 3.2(b) apresenta uma ilustração para a expansão incremental. Na Figura 3.2(a), é apresentado o que seria a execução do algoritmo proposto. Veja que na Figura 3.2(a) uma quantidade menor de vértices está em amarelo se comparada à Figura 3.2(b), ou seja, uma menor quantidade de vértices é expandida.

A seguir, são descritos alguns conceitos necessários para definição da função de potencial da estratégia proposta.

O *lower bound graph* de um TDG $G = (V, E, C)$ é um grafo $\underline{G} = (V, E, \underline{C})$ onde V e E são o mesmo conjunto de vértices e arestas do grafo original G e $\underline{C}$ é um conjunto de custos $\underline{c}_{(v_i,v_j)} = min_{t\in[0,T]}\{c_{(v_i,v_j)}(t)\}$, para todo $c_{(v_i,v_j)} \in C$. Similarmente, um *upper bound graph* $\overline{G} = (V, E, \overline{C})$ tem o mesmo conjunto de vértices e arestas do grafo original G e $\overline{C}$ é um conjunto de custos $\overline{c}_{(v_i,v_j)} = max_{t\in[0,T]}\{c_{(v_i,v_j)}(t)\}$, para todo $c_{(v_i,v_j)} \in C$.

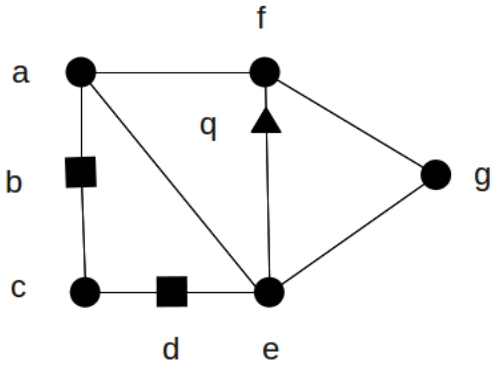


Figura 3.3: Estrutura em grafo de uma rede de rodovias. Os vértices da rede são representados pelos círculos, os vértices que são pontos de interesse por quadrados. O triângulo representa um ponto de consulta.

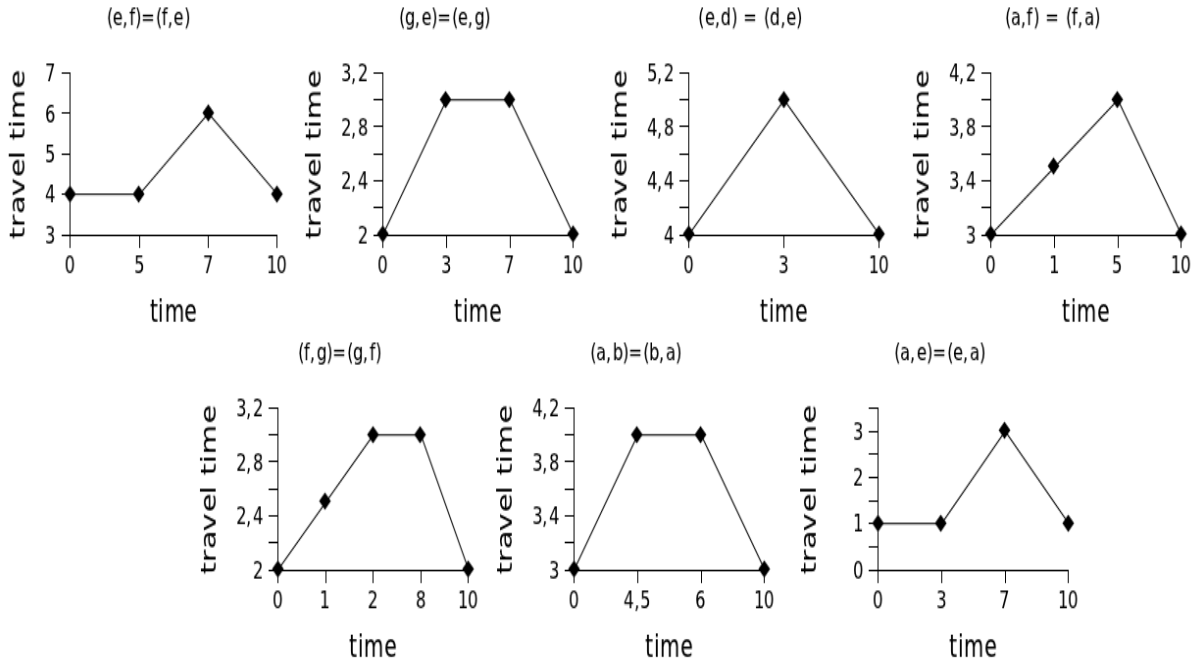
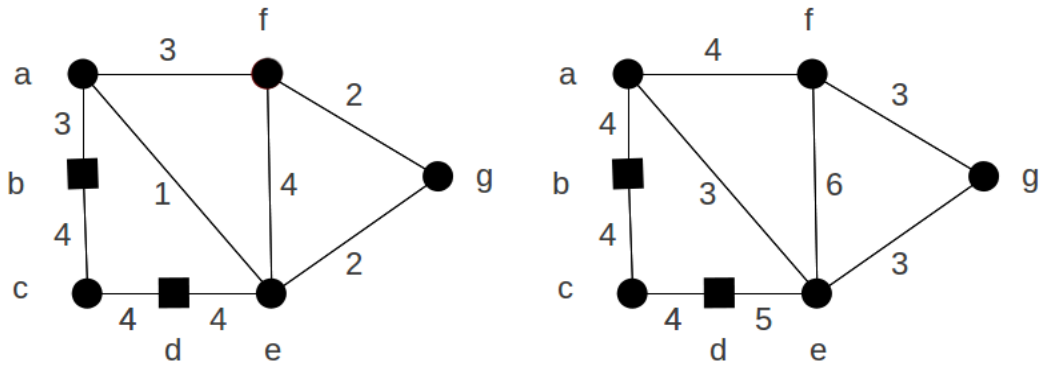


Figura 3.4: Funções representando o custo (tempo de viagem) para cada aresta em 3.3.

Em termos gerais, o *lower bound graph* e o *upper bound graph*, ou simplesmente $\underline{G}$ e $\overline{G}$, apresentam visões de situações "especiais" da rede. O $\underline{G}$ representa todos os momentos onde o congestionamento é o mínimo, enquanto o $\overline{G}$ apresenta todos os momentos onde o congestionamento é o máximo , para cada via. No $\underline{G}$ os custos de cada aresta é o menor custo observado durante todo o período representado pelo TDG, assim como no $\overline{G}$ os custos são os maiores custos observados. As Figuras 3.5(a) e 3.5(b) representam, respectivamente o $\underline{G}$ e o $\overline{G}$

do TDG na Figura 3.3. Note que, o $\underline{G}$ e o $\overline{G}$ são grafos estáticos, ou seja, os custos das arestas têm valor constante. Esse valor é dado pelos pontos de mínimo e máximo das funções de custo das arestas.

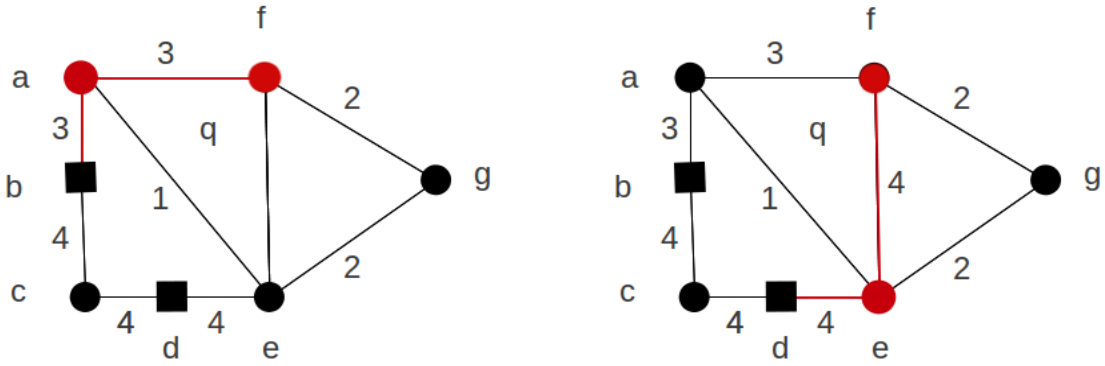


(a) Grafo de limite inferior.

(b) Grafo de limite superior.

Figura 3.5: Exemplo dos grafos de limites do grafo da Figura 3.3 de acordo com funções de custo da Figura 3.4.

Os conceitos de $LTDD(v_i, v_j)$ e $UTDD(v_i, v_j)$ foram definidos como o tempo de viagem do caminho mais rápido entre v_i e v_j em $\underline{G}$ e $\overline{G}$, respectivamente. Como os custos das funções em $\underline{G}$ e $\overline{G}$ são constantes, $LTDD(v_i, v_j)$ e $UTDD(v_i, v_j)$ não dependem de um tempo de partida. A Figura 3.6 auxilia a ilustrar conceito de LTDD. Nesta figura pode-se visualizar o $\underline{G}$ do grafo da Figura 3.3. O valor de $LTDD(f, b)$, por exemplo, é dado pelo custo do menor caminho (destacado em vermelho), entre f e b no $\underline{G}$.



(a) Caminho mais rápido de f até a em $\underline{G}$. Custo do caminho tem é o valor de $LTDD(f, b)$.

(b) Caminho mais rápido de f até d em $\overline{G}$. Custo do caminho tem é o valor de $LTDD(f, d)$.

Figura 3.6: Vizinhos mais próximos de f, usados para calcular $H(f)$ e $UTT(f)$, respectivamente.

3.2.1 Formalização da Função de Potencial Utilizada

Na solução proposta, para a execução da busca A*, a função heurística (ou de potencial) de um vértice u, $H(u)$ é igual ao tempo de viagem de u para o ponto de interesse mais próximo em $\underline{G}$. Este é um valor otimista, desde que é uma estimativa para a viagem mais rápida

que leva u a algum ponto de interesse. Note que, na realidade, poderá nunca existir um caminho entre u e o ponto de interesse utilizado para o cálculo do valor de $H(u)$, desde que os custos utilizados no cálculo do custo do caminho podem eventualmente, serem obtidos de instantes diversos do domínio temporal. Além disso, é uma heurística admissível e consistente, como mostradas nos lemas as seguir. Portanto, $H(u)$ pode ser utilizada no algoritmo de busca A*. A Figura 3.6 apresenta os menores caminhos de f para os pontos de interesse de $\underline{G}$, b (Figura 3.6(a)) e d (Figura 3.6(b)). O valor do menor caminho de f até b é $LTDD(f,b) = 6$ e de f até d é $LTDD(f,d) = 8$. Como o menor valor de distância entre f e todos os outros pontos de interesse é 6, então $H(u) = 6$.

Lema 1. *$H(\cdot)$ é uma heurística admissível.*

Demonstração. Para mostrar que $H(\cdot)$ é admissível, considere um ponto de consulta $q = \langle (u,v), \tau_q \rangle$ e seja nn_q o ponto de interesse mais próximo a q no tempo de partida t in G. Seja u um vértice que já foi visitado pela busca e nn_u o ponto de interesse mais próximo a u em $\underline{G}$. Suponha por contradição que $H(u)$ não é admissível, ou seja

$$\begin{gathered} TDD(q,u,t) + H(u) = \\ TDD(q,u,t) + LTDD(u,nn_u) > \\ TDD(q,u,t) + TDD(u,nn_q,AT(q,u,t)). \end{gathered}$$

A desigualdade $LTDD(u,nn_u) > TDD(u,nn_q,AT(q,u,t))$ é uma contradição contra a hipótese de que nn_u é o ponto de interesse mais próximo a u, pois $LTDD(u,nn_u)$ é um limite inferior para todos os possíveis tempos de viagem de u para qualquer ponto de interesse em qualquer instante de partida. Pode-se concluir, portanto, que $H(u)$ não superestima a distância para o próximo ponto de interesse mais próximo. Portanto, $H(.)$ é admissível. □

Lema 2. *$H(\cdot)$ é uma heurística consistente.*

Demonstração. Dado um TDG $G = (V,E,C)$ e um conjunto de pontos de interesse $S \subseteq V$, para mostrar que $H(.)$ é consistente para qualquer tempo de partida t com respeito a S, precisa-se mostrar que

$$H(u) \leq c_{(u,v)}(t) + H(v)$$

para qualquer aresta $(u,v) \in E$.
Seja nn_u o ponto de interesse mais próximo a u em $\underline{G}$. Suponha por contradição que

$$H(u) > c_{(u,v)}(t) + H(v).$$

Portanto,

$$\begin{gathered} LTDD(u,nn_u) = \\ H(u) > c_{(u,v)}(t) + H(v) > \\ \underline{c}_{(u,v)} + H(v) = \\ LTDD(u,p) \end{gathered}$$

para algum vértice $p \in S$.
Neste caso, p é um ponto de interesse mais próximo de u do que nn_u, uma contradição contra a hipótese de que nn_u seria o ponto de interesse mais próximo de u. Portanto, $H(.)$ é admissível. □

3.2.2 Etapa de Pré-processamento

A solução proposta considera duas etapas de processamento, executadas uma única vez de maneira *offline*. Em ambas as etapas, um algoritmo para encontrar o ponto de interesse mais próximo em redes não dependentes do tempo é executado.

A primeira etapa de pré-processamento calcula o valor da função heurística para os vértices de G. Para cada vértice v, a distância entre ele e o ponto de interesse mais próximo em $\underline{G}$ é calculada. Esta distância é atribuída a $H(v)$ para ser utilizada como a função de potencial. A segunda etapa computa o vizinho mais próximo de v em $\overline{G}$, denotado por $UNN(v)$, e a distância de v até $UNN(v)$ em $\overline{G}$ é denotada por $UTDD(v, UNN(v))$. Os valores $UNN(v)$ e $UTDD(v, UNN(v))$ são utilizados para podar vértices que, certamente, levam a pontos de interesse que estão mais distantes que um determinado conjunto de candidatos.

A etapa de pré-processamento tem a importante vantagem de poder utilizar soluções familiares que executam em tempo polinomial sobre redes não dependentes do tempo (estáticas). Para executá-las, foi utilizado o algoritmo proposto em (PAPADIAS et al., 2003). Cada passo do pré-processamento faz uma chamada ao algoritmo que encontra o 1NN de cada vértice. No pior caso, o algoritmo executa em tempo $O(|V||E| + |V|^2 \log |V|)$. Entretanto, uma melhoria pode ser feita neste processo. O primeiro ponto de interesse mais próximo o de v é também ponto de interesse mais próximo de todo vértice que está no caminho de v até o. Como demonstrado no seguinte lema.

Lema 3. *Seja v um vértice de $\underline{G}$, o o ponto de interesse mais próximo de v. Seja $P = \langle v = u_0, u_1, \ldots, u_i, u_{i+1}, \ldots, u_m = o \rangle$ o menor caminho de v até o. Então o ó ponto de interesse mais próximo de u_j, para todo $u \in P$.*

Demonstração. Suponha $u_j \in P$, tal que o não é o ponto de interesse mais próximo a u_j. Seja o' o ponto de interesse mais próximo a u_j, ou seja, $LTDD(u_j, o') < LTDD(u_j, o)$. O custo do caminho P é dado por $LTDD(u_0, o) = LTDD(u_0, u_j) + LTDD(u_{j+1}, o)$. Portanto, $LTDD(u_0, o) = LTDD(u_0, u_j) + LTDD(u_{j+1}, o) > LTDD(u_0, u_j) + LTDD(u_{j+1}, o') = LTDD(u_0, o')$. Logo, o caminho de v a o' tem um custo menor que P. Um absurdo contra a hipótese de que o é o ponto de interesse mais próximo a v. Portanto, o também é o ponto de interesse mais próximo de u_j em $\underline{G}$. □

Portanto, a busca a partir de v pode ser aproveitada para parte dos vértices da rede, acelerando o tempo de execução do pré-processamento.

3.2.3 Atualizações nos Dados

Em caso de atualizações nos dados históricos, a informação pré-computada deve ser alterada. Para alteração são considerados três casos: atualizações nas funções de tráfego; exclusão de um objeto POI; inclusão de um objeto POI.

Atualizações nas funções de tempo de viagem.

Quando o valor da função de tempo de viagem é atualizado, a etapa de pré-processamento sobre o grafo $\underline{G}$ deverá ser executada novamente apenas quando o valor de máximo de alguma função é modificado. Da mesma forma, a etapa de pré-processamento sobre o grafo $\overline{G}$ deverá ser novamente executada apenas se o valor de mínimo de alguma função de tempo de viagem se modifica.

Exclusão de um POI.

Quando um POI é excluído da rede, deve-se executar novamente a etapa de pré-processamento apenas para os vértices que têm o objeto excluído como um ponto de interesse mais próximo. Se o objeto é ponto de interesse mais próximo de um vértice em $\underline{G}$, uma nova busca a partir deste vértice em $\underline{G}$ deve ser iniciada. O mesmo ocorre com relação a $\overline{G}$.

Inclusão de um POI.

Quando um novo ponto de interesse é incluído deve-se executar toda a etapa de processamento novamente.

3.2.4 Processamento da Consulta

O Algoritmo 3.3.2 apresenta o pseudo código de TD-NE-A* e funciona da seguinte forma. Ele recebe três parâmetros como entrada, o ponto de consulta $q = \langle (u,v), \tau_q \rangle$, que representa a localização do objeto que irá visitar os pontos de interesse, o número k de pontos de interesse mais próximos que se deseja recuperar e o ponto de partida t.

De modo geral, ele executa de maneira similar às soluções de expansão incremental. Os vértices do grafo têm três estados distintos: *não-visitado*, *visitado* e *avaliado*. Entretanto, duas estratégias para guiar a busca e evitar acessos desnecessários a vértices não-promissores são incluídas no algoritmo de expansão. O algoritmo inicia obtendo a aresta (u,v) que representa o segmento de rodovia no qual q se encontra. Os vértices u e v são inseridos numa fila de prioridade Q. Os vértices em Q são ditos visitados, os vértices que nunca foram incluídos em Q são ditos não-visitados. Todo vértice incluído em Q é um candidato à expansão na próxima etapa. Uma entrada de Q é uma tupla $(v_i, AT_{v_i}, TT_{v_i}, L_{v_i})$, onde:

- $TT_{v_i} = TT(q, v_i, t)$ representa o menor tempo de viagem calculado para chegar em v_i a partir de q;
- $AT_{v_i} = AT(q, v_i, t)$ representa o instante de chegada do menor caminho visto até o momento para ir de q a v_i;

- L_{v_i} é dado pela soma $TT_{v_i} + H(v_i)$ e representa uma estimativa otimista para o tempo de viagem de q até um ponto de interesse através de um caminho que passa por v_i.

As entradas em Q são ordenadas em ordem não-decrescente dos valores L_{v_i}. Isso significa que, diferente das estratégias de expansão incremental anteriores, os vértices não são priorizados em função da distância do ponto inicial q até eles, mas sim em função da expectativa de encontrar um ponto de interesse mais próximo a partir daquele vértice. A motivação por trás dessa estratégia é avaliar primeiramente aqueles vértices que têm uma maior chance de fazer parte de um menor caminho até um ponto de interesse mais próximo. Quando um vértice é removido de Q o seu estado muda para avaliado.

Uma outra fila de prioridade, denominada Q_U, é mantida para armazenar informações sobre pontos de interesse candidatos e sobre os limites superiores do tempo de viagem até eles. Para cada vértice v avaliado, o valor $UTDD(v, UNN(v))$ é obtido. $UNN(v)$ é o ponto de interesse mais próximo a v em $\overline{G}$, portanto $UTDD(v, UNN(v))$ é um expectativa pessimista do tempo de viagem do caminho que passa por v e vai até um ponto de interesse mais próximo a v, representando um limite superior para qualquer caminho que leva a um ponto de interesse mais próximo passando por v.

Observe que um mesmo vértice u pode ser tal que $u = UNN(u_1) = \ldots UNN(u_i) = UNN(u_{i+1}) = \ldots = UNN(u_m)$. Ou seja, u pode ser um ponto de interesse mais próximo a diversos vértices de $\overline{G}$. O valor mantido em Q_U é o mínimo entre $UTDD(u_1, UNN(u_1))$, $\ldots, UTDD(u_i, UNN(u_i))$, $UTDD(u_{i+1}, UNN(u_{i+1}))$, $\ldots$, $UTDD(u_m, UNN(u_m))$. Ou seja, o menor limite superior para o tempo de viagem até u. Q_U é ordenada em ordem não-decrescente dos valores de $UTDD(v, UNN(v))$ e é utilizada para o processo de poda. Mais especificamente, antes de um vértice v ser incluído em Q, o valor L_v é comparado ao k-ésimo elemento de Q_U. Este elemento é o k-ésimo candidato com o menor limite superior de distância. Se L_v é maior que o k-ésimo limite superior em Q_U, significa que a melhor estimativa para encontrar um ponto de interesse a partir de v não é melhor que a k-ésima pior estimativa já encontrada. Ou seja, pelo menos k vértices visitados levam a pontos de interesse mais próximos que os pontos de interesse no caminho passando por v. Portanto, v pode ser descartado.

Quando v é avaliado, checa-se se o vértice $UNN(v)$ já foi inserido em Q_U. Então, verifica-se se $UTDD(v, UNN(v))$ é menor que o valor corrente de limite superior atribuído a ele. Neste caso, o limite superior e a aposição de $UNN(v)$ são atualizadas em Q_U.

3.2.5 Análise do Algoritmo TD-NE-A*

A manutenção das fila de prioridade utilizadas no algoritmo TD-NE-A* é feita por operações de remoção, inserção e reordenação das entradas destas filas. O custo dessa operações é descrito como segue. Cada inserção ou remoção em Q custa $O(\log |V|)$, no pior caso. Além disso, cada atualização na fila de prioridade requer que a fila seja reordenada. Cada reordenação custa $O(|V|)$. Da mesma forma, uma operação de inserção em Q_U, requer $O(\log |POI|)$ operações e a reordenação requer $O(|POI|)$ operações, desde que apenas pontos de interesse são inseridos em Q_U.

Faz-se necessário recordar que o tempo de processamento é dado principalmente pelas operações de E/S, executadas para obter informações sobre adjacência dos vértices. Desde que as operações executadas para manter as filas são feitas em memória, e em tempo polinomial, elas não representam um gargalo no nosso método.

A seguir, a corretude do algoritmo proposto é provada.

Teorema 1. *Seja $S_{NN} = \{v_{NN_1}, \ldots, v_{NN_k}\}$ o conjunto de pontos de interesse retornados pelo algoritmo TD-NE-A*(q,k,t). S_{NN} é o conjunto de k pontos de interesse mais próximos a partir de $q = (a,b,\tau_q)$ no instante de tempo t.*

Demonstração. Para provar o teorema, é suficiente mostrar que quando um ponto de interesse r é removido da fila de prioridade Q:

1. O custo do caminho percorrido de q até r, representado por T_r, tem o mesmo valor de tempo de viagem do caminho mais rápido a partir de q;
2. Não existe ponto de interesse que tenha um caminho a partir de q até ele mais rápido que $TDD(q,r,t)$.

Os itens 1 e 2 são demonstrados a seguir.

1. Por indução no número de vértices avaliados. O caso base é o primeiro vértice removido de Q; este caso é trivial.

   Agora, suponha que para o i-ésimo vértice a afirmação é verdadeira, para todo $1 \leq i \leq l-1$. Seja v o l-ésimo vértice avaliado.

   Suponha que $T_v > TDD(q,v,t)$, portanto v não foi alcançado pelo caminho mais rápido. Seja z o último vértice avaliado que está no caminho mais rápido de q para v. Seja w o próximo vértice no caminho de q para v. Tem-se que

$$\begin{gathered} L_w = TDD(q,z,t) + c_{(z,w)}(AT(q,z,t)) + H(w) = \\ TDD(q,w,t) + H(w) < \\ TDD(q,w,t) + TDD(w,v,AT(q,v,t)) + H(v) < \\ TDD(q,u,t) + TDD(u,v,AT(q,u,t)) + H(v) = L_v. \end{gathered}$$

   Portanto $L_w < L_v$. Entretanto, neste caso w teria sido removido antes de v. Uma contradição contra a hipótese de que z foi o último vértice no caminho mais rápido até v que foi avaliado. Portanto, v foi encontrado pelo caminho mais rápido e $T_v = TDD(q,v,t)$.

2. Suponha por contradição que o ponto de interesse r foi removido da fila Q e existe um outro ponto de interesse r^* tal que $TDD(q,r^*,t) < TDD(q,r,t)$. Seja v o último vértice visitado que pertence ao caminho mais rápido partindo de q até r^*. De acordo com (1) $T_r = TDD(q,r,t)$.

   Como r foi removido antes de v, então pode-se afirmar que

$$L_r = TDD(q,r,t) \leq$$
$$T_v + H(v) =$$
$$TDD(q,u,t) + TDD(u,v,AT(q,u,t)) + H(v),$$

onde u é o vértice removido no momento em que v é visitado. Portanto,

$$L_r \leq TDD(q,u,t) + c_{(u,v)}(AT(q,u,t)) + H(v) =$$
$$TDD(q,v,t) + H(v) \leq$$
$$TDD(q,v,t) + TDD(v,r^*,AT(q,v,t)) =$$
$$TDD(q,r^*,t).$$

Logo, $L_r \leq TDD(q,r^*,t)$, violando a hipótese de que o caminho entre q e r^* é mais rápido do que o caminho entre q e r. Portanto, $TDD(q,r,t) = TDD(q,r^*,t)$ e a afirmação (2) segue válida.

□

v	H(v)	UNN(v)	UTDD(v, UNN(v))
a	3	b	4
b	0	b	0
c	4	d	4
d	0	d	0
e	4	d	5
f	6	b	8
g	6	d	8

Figura 3.7: Valor da função H, vizinho mais próximo em $\overline{G}$, e o limite superior do tempo de viagem.

3.3 Exemplo de Execução

Como um exemplo de entrada, considere o grafo da Figura 3.3 e o custo das arestas como apresentados na Figura 3.4. Note que o tempo de viagem para as arestas (b,c) e (c,d) não são apresentados somente porque tais custos não serão úteis no exemplo dado.

3.3.1 Pré-processamento

O resultado da etapa de pré-processamento para o grafo da Figura 3.3 e custos de Figura 3.4 é apresentado na Figura 3.7. Para cada vértice, uma busca 1NN não dependente do tempo (ou seja, estática) é executada nos grafos $\underline{G}$ e $\overline{G}$.

Por exemplo, considere o vértice f na Figura 3.3. A distância de f até b em $\underline{G}$ é $LTDD(f,b) = 6$ e de f até d é $LTDD(f,d) = 8$, portanto $H(f) = 6$. A busca 1NN executada

em $\underline{G}$ a partir de f deve encontrar como ponto de interesse mais próximo o vértice b. Além disso, a distância de f até b em $\overline{G}$ é $UTDD(f,b) = 8$ e de f até d e $UTDD(f,d) = 9$, portanto $UNN(f) = b$ e $UTDD(f,UNN(f)) = 8$. Portanto, a busca 1NN executada em $\overline{G}$ a partir de f deve encontrar como ponto de interesse mais próximo o vértice d.

3.3.2 Processamento da Consulta

Considere que o algoritmo recebe com entrada o ponto de consulta $Q = \langle (f,e), \frac{1}{4} \rangle$, $k = 1$, e o instante de partida $t = 0$. O estado inicial do grafo pode ser visto na Figura 3.8, onde o ponto de consulta é representado pelo triângulo azul.

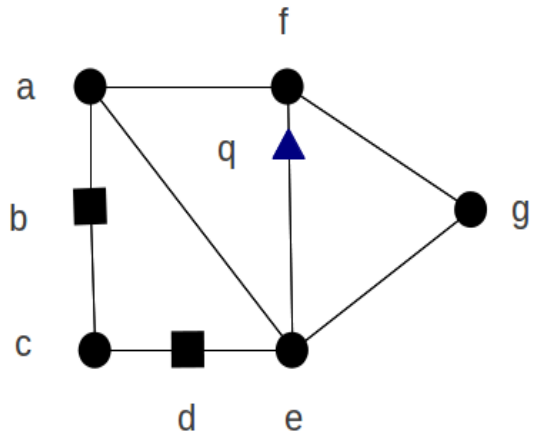
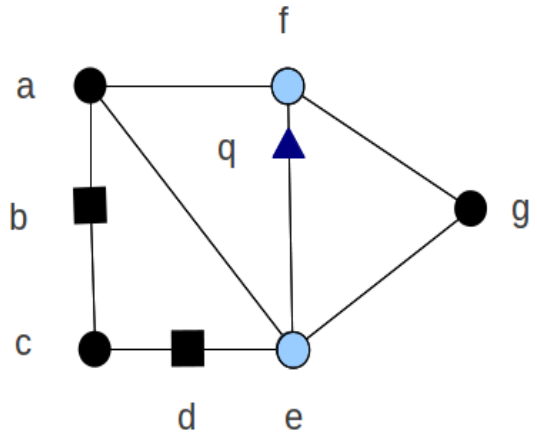


Figura 3.8: No estado inicial todos os vértices são não-visitados.

Primeiramente, o algoritmo calcula os tempos de viagem T_e e T_f de q para f e para e, os valores L_f e L_e, e os tempos de chegada AT_f e AT_e. Então a fila Q é inicializada, seu estado agora é $Q = \langle (f, AT_f = 1, T_f = 1, L_f = 7), (e, AT_e = 3, T_e = 3, L_e = 7) \rangle$, como apresentado na Figura 3.9(b). Além disso, Q_U também é inicializada e seu estado atual é $Q_U = \langle (d,8), (b,9) \rangle$ (Figura 3.9(c)). Os valores em Q_U são estes pois d é o vizinho mais próximo de e em $\overline{G}$ (veja a Figura 3.7) e $TT(q,e,t) + UTDD(e,d) = 8$; e b é o vizinho mais próximo de f em $\overline{G}$ e $TT(q,f,t) + UTDD(f,b) = 9$.

O primeiro vértice avaliado é f. A adjacência de f são os vértices a e g. Como o estado de a e g é não visitado, duas novas entradas são criadas $(a, AT_a = 4.5, TT_a = 4.5, L_a = 7.5)$ e $(g, AT_g = 3.5, TT_g = 3.5, L_g = 9.5)$. Como $L_g > Q_U(1)$, a entrada referente ao vértice g não é enfileirada. Como $UNN(a) = b$ e $TT_a + UTDD(a,b) = 8.5$. Note que foi encontrado um limite superior melhor do que 9.5 para b, então Q_u deve ser atualizada para $Q_u = \langle (d,8), (b,8.5) \rangle$. O estado das filas está ilustrado na Figura 3.10(c)

O segundo vértice avaliado é e. A adjacência de e são os vértices a, d e g. As entradas da fila referentes a estes vértices são $(g, AT_g = 6, TT_g = 6, L_g = 12)$, $(d, AT_d = 8, TT_d = 8, L_d = 8)$ e $(a, AT_a = 4, TT_a = 4, L_a = 7)$. O vértice g pode ser descartado, pois $L_g > Q_U(1)$. Além disso, a entrada para a em Q deve ser atualizada para o novo valor, pois a foi encontrado através de um caminho mais rápido que o anterior e o vértice d deve ser incluído em Q. Neste



(a) Vértices em azul representam os vértices que já foram visitados. Nesse caso, f e e.

f	e	$\perp$

(b) Q

(d,8)	(b,9)	$\perp$

(c) Q_U

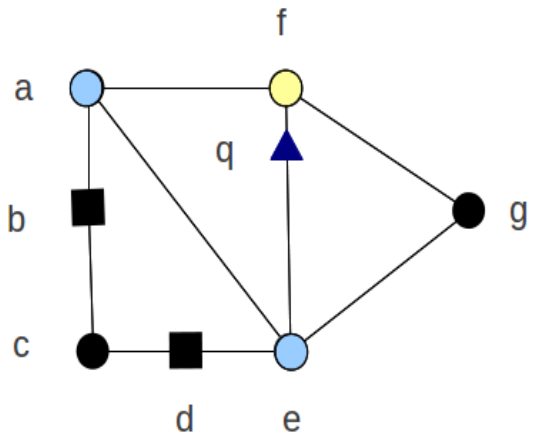
Figura 3.9: Acima, o estado da entrada do algoritmo (a). Abaixo, o estado da fila de prioridade de vértices candidatos à expansão (b) e estado da fila de prioridade de limites superiores (c).

momento, o estado da fila Q é $Q = \langle (a, AT_a = 4, TT_a = 4, L_a = 7), (d, TT_d = 8, TT_d = 8, L_d = 8) \rangle$ (Figura 3.11(b)). Além disso, Q_U é atualizada, sua nova configuração é $Q_u = \langle (d,8), (b,8) \rangle$ (Figura 3.11(c)).

O próximo vértice removido é a. Embora a adjacência de a seja o ponto de interesse b, b é descartado, pois $L_b = 8.5 > Q_U(1)$, significando que b pode ser encontrado por um caminho mais rápido do que o atual. Depois disso, $Q = \langle (d, AT_d = 8, TT_d = 8, L_d = 8) \rangle$ e d é removido e incluído em S_{NN}. Como $k = 1$ o algoritmo finaliza.

3.4 Conclusão

Neste capítulo foi definido o problema de processamento de consultas kNN em redes dependentes do tempo. O algoritmo proposto para este problema também foi apresentado. A solução é baseada no algoritmo de expansão incremental com inclusão de busca A*. Para utilização da busca direta na expansão incremental, foi definida uma função heurística que dá o potencial de cada vértice de levar a um ponto de interesse mais próximo na rede. As provas de que a função heurística é admissível e consistente também foram apresentadas, bem como uma discussão da corretude do algoritmo e um exemplo de execução. O próximo capítulo descreve como as informações referentes ao TDG que representa a rede foram armazenadas e acessadas no disco.



(a) Os vértices visitados são e e a, em azul. O vértice avaliado é f, em amarelo.

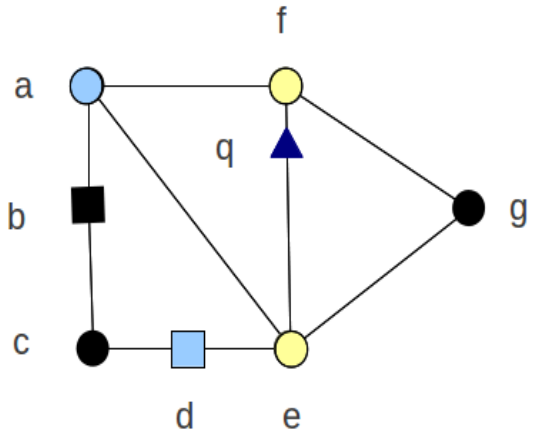
a	e	$\perp$

(b) Q

$(b,8.5)$	$(d,8)$	$\perp$

(c) Q_U

Figura 3.10: Acima, o estado da entrada do algoritmo após a primeira remoção de Q (a). Abaixo, o estado da fila de prioridade de vértices candidatos à expansão (b) e estado da fila de prioridade de limites superiores (c).



(a) Os vértices visitados são e e d, em azul. Os vértices avaliados são e e f, em amarelo.

a	d	$\perp$

(b) Q

$(b,8)$	$(d,8)$	$\perp$

(c) Q_U

Figura 3.11: Acima, o estado da entrada do algoritmo após a segunda remoção de Q (a). Abaixo, o estado da fila de prioridade de vértices candidatos à expansão (b) e estado da fila de prioridade de limites superiores (c).

Algorithm 4: TD-NE-A*

Input: A query point $q = ((u,v), \tau_q)$, an integer value k, a departure time t
Output: The set of k nearest neighbors of q

$TT_v \leftarrow \tau_q \times c_{(u,v)}(t)$;
$AT_v \leftarrow (t + TT_v) \bmod T$;
$L_v \leftarrow T_v + H(v)$;
En-queue (v, AT_v, TT_v, L_v) in Q;
if $(v,u) \in E$ **then**
    $TT_u \leftarrow (1 - \tau_q) \times c_{(v,u)}(t)$;
    $AT_u \leftarrow (t + TT_u) \bmod T$;
    $L_u \leftarrow T_u + H(u)$;
    En-queue (u, AT_u, TT_u, L_u) in Q;
end
$S_{NN} \leftarrow \emptyset$;
while $Q \neq \emptyset \wedge |S_{NN}| < k$ **do**
    $(u, AT_u, TT_u, L_u) \leftarrow$ De-queue Q;
    Mark u as de-queued;
    if $TT_u = H(u)$ **then**
        $S_{NN} \leftarrow S_{NN} \cup \{u\}$;
    end
    for $v \in adjacency(u)$ **do**
        $TT_v \leftarrow TT_u + c_{(u,v)}(AT_u))$;
        $AT_v \leftarrow (t + TT_v) \bmod T$;
        $L_v \leftarrow TT_v + H(v)$;
        if $L_v \leq Q_U(k)$ **then**
            if *v is not in Q* **then**
                En-queue (v, AT_v, TT_v, L_v) in Q;
                Mark v as en-queued;
            else
                Update L_v, if it is necessary;
                Re-order Q;
            end
            $U_{UNN(v)} \leftarrow TT_v + UTDD(v, UNN(v))$;
            if $UNN(v)$ *is not in* Q_U **then**
                En-queue $(UNN(v), U_{UNN(v)})$ in Q_U;
            else
                Update $U_{UNN(v)}$, if it is necessary;
                Re-order Q_U;
            end
        end
    end
end
Return S_{NN};

4 MÉTODO DE ACESSO EM TRÊS NÍVEIS PARA REDES DEPENDENTES DO TEMPO

Para a execução de algoritmos baseados em expansão da rede, fazem-se necessárias soluções para armazenamento da rede e execução de operações de acesso a certas informações, tais como listas de adjacência dos vértices e funções de custo (*travel-time*) em um determinado instante de tempo. Este capítulo apresenta o método de acesso proposto. A Seção 4.1 apresenta uma discussão inicial sobre este problema. A Seção 4.2 apresenta possíveis propostas e discute o porquê estas não se aplicam e em seguida, o método proposto é apresentado. Por fim, a Seção 4.4 conclui o capítulo.

4.1 Introdução

É interessante chamar atenção para algumas características das redes dependentes do tempo que devem ser consideradas no que diz respeito ao armazenamento e acesso dos dados. Tais características mostram que estas redes não podem ser armazenadas da mesma forma que redes estáticas, o mesmo vale para o acesso às informações da rede. A primeira observação é que os custos das arestas são dependentes do tempo, portanto as informações de dependência temporal devem ser armazenadas. Para representar tal dependência, para cada intervalo de tempo de tamanho constante, um novo custo (tempo de viagem) é calculado e essa informação deve ser armazenada. Portanto é necessário mais espaço para armazenamento dos custos.

Outra observação importante é que o custo de armazenamento das arestas da rede cresce à medida que a granularidade no tempo (número de intervalos) aumenta. De fato, torna-se difícil armazenar todos os custos das arestas de uma mesma lista de adjacências agrupados sem particionar a lista de adjacência, sendo necessárias duas ou mais páginas de disco para uma mesma lista de adjacência.

Por fim, uma última observação é a de que, dado que o acesso à lista de adjacências é executado para obter o custo das arestas em um determinado instante de tempo, armazenar todos os custos de uma mesma aresta agrupados implica em recuperar informações desnecessárias ao recuperar a página em questão.

Para apresentar o método de acesso proposto, serão apresentadas primeiramente duas propostas iniciais e uma breve discussão do motivo pelo qual estas propostas não se mostraram ser uma solução adequada para o problema em questão.

4.2 Possibilidades de Métodos de Acesso

A primeira proposta a ser discutida refere-se ao esquema com as seguintes características: os vértices da rede são indexados e a lista de adjacência de cada vértice indexado é recuperada; cada lista de adjacência recuperada contém os vértices da rede; todas as funções de custo de uma mesma aresta são armazenadas em conjunto. Cada vértice na lista de adjacência

contém um ponteiro para a página onde se encontram as funções de custo. Ao recuperar o custo para um vizinho do vértice recuperado, recupera-se o custo para todos os instantes de tempo possíveis. A Figura 4.1 ilustra o método descrito.

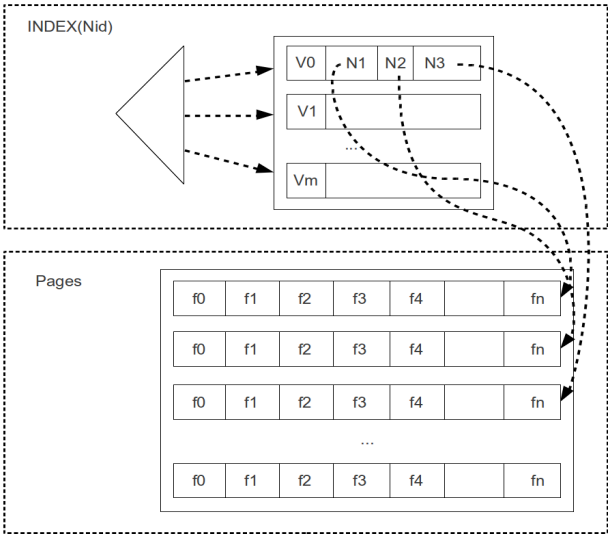


Figura 4.1: Método de indexação por vértice.

Neste método, à medida que a granularidade temporal das funções de tráfego aumenta, ou seja, que se tem um maior número de intervalos temporais, fica mais difícil armazenar o custo para a adjacência de um dado vértice em uma mesma página de disco. Isso acontece porque o espaço requerido para armazenar uma função de tráfego é proporcional à granularidade temporal. Outra desvantagem deste método é que se deseja recuperar apenas as informações correspondentes a um dado instante de tempo. Entretanto, o método recupera o custo para todos os instantes de tempo possíveis. Dessa forma, informações desnecessárias são obtidas. Assim, este método não é escalável com relação à granularidade temporal da rede.

Outro possível esquema de acesso consiste em agrupar as arestas da rede e seu respectivo custo de acordo com as partições temporais e armazená-las de tal forma que as arestas são replicadas para cada partição e os custos são armazenados para cada réplica de acordo com a partição. Dado um instante de tempo, as arestas da rede com os custos referentes àquele dado instante são recuperadas. A Figura 4.2 ilustra o método descrito.

Este método não é escalável no que diz respeito ao número de arestas da rede. À medida que o número de arestas aumenta, uma maior quantidade de páginas de disco deve ser recuperada e alocada em memória, o que o torna inviável para grandes redes. O método também implica na recuperação de informações desnecessárias, desde que não se deseja avaliar todas as arestas da rede e seu custo em uma determinada partição temporal, mas sim apenas às que incidem a um determinado vértice.

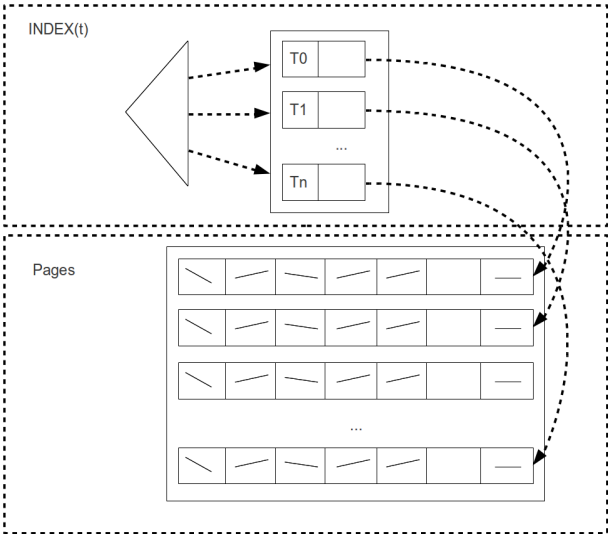


Figura 4.2: Método de indexação por tempo.

4.3 Método de Acesso em Três Níveis (ATN)

O método proposto, ilustrado na Figura 4.3, visa mesclar as duas soluções anteriores de forma a superar sua inviabilidade. Assim, o método em três níveis foi proposto como uma alternativa escalável de acordo com ambos, número de arestas da rede e granularidade temporal das funções de tráfego. O método ATN é assim chamado por ser composto por três níveis: dois níveis de índice, o Nível de Índice Temporal e o Nível de Índice de Grafo; e o Nível de Dados.

- **Nível de Índice Temporal (*Time-Level*)** O nível temporal permite acessar apenas as informações referentes a um dado ponto no tempo. Dessa forma, para cada aresta, apenas o tempo de viagem referente ao instante de tempo de partida desejado é recuperado. As entradas de dados no nível temporal contem ponteiros para estruturas de índice do nível de grafo.

- **Nível de Índice de Grafo (*Graph-Level*)** Cada índice no nível de grafo indexa informações de uma determinada partição temporal. Mais especificamente, cada estrutura de índice neste nível permite recuperar a lista de adjacência do vértice de interesse de um instante de tempo específico, de acordo com os identificadores dos vértices (*Nids*). As entradas de dados no nível de grafo contem ponteiros para as páginas de disco no nível de dados.

- **Nível de Dados (*Data-Level*)** As páginas de disco no nível de dados armazenam a lista de adjacência dos vértices. Os dados são clusterizados de acordo com o tempo. De tal forma que, os dados referentes a um mesmo instante de tempo fiquem tão próximos quanto possível no que diz respeito à organização do arquivo.

O acesso aos dados funciona como descrito pelo Algoritmo GETADJACENCYLIST. Primeiramente, o instante de tempo é passado como parâmetro para consulta ao índice *index*, através do método $query(t\_index, v)$. Esta chamada recupera a estrutura de índice no nível de

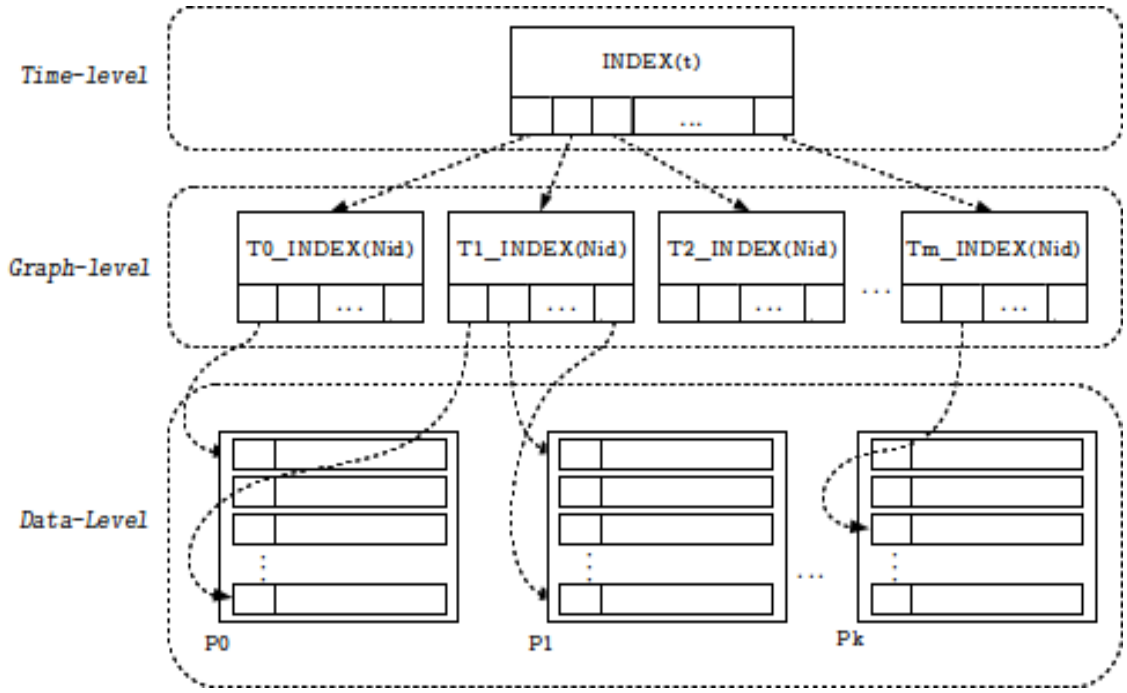


Figura 4.3: Representação do dos níveis do método de acesso proposto.

dados que indexa os dados da partição de tempo do instante t, indicada pela variável t_index. Em seguida, uma consulta é executada na estrutura de índice representada por t_index. Para a execução da consulta é feita uma chamada ao procedimento $query(t\_index, v)$ que recupera uma tupla composta pelo ponteiro p para a página onde se encontram os dados referentes à lista de adjacência do vértice v e pelo $offset$, que é o deslocamento em bytes do início da página apontada por p até o primeiro byte dos dados da lista de adjacência.

Algorithm 5: GETADJACENCYLIST

Input:
A vertex label v;
A time instante t;
Output: The adjacency list of v at the time instant t.

```
1 t_index ← query(index,t);
2 ⟨p,offset⟩ ← query(t_index,v);
3 adjlist ← read(p,offset);
4 return adjlist;
```

Cabe ressaltar que no método de acesso proposto as estruturas de índice são genéricas. Pode-se utilizar, por exemplo, uma árvore B^+ nos dois níveis de índice. Em tal caso, o número de páginas acessadas para cada entrada de dados recuperada é da ordem de $O(\log_F |T| + \log_F |V|)$, onde F é o *fan out* das árvores, T é o número de partições temporais que compõem o custo de uma aresta e $|V|$ é o número de vértices da rede. Note que, o índice do nível temporal pode ser pequeno o suficiente para caber na memória principal. Neste caso, não são contabilizadas quaisquer operações de E/S para acessa o nível temporal.

4.4 Conclusão

Neste capítulo foi feita uma discussão sobre o armazenamento dos dados de redes dependentes do tempo. Foram discutidos dois possíveis métodos e suas limitações. O método de acesso proposto para dar suporte à execução do algoritmo de processamento da consulta foi apresentado. O próximo capítulo descreve os trabalhos que mostram soluções para problemas relacionados com o problema de interesse deste trabalho.

5 TRABALHOS RELACIONADOS

Dentre as pesquisas relacionadas ao presente trabalho, destacam-se aquelas que apresentam soluções para o bem conhecido problema de menor caminho, e para a sua versão dependente do tempo, para consultas *k*-NN em redes estáticas e para o próprio problema de consultas *k*-NN em redes dependentes do tempo. Este capítulo apresenta os principais trabalhos e técnicas desenvolvidas nestes tópicos. O capítulo está organizado da seguinte forma. A Seção 5.1 apresenta técnicas para solucionar o problema do menor caminho, bem como sua versão dependente do tempo. A Seção 5.2 apresenta soluções para consultas *k*NN em redes estáticas. A Seção 5.3 apresenta outras soluções para consultas *k*NN em redes dependentes do tempo. Por fim, a Seção 5.4 conclui o capítulo.

5.1 Problema do Menor Caminho (estático) e Problema do Caminho mais rápido (dependente do tempo)

Estes problemas estão intrinsecamente relacionados com a solução de consultas de TD*k*-NN, desde que seu resultado depende do custo (tempo de viagem) do mais rápido para cada ponto de interesse do resultado. A solução mais conhecida para o problema do menor caminho em redes estáticas (não dependente do tempo) é o algoritmo de Dijkstra (DIJKSTRA, 1959), em sua execução os nós do grafo são visitados a partir do nó fonte, em ordem crescente da distância para o mesmo, até alcançar o nó destino. Nesta seção, serão descritas algumas técnicas que foram propostas para acelerar a execução do algoritmo de Dijkstra, bem como para a solução da versão dependente do tempo do problema de menor caminho.

Além do algoritmo de Dijkstra, muitas outras soluções para a solução do problema de menor caminho foram propostas. A maior parte destas soluções é definida por técnicas que têm o objetivo de acelerar o tempo de execução do algoritmo de Dijkstra. Elas são em geral baseadas em etapas de pré-processamento que anotam a rede com informações utilizadas para podar ou guiar a busca. Em (WAGNER; WILLHALM, 2007) pode-se encontrar um estudo sobre algumas das técnicas aplicadas a redes estáticas. Entretanto, vale ressaltar que as técnicas utilizadas em redes estáticas não são diretamente aplicáveis no caso dependente do tempo. Muito menos esforço foi dedicado ao caso em que a rede é dependente do tempo. O primeiro algoritmo que considera a variante dependente do tempo do problema de menor caminho é abordado em (COOKE; HALSEY, 1966). Este algoritmo é uma forma modificada do algoritmo de Bellman para encontrar o caminho mais rápido entre dois vértices numa rede. A seguir, são apresentadas algumas destas técnicas propostas para acelerar a execução do algoritmo de Dijkstra. Alguns exemplos de trabalhos que as utilizam, em redes estáticas ou em redes dependentes do tempo, são apresentados.

Uma das técnicas utilizadas para melhorar o desempenho do algoritmo de Dijkstra é a Busca Bidirecional. Um algoritmo baseado em busca bidirecional realiza simultaneamente duas buscas unidirecionais: uma busca na direção original, *forward* e uma busca na direção contrária, *backward*. Na busca *forward* a árvore de busca cresce a partir do nó fonte na direção

do nó destino, na qual a distância mínima da fonte para um conjunto de nós é descoberta. A outra busca, *backward*, acontece na direção reversa, do nó destino para o nó fonte. Nesta busca, a distância mínima de um conjunto de nós para o nó destino é descoberta. A busca *backward* é aplicada ao grafo reverso, ou seja,o grafo cujo conjunto de arestas é formado pelas arestas do grafo original na direção oposta. Dessa forma, por serem mantidas duas árvores de busca (*forward* ou *backward*) , uma para cada direção, uma das escolhas a ser feita ao utilizar uma busca bidirecional é qual árvore de busca será escolhida para expansão no próximo passo. Outra técnica é a Busca Direta ou Busca A*. A busca direta utiliza conhecimentos do domínio do problema para alterar a prioridade dos vértices enfileirados e, portanto, a ordem em que estes são visitados. A ideia é dar maior prioridade àqueles que têm o maior potencial de alcançar o objetivo. No caso do problema de menor caminho, o objetivo é o vértice destino. O potencial é dado por uma função heurística que deve ser consistente.

Em (GOLDBERG; HARRELSON, 2005) é apresentada uma proposta baseada em busca direta para solucionar o problema do menor caminho em grafos estáticos. A solução proposta neste trabalho combina a busca direta com busca bidirecional e uma outra técnica baseada em marcos (em inglês, *landmarks*, termo utilizado na literatura) e na desigualdade triangular. Nesta estratégia, inicialmente $d = \infty$. Então, partindo de um vértice fonte s e um vértice destino t, quando uma aresta (v,w) é avaliada pela busca *forward* e w já foi avaliado pela direção reversa, os menores caminhos $s-v$ e $w-t$ já são conhecidos, com tamanhos $d_f(v)$ e $d_b(w)$. Sendo $l(v,w)$ um limite inferior para a distância entre v e w, se $d < d_f(v) + l(v,w) + d_b(w)$, um caminho menor que o menor caminho visto até então foi encontrado, portanto o valor de d é atualizado. O algoritmo termina quando a busca em uma das direções encontra um vértice que foi avaliado na outra direção. Para estimar os limites inferiores $l(v,w)$, a estratégia seleciona um conjunto de *landmarks* (vértices específicos da rede) e, para cada vértice, pré-computa de e para todos os *landmarks*. Considerando um *landmark* L e seja $d(.)$ um distância para L. Então, pela desigualdade triangular, $d(v) - d(w) < dist(v,w)$. Similarmente, se $d(.)$ é uma distância de um caminho partindo de L, $d(w) - d(v) < dist(w,v)$. O limite inferior escolhido é o máximo entre todos os limites inferiores calculados utilizando os *landmarks*.

Em (NANNICINI et al., 2008) é proposta uma solução para o problema do menor caminho em redes dependentes do tempo que utiliza também uma estratégia baseada em busca direta e busca bidirecional. A solução proposta visa reduzir o espaço de busca, entretanto, não alcança resultados ótimos. Em redes dependentes do tempo a busca bidirecional não pode ser aplicada diretamente, pois o tempo de chegada (*arrival-time*) no destino é desconhecido, e este tempo influenciaria no custo da busca *backward*. Para solucionar este problema, a solução proposta utiliza na busca *backward* limites inferiores sobre custo dos arcos para restringir o conjunto de nós a ser explorado na busca *forward*. A busca direta é feita através do uso de *landmarks*. Em redes não-dependentes do tempo, como já foi exemplificado, *landmarks* podem ser usados na implementação da busca bidirecional. Para isso, a função heurística deve ser consistente para ambas as buscas, *forward* e *backward*. No método proposto em (NANNICINI et al., 2008), a busca *backward* é executada sobre o grafo de limites inferiores. Para cada aresta, o custo neste grafo é o limite inferior do custo da função na aresta. A busca *backward* é utilizada para fornecer limites para o critério de poda da busca *forward*. De maneira geral, a estratégia consiste em seguir com as duas buscas até que a busca *backward* contenha apenas os vértices

cuja chave associada não exceda um parâmetro μ, onde μ é um limite superior do custo da solução ótima. Quando a busca *backward* pára, todos os vértices avaliados por ela são incluídos em um conjunto M, de tal forma que a busca *forward*, a partir de então, avalia somente os vértices incluídos em M.

(DEMIRYUREK et al., 2011) estuda a computação *on-line* do caminho mais rápido em redes de rodovias dependentes do tempo e apresenta uma técnica baseada em busca bidirecional e A* que acelera a computação do caminho. A estratégia particiona a rede em partições não sobrepostas. Uma pré computação cria as partições e computa o limite inferior das distâncias entre as bordas das partições, entre vértices e borda das partições e entre bordas e vértices. O particionamento atribui a cada vértice um conjunto de partições. Uma outra etapa computa para cada par de partições o limite inferior do custo caminho mais rápido entre elas. Todos os custos das distâncias entre os vértices borda das partições são armazenados. O algoritmo *on-line* visita todos os nós alcançáveis a partir da consulta em direção ao destino, até que o destino seja alcançado. A busca aplicada é direcionada, a função heurística é obtida a partir dos valores de limites inferiores da distância entre bordas das partições, obtidas na pré computação.

5.2 *k* vizinhos mais próximos

Nesta Seção, algumas estratégias para o problema dos k vizinhos mais próximos, kNN (do inglês *k-Nearest Neighbor*) em redes estáticas são apresentadas. Tais estratégias não são diretamente aplicáveis em redes dependentes do tempo.

5.2.1 Algoritmos IER e INE

O problema de consulta kNN em redes de rodovias foi introduzido em (PAPADIAS et al., 2003). Nesse trabalho, os autores apresentam duas soluções para este problema, os algoritmos *Incremental Euclidean Restriction (IER)* e *Incremental Network Expansion (INE)*. O algoritmo IER usa a suposição de que a distância Euclidiana entre quaisquer dois pontos da rede é sempre menor que a distância na rede. Esta suposição é utilizada para recuperar os kNN pontos de acordo com a distância Euclidiana usando um índice espacial. O próximo vizinho mais próximo segundo a distância Euclidiana é recuperado, a distância na rede do ponto recuperado é utilizada como limite superior para o próximo vizinho de acordo com a distância na rede. O algoritmo IER vai expandindo vértices da rede, quando o vértice expandido tem um vizinho cuja distância entre o objeto de consulta e ele é maior que o limite superior, aquele caminho é descartado, ou seja, o vértice vizinho em questão não é enfileirado. O mesmo é feito para o próximo ponto mais próximo segundo a distância Euclidiana, até que k pontos sejam recuperados. O algoritmo INE é uma adaptação do algoritmo de Dijkstra. Iniciando do ponto de consulta q todos os vértices da rede alcançáveis a partir de q são visitados por ordem de sua proximidade para q até que os k objetos mais próximos sejam localizados. Os algoritmos IER e INE foram também comparados em (PAPADIAS et al., 2003). A avaliação dos algoritmos mostrou que o INE supera o IER em termos de custos de CPU. A razão para isto é o fato de que o INE visita apenas as arestas necessárias, enquanto os vizinhos mais próximos recuperados

pelo IER segundo a distância Euclidiana podem ser pistas falsas. Ou seja, embora os pontos sejam os mais próximos Euclidiana, segundo a distância na rede tais pontos estão mais distantes que muitos outros, principalmente quando a rede tem poucos pontos de interesse.

5.2.2 Solução baseada em células de Voronoi (VN^3)

Em (KOLAHDOUZAN; SHAHABI,) é apresentado um método baseado na pré computação dos polígonos de Voronoi da rede (NVP) e das distâncias entre alguns vértices para computação dos vizinhos mais próximos. Originalmente, um polígono de Voronoi caracteriza uma região de proximidade para k lugares em um plano onde a distância é definida pela distância Euclidiana. A definição de polígonos de voronoi em um grafo foi introduzida em (ERWIG,), como uma extensão da definição inicial. Na solução dada por (KOLAHDOUZAN; SHAHABI,), os NVPs são indexados por um método de acesso espacial. Os NVPs preservam as distâncias na rede, portanto usando NVPs indexados pode-se encontrar imediatamente o primeiro vizinho mais próximo de um objeto de consulta e reduzir o custo *on-line* do processamento de um consulta kNN. Os autores também demostraram uma propriedade que mostra que os próximos vizinhos mais próximos estão nas células adjacentes à célula que contém o último vizinho mais próximo a ser explorado, permitindo que apenas as células adjacentes precisem ser avaliadas. A pré-computação de algumas distâncias na rede é feita para dar suporte a uma maneira de calcular a distância exata entre o objeto de consulta e os pontos retornados, sem necessidade de computar todo o caminho.

5.2.3 Método das Ilhas

O Método das Ilhas, proposto em (HUANG; JENSEN; SALTENIS, 2005), consiste de um componente de pré computação e um componente de expansão da rede *on-line*. O componente de pré-computação armazena, para cada vértice na rede rodoviária, a referência para uma "ilha"que cobre o vértice e as distâncias na rede do vértice para o ponto de dados que gerou a ilha. Dado um valor de distância r, a ilha do ponto de dados p é o subconjunto da rede coberta pela expansão da rede partindo de p com limite de distância r. O valor r é dito o raio da ilha. O componente de expansão checa quais ilhas contêm o ponto de consulta q e mantém os pontos de dados dessas ilhas em uma fila de prioridade, então inicia o processo de expansão partindo de q até as bordas das ilhas. Na expansão, para cada vértice que entra na fila de prioridade são verificadas quais ilhas contêm tal vértice e os pontos de dados geradores das ilhas são novos candidatos. O processo termina quando a soma do raio expandido e o raio mínimo de todas as ilhas pré-computadas excedem a distância do ponto de consulta q para o k-ésimo ponto de dados da fila de prioridade de pontos candidatos.

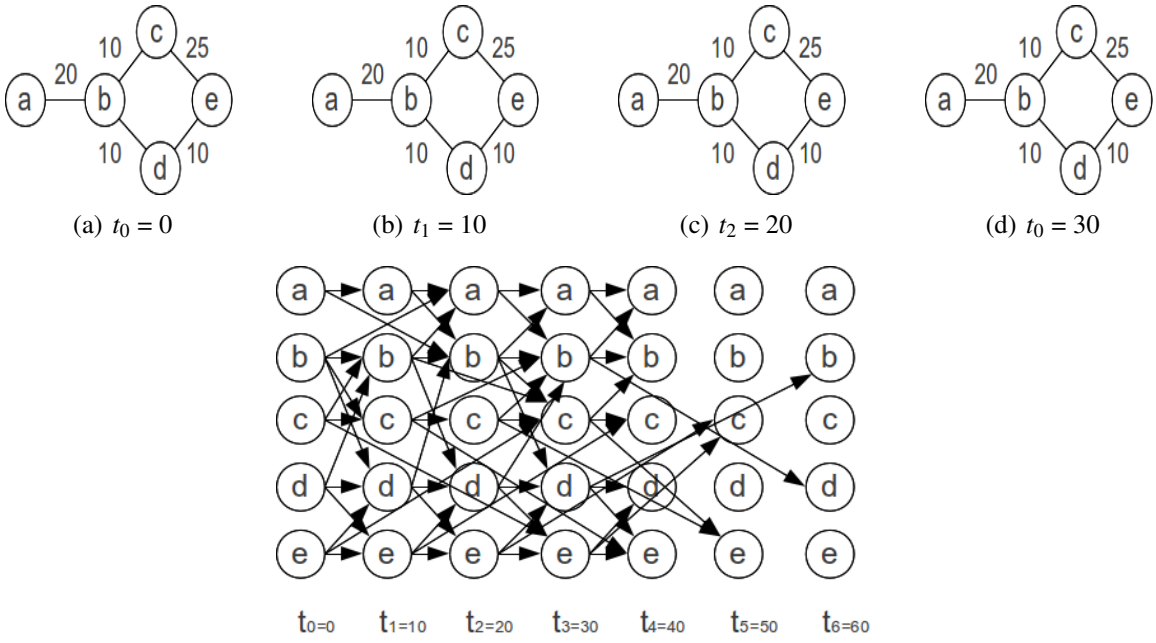
5.3 k-NN em Redes Dependentes do Tempo

Enquanto alguns trabalhos têm sido propostos para acelerar a computação dos menores caminhos em redes dependentes do tempo, poucos trabalhos recentes propõem métodos para

processar consultas *k*NN em redes dependentes do tempo. Esta seção apresenta os trabalhos que propõem soluções para o processar consultas *k*NN em redes dependentes do tempo (TD-*k*NN, do inglês *Time-Dependente k Nearest Neighbors*).

5.3.1 Algoritmos de Expansão Incremental

O problema de solucionar consultas *k*NN em redes dependentes do tempo foi introduzido em (DEMIRYUREK; KASHANI; SHAHABI, 2010). Neste trabalho, os autores comparam duas diferentes soluções para este problema. Os métodos apresentados são uma adaptação do algoritmo INE para redes dependentes do tempo. Uma das soluções aplica expansão incremental enquanto utiliza o modelo de rede expandida no tempo (Figura 5.3.1) como modelo de representação da rede. Este modelo discretiza o domínio temporal $T = [t_0; t_n]$ em n instantes de tempo igualmente espaçados $t = 0, 1, ..., t_n$ e constrói uma rede estática para cada instante de tempo. Portanto, o modelo de rede expandida no tempo replica a rede original n vezes. Então, o modelo conecta cada vértice de um determinado instante em t aos seus vizinhos no próximo instante, calculado pelo instante atual mais o custo da aresta nesse instante. Embora este modelo possa utilizar os mesmos algoritmos aplicados a redes estáticas, as soluções que o utilizam podem apresentar resultados incorretos. Neste modelo, o custo de uma aresta para todo um intervalo é dado pelo custo do instante inicial, resultando em um erro entre o valor do menor caminho e do caminho calculado.



(a) $t_0 = 0$ (b) $t_1 = 10$ (c) $t_2 = 20$ (d) $t_0 = 30$

(e) Grafo expandido

Figura 5.1: Exemplo de grafo expandido no tempo. Fonte (DEMIRYUREK; KASHANI; SHAHABI, 2010).

O segundo método é o algoritmo TD-NE que expande os vértices da rede na ordem de menor distância para o ponto de consulta considerando as funções de custo. Uma tabela

hash que associa as funções de custo as arestas da rede é consultada durante a expansão para atualizar o custo dos caminhos avaliados.

5.3.2 Índices TNI e LNI

Em (DEMIRYUREK; BANAEI-KASHANI; SHAHABI, 2010) é apresentada uma solução que assemelhasse com uma versão da solução baseada em células de Voronoi em redes dependentes do tempo. Neste trabalho, é proposto um processo de pré-computação que constrói duas estruturas que indexam a rede e os pontos de interesse chamadas de *Tight Network Index (TNI)* e *Loose Network Index (LNI)*. Ambas as estruturas são compostas por células que referenciam um ponto de interesse me particular, tal que, se um ponto de consulta q está dentro de uma célula de um ponto de interesse p no TNI, então certamente p é seu vizinho mais próximo. Entretanto, se q está fora de uma célula de um ponto de interesse p no LNI, então p certamente não é o vizinho mais próximo de q. Como na estratégia NVP para redes estáticas, usando o TNI pode-se encontrar imediatamente o primeiro ponto de interesse mais próximo ao objeto de consulta. Para os demais pontos de interesse ($k > 1$), o próximo vizinho mais próximo é o ponto de interesse gerador de uma das células vizinhas às células dos pontos de interesse já encontrados. Para decidir qual o próximo ponto de interesse, a rede é expandida incrementalmente até encontrar um ponto de interesse em uma das células vizinhas. Não está claro como este processo poderia funcionar no caso em que considera-se que o tráfego em direções opostas de uma mesma via da rede pode ser diferente. Esta é uma importante consideração que reflete mais fielmente cenários reais, desde que o custo de um caminho é determinado, também, pela sua orientação.

5.3.3 Estratégia dos grafos incorporados

Uma estratégia baseada em grafos incorporados mapeia o grafo da rede em um espaço vetorial sobre a distância de um conjunto de vértices referenciados. Estes vértices são, em geral, selecionados randomicamente ou de acordo com alguma heurística. Então, as distâncias entre cada vértice referenciado e todos os outros vértices da rede são calculadas. O grafo incorporado é utilizado para estimar limites para a distância real entre o ponto de consulta e um ponto de interesse, o limite inferior que subestima a distância e o superior que superestima a distância. Esta técnica foi proposta em (KRIEGEL et al., 2007) para acelerar o cálculo de consultas kNN e de *range* em redes estáticas. Primeiramente, uma consulta de ranqueamento no espaço incorporado é executada. O ranquemento obtém os pontos de interesse na ordem dos limites inferiores da distância entre o ponto de consulta e os pontos de interesse. Dentre os k primeiros candidatos, aqueles que têm o limite superior da distância menor que o limite inferior do k-ésimo candidato são reportados como resultado da consulta. Após esta etapa, o algoritmo refina o conjunto de candidatos, calculando suas distâncias para o ponto de consulta, e retornando aqueles que tem o limite inferior menor ou igual a k-ésima distância.

Para aplicar esta técnica em redes dependentes do tempo, é necessário adaptá-la. Em (KRIEGEL et al., 2011), os autores propuseram duas adaptações do método. A primeira

delas é uma estratégia estática, pois as distâncias entre cada objeto e os vértices referenciados são computadas de maneira independente da variação dos custos das arestas sobre o tempo. A segunda é uma estratégia dependente do tempo, pois considera a dependência temporal quando computa a distância entre os objetos e os vértices referenciados.

Na estratégia estática, a ideia é extrair dois grafos estáticos com custos constantes de valores mínimos e máximos dos custos do grafo original. Então, os grafos incorporados destes grafos são utilizados para estimativa dos limites inferiores e superiores das distâncias entre dois vértices. Na estratégia dependente do tempo, ao invés de utilizar apenas uma distância entre um vértice e cada vértice referenciado, são necessárias as distâncias para cada possível tempo de partida. Portanto, no espaço incorporado, cada vértice é representado por uma série de posições dependentes do tempo, ao invés de um ponto estático. Nesse caso, os limites inferiores e superiores são calculados de uma maneira diferente. Ambas as estratégias, aplicam a estratégia de processamento de consulta de filtragem/refinamento. Na etapa de filtragem, as aproximações da distância são utilizadas para podar falsos positivos, utilizando os limites inferiores, e identificar sucessos, utilizando os limites superiores. A etapa de refinamento calcula a distância real entre o ponto de consulta e os pontos de interesse candidatos, utilizando o algoritmo de Dijkstra.

Uma desvantagem deste método é a etapa de pré-processamento. Nesta etapa, para cada tempo de partida possível, deve ser feita uma nova busca a partir de todo vértice do grafo para o cálculo dos menores caminhos. E como o custo do caminho entre dois vértices não é simétrico, o algoritmo não pode expandir os vértices na computação dos menores caminhos iniciando dos nós referenciados e visitar todos os outros nós, ao invés disso, deve iniciar uma busca a partir de cada um dos nós, o que faz o algoritmo computacionalmente caro. Note que a etapa de pré-processamento proposta neste trabalho não depende do número de partições temporais da rede. Além disso, como o domínio temporal é contínuo, este método assume que para instante para intervalo de tempo para o qual o pré-processamento foi executado, o tempo de viagem por via é constante, o que pode acumular erros no cálculo dos custos.

5.4 Conclusão

Este capítulo apresentou algumas soluções para os principais problemas relacionados ao problema de interesse deste trabalho: consultas de menor caminho; consultas de menor caminho dependente do tempo; consultas *k*NN em redes estáticas e consultas *k*NN em redes dependentes do tempo. Os próximos capítulos descrevem os experimentos feitos para avaliação do algoritmo.

6 AVALIAÇÃO EXPERIMENTAL DO ALGORITMO TD-NE-A*

Este capítulo descreve o processo de avaliação experimental do algoritmo proposto neste trabalho utilizando um conjunto de dados sintéticos. O capítulo está organizado da seguinte maneira. A Seção 6.1 descreve como os experimentos foram realizados. A Seção 6.2 descreve os resultados em relação à densidade de pontos de interesse da rede. A Seção 6.3 descreve os resultados em relação ao valor de entrada k. A Seção 6.4 descreve os resultados em relação ao tamanho da rede. Por fim, a Seção 6.5 conclui o capítulo.

6.1 Descrição do ambiente e configuração dos experimentos.

O algoritmo proposto neste trabalho (TD-NE-A*) foi comparado ao algoritmo proposto por (DEMIRYUREK; KASHANI; SHAHABI, 2010) (TD-NE). Os experimentos descritos nesta seção foram conduzidos em um computador com sistema operacional baseado em Linux, com 3.00GHz de CPU e 4 GB de memória RAM. Os algoritmos foram implementados em linguagem C++. Os dados consistiram de um conjunto de redes dependentes do tempo geradas sinteticamente. Cada rede foi gerada com um particionamento temporal de 96 pontos no tempo, ou seja, uma função de tráfego a cada 15 minutos de um dia. Os vértices das redes têm grau (número de arestas incidentes) uniformemente distribuído de valor médio igual a 4.

O método foi avaliado com dados de entrada gerados de acordo com três parâmetros distintos: o tamanho das redes (número de vértices), o tamanho da consulta k e a densidade dos POIs (a taxa dada pela cardinalidade de POIs dividida pelo tamanho da rede). Os valores atribuídos a cada um destes parâmetros são apresentados na Tabela 6.1. Para cada experimento, um parâmetro tem seu valor variando entre os valores referentes a sua linha na tabela, enquanto os outros dois tem seus valores fixados e iguais ao valor em negrito.

Em todos os experimentos, para cada configuração de parâmetros, foram geradas 10 redes sintéticas que se distinguiram na estrutura, na atribuição dos pontos de interesse e nas funções de custo. As funções de custo foram geradas de tal forma a respeitar a propriedade FIFO. Para cada um dessas redes, foram executadas 10 consultas com os pontos de partida randomicamente selecionados. Portanto, para cada configuração de parâmetros foram geradas um total de 100 consultas. Os gráficos com os resultados apresentam barras com os limites inferiores e superiores de intervalo de confiança de 95% para o ganho relativo da estratégia proposta, assumindo distribuição normal dos dados.

Além disso, os algoritmos foram comparados de acordo com duas métricas: o número de vértices expandidos e com o número de operações de entrada-saída (E/S) para aces-

Tabela 6.1: Valores dos parâmetros dos experimentos.

Densidade de POIs	5% ,**10%**, 20%
Tamanho da consulta k	1 , 10, **20**, 30
Tamanho da rede	1000, **2000**, 4000, 10000

sar as páginas de disco no nível de dados. Cada uma dessas métricas foi analisada em relação à densidade de POIs, ao tamanho da rede e ao tamanho da consulta. As operações de E/S às páginas de disco no nível de dados foram simuladas considerando LRU (*least recently used*) como política de cache, o número de páginas na cache foi de 5% do número de páginas de disco no nível de dados. Com os experimentos, pode-se verificar que uma diminuição do número de vértices expandidos implica também em uma diminuição do número de páginas de disco acessadas.

6.2 Avaliação do efeito da densidade de POIs.

A densidade de POIs variou entre 5%, 10% e 20% de números de pontos de interesse. Os pontos de interesse foram uniformemente distribuídos entre os vértices da rede. Para cada diferente valor de densidade, foram geradas 10 redes dependentes do tempo distintas (no que diz respeito à estrutura e funções de *travel-time*), com 2000 vértices cada um delas. Foram executadas 10 consultas, cada consulta com valor k=20 e posição randomicamente selecionada sobre cada rede uma aresta da rede. Vale ressaltar que os valores percentuais atribuídos não refletem a densidade de pontos de interesse de uma rede real (que deve ser menor), contudo o experimento é uma prova de conceito a respeito do comportamento do algoritmo em relação à densidade de pontos de interesse.

A Figura 6.1 ilustra a diferença entre a média do número de vértices acessados pelos algoritmos. Pode-se observar que, em ambos os algoritmos, é necessário expandir uma quantidade menor de vértices à medida que a rede torna-se mais densa. Esta observação é decorrente do fato de que o esforço necessário para encontrar os k pontos de interesse é cada vez menor à medida que a densidade de POIs aumenta, pois número de POIs cresce para toda sub-rede avaliada. Note também que à medida que a rede torna-se esparsa, o ganho obtido pelo algoritmo TD-NE-A* em relação TD-NE em número de vértices acessados aumenta. Nestes experimentos, TD-NE-A$*$ descartou uma média de 33,33% a 51% dos vértices expandidos por TD-NE, como apresentado na Figura 6.2.

Essa primeira avaliação apresentada mostra que a heurística utilizada pode reduzir o número de vértices expandidos em redes dependentes do tempo não-estacionárias. Entretanto, dado que os dados estão armazenados em disco, a próxima avaliação mostra como isso se reflete em relação ao número de operações E/S. As Figuras 6.3 e 6.4 mostram que a diferença em relação ao número de vértices expandidos apresenta-se também quando o número de páginas de dados acessadas é avaliado.

A Figura 6.3 apresenta como o número de páginas de disco acessadas se modifica em função da densidade de POIs. Mais uma vez, a diferença na avaliação dos algoritmos torna-se menor à medida que a densidade aumenta. Como pode ser visto na Figura 6.4, a redução no número de operações E/S do TD-NE-A* chega a até 50% em relação ao algoritmo concorrente.

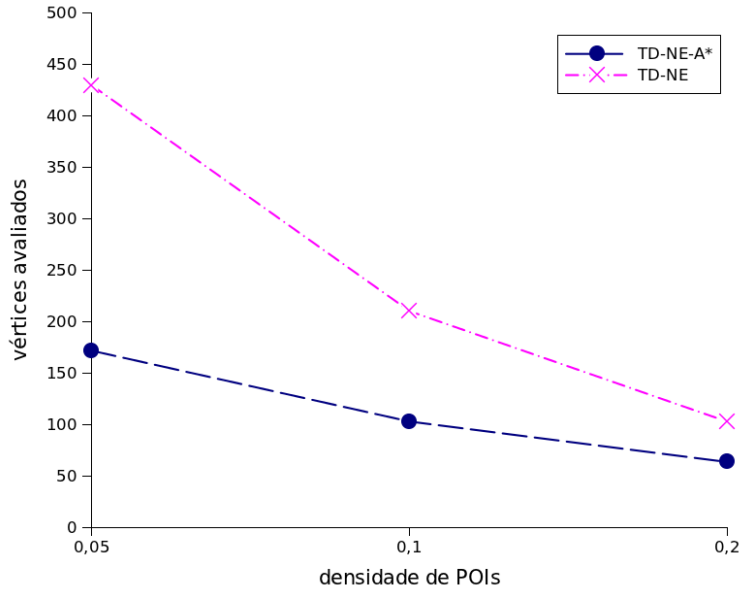


Figura 6.1: Avaliação do número de vértices expandidos quando a densidade de pontos de interesse cresce.

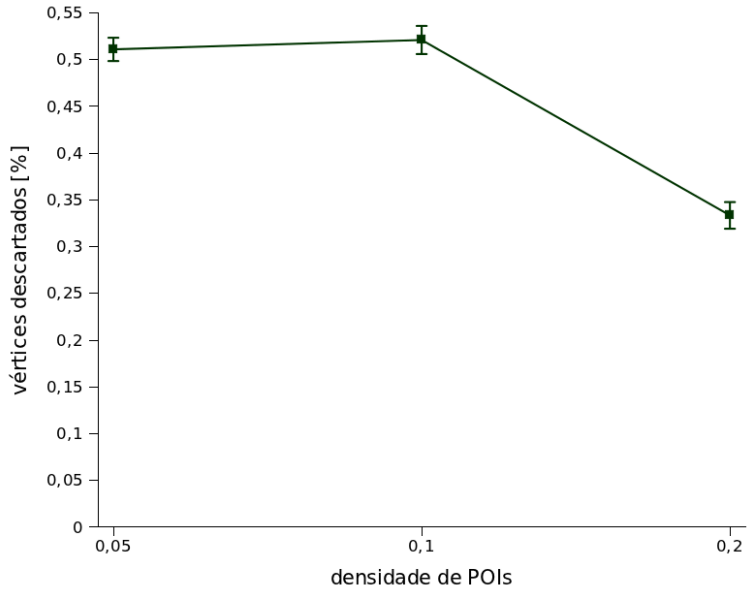


Figura 6.2: Avaliação do ganho em percentual de vértices expandidos quando a densidade de pontos de interesse cresce.

6.3 Avaliação do efeito do tamanho da consulta.

A seguir, são apresentados os resultados experimentais do algoritmo TD-NE-A* em relação à variação dos valores de k. Para que o efeito do valor de k fosse avaliado, foram geradas 10 redes distintas cada uma com 2000 vértices e densidade 10% do número de vértices como pontos de interesse (valores em negrito na tabela de parâmetros). O tamanho da consulta variou em $k = \{1, 10, 20, 30\}$. Para cada rede e valor de k foram geradas 10 consultas com ponto de partida e instante inicial randomicamente selecionados.

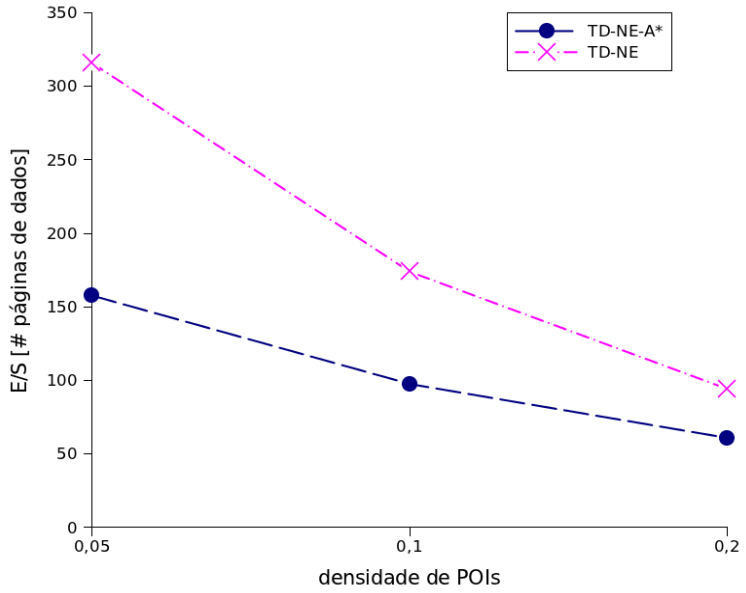


Figura 6.3: Avaliação do número de páginas acessadas quando a densidade de pontos de interesse cresce.

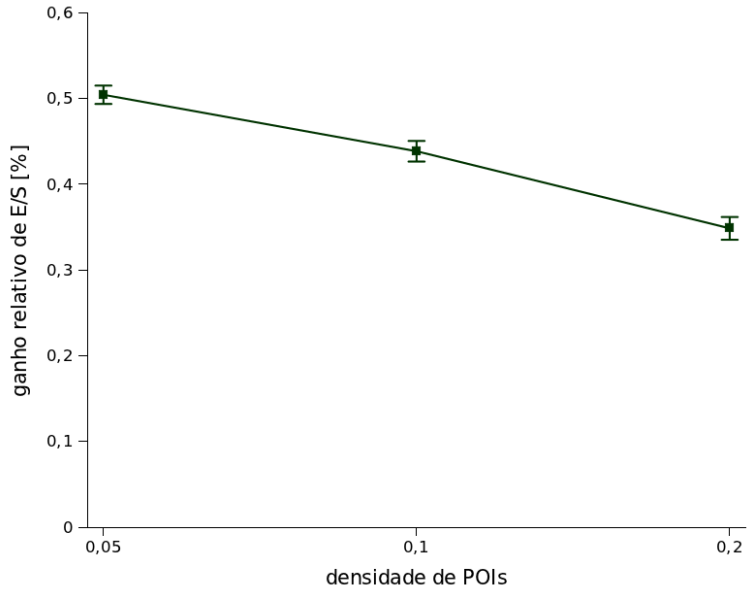


Figura 6.4: Avaliação do ganho em percentual de páginas acessadas quando a densidade de pontos de interesse cresce.

A Figura 6.5 compara o número médio de vértices expandidos quando o valor de k aumenta. Esse número aumenta com k em ambos os algoritmos, devido ao fato de que mais vértices precisam ser avaliados para que todos os pontos de interesse sejam encontrados. A Figura 6.6 mostra o percentual médio de ganho em relação aos vértices expandidos pelo algoritmo TD-NE-A*. Para todos os valores de k, TD-NE-A* superou TD-NE em número de vértices expandidos, conseguindo reduzir em até 50%, em média, o número de vértices expandidos em relação ao algoritmo TD-NE. As barras de erro na Figura 6.6 indicam o intervalo de confiança das médias obtidas.

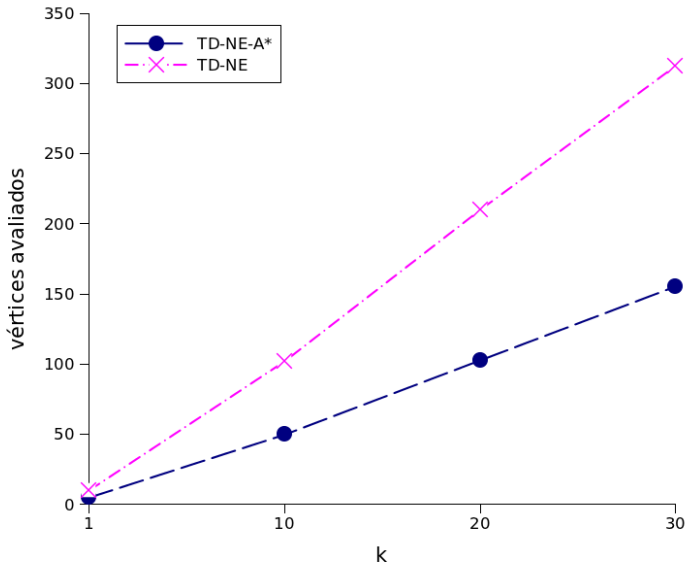


Figura 6.5: Avaliação do número de vértices expandidos quando o valor de k aumenta.

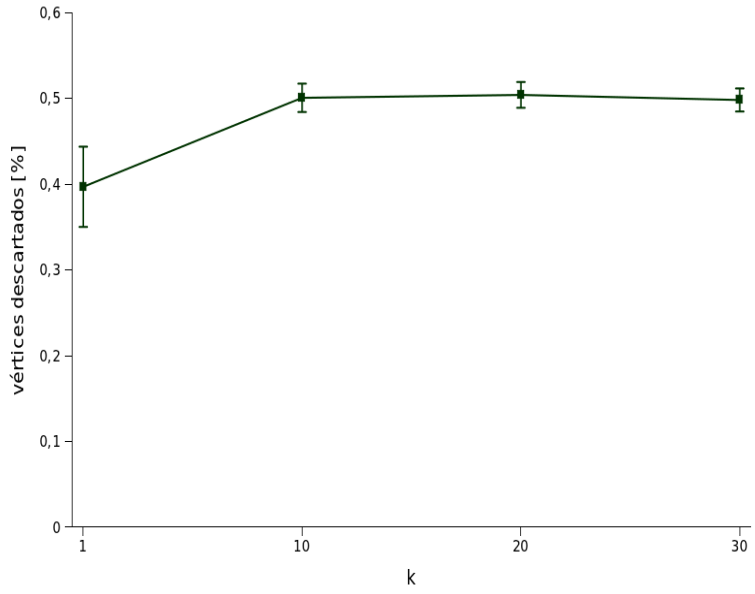


Figura 6.6: Avaliação do ganho em percentual de vértices expandidos quando ao valor de k aumenta.

Da mesma forma, o número de acessos às páginas de disco também aumenta em ambos os algoritmos quando o valor de k aumenta. A Figura 6.7 compara o número de páginas acessadas pelos algoritmos no experimento. Note que a diferença entre o número de páginas acessadas torna-se maior à medida que o valor de k aumenta. Essa diferença pode ser muito pequena em valor nominal quando o k é pequeno, entretanto em valor relativo mostrou-se sempre maior que 40%. A Figura 6.8 mostra o percentual de ganho em relação ao número de operações E/S. Note que embora para $k = 1$ a diferença entre páginas acessadas pareça bem pequena, o ganho relativo é maior que 40% neste caso. Esse ganho manteve-se entre 40% e 50%. As barras de erro indicam que o valor médio de acesso torna-se mais confiável à medida que o valor de k

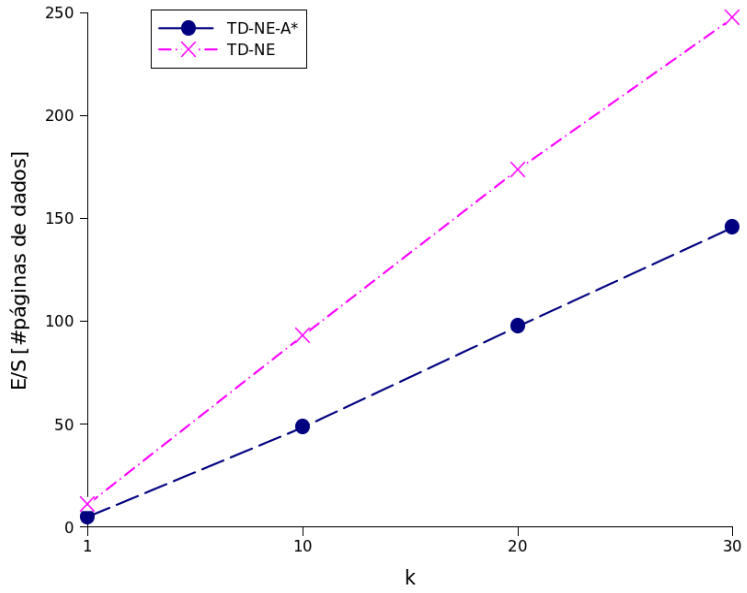


Figura 6.7: Avaliação do número de páginas acessadas quando o valor de k aumenta.

cresce.

6.4 Avaliação do efeito do tamanho da rede.

Neste experimento, foram geradas 10 redes dependentes do tempo para cada um dos tamanhos: 1000, 2000, 4000 e 10000. Cada uma das redes com 10% de densidade de pontos de interesse. Foram executadas 10 consultas sobre cada rede distinta. As consultas foram geradas com ponto de partida e instante inicial randomicamente selecionados e $k = 20$.

A Figura 6.9 ilustra o comportamento médio dos algoritmos quando o tamanho da rede cresce. Este experimento indica que o tamanho da rede não afetou significantemente o número médio de vértices expandidos. Isto ocorreu porque os pontos de interesse foram distribuídos uniformemente sobre a rede. A distribuição uniforme para os pontos de interesse permite afirmar que, a partir de um ponto de consulta, o mesmo número de vértices, em média, deve ser expandido, pois cada sub-rede de um mesmo tamanho deverá conter o mesmo número de pontos de interesse.

Portanto, em uma rede pequena, a sub-rede expandida deve ser similar em tamanho à sub-rede expandida de uma rede de maior tamanho, desde que o valor de k é o mesmo. Entretanto, como mostra a Figura 6.9, o algoritmo TD-NE-A* supera o concorrente em todos os tamanhos de rede. A Figura 6.10 mostra que o ganho em vértices expandidos chega a superar 50% quando a rede ultrapassa o tamanho de 4000 vértices.

Ao contrário do número de vértices expandidos, o tamanho da rede interfere no número de páginas acessadas. Entretanto, isso ocorre apenas para o algoritmo TD-NE. Veja na Figura 6.11 que quanto maior o tamanho da rede, maior o número de operações E/S executadas pelo TD-NE. Entretanto, esse mesmo número permanece estável na execução do TD-NE-A*.

Isto mostra a escalabilidade do algoritmo e método de acesso e armazenamento propostos em função do tamanho da rede.

A Figura 6.12 apresenta o ganho com respeito ao número de acessos às páginas de disco. A Essa figura mostra que o ganho varia de 32% a até 52% em operações de leitura. As barras de erro indicam o intervalo de confiança para estas estimativas.

6.5 Conclusão

Neste capítulo foram apresentados os resultados dos experimentos feitos com dados sintéticos. Para experimentação foram geradas 40 redes dependentes do tempo e executadas 100 consultas em cada um, para cada configuração de parâmetros. Os experimentos mostraram que o algoritmo TD-NE-A* pode reduzir em até 50% o número de páginas de dados acessadas.

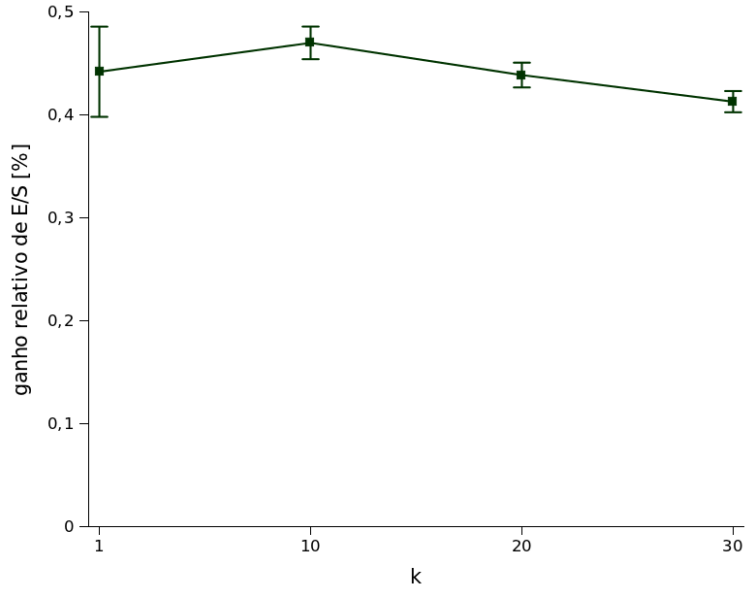


Figura 6.8: Avaliação do ganho em percentual de páginas acessadas quando o valor de k aumenta.

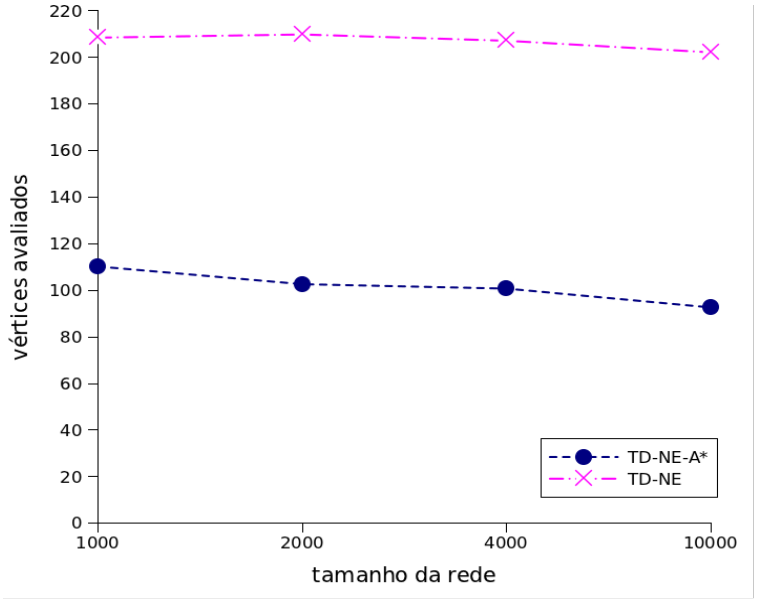


Figura 6.9: Avaliação do número de vértices expandidos quando o tamanho da rede aumenta.

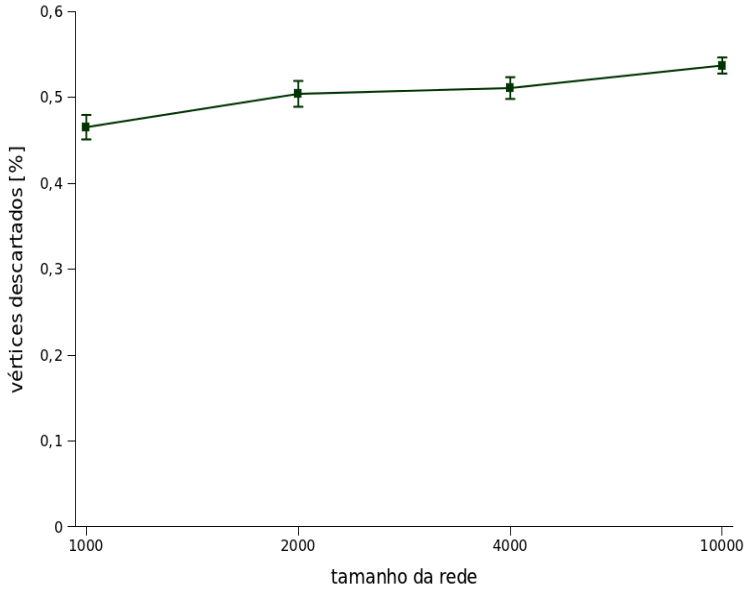


Figura 6.10: Avaliação do ganho em percentual de vértices expandidos quando o tamanho da rede aumenta.

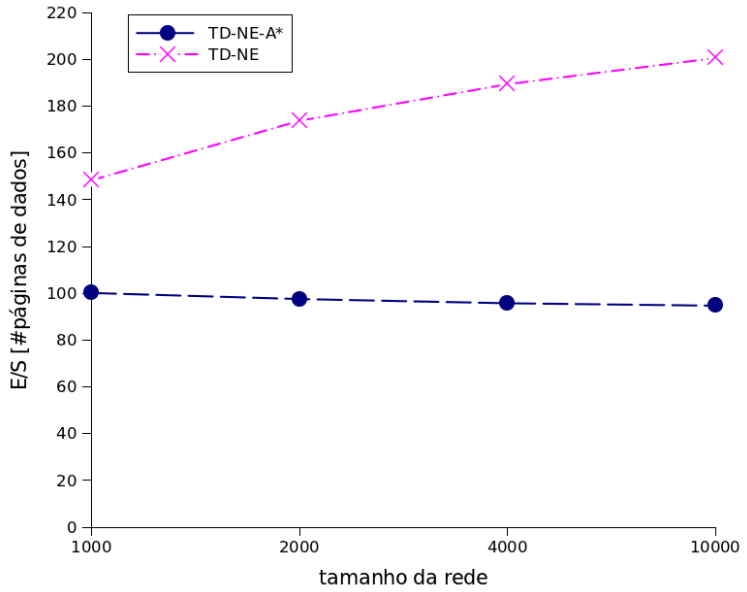


Figura 6.11: Avaliação do número de páginas acessadas quando o tamanho da rede aumenta.

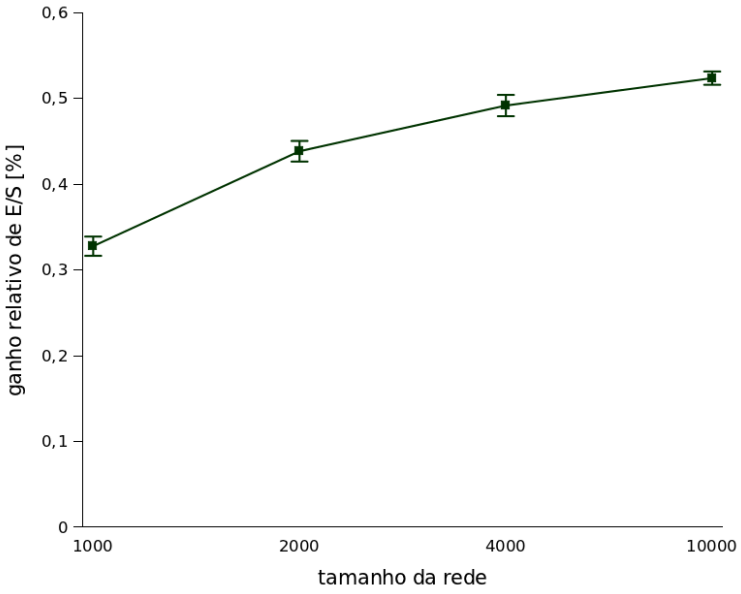


Figura 6.12: Avaliação do ganho em percentual de páginas acessadas quando o tamanho da rede aumenta.

7 AVALIAÇÃO EXPERIMENTAL COM DADOS REAIS

Este capítulo apresenta os experimentos feitos para avaliação do algoritmo utilizando um conjunto de dados reais. A Seção 7.1 apresenta o conjunto de dados utilizado. A Seção 7.2 apresenta os resultados e a Seção 7.3 conclui o capítulo.

7.1 Conjunto de dados

Para esta avaliação do algoritmo TD-NE-A* foi utilizada a rede de rodovias da cidade de Pisa, Itália. O algoritmo utilizado para geração das velocidades foi o GMatch (CINTRA P. ; TRASARTI, 2013). Os dados das velocidades médias foram cedidos pelo laboratório KDD Lab [1]. As velocidades foram construídas com base em um conjunto de observações de GPS de 30.000 usuários reais de veículos em Toscana, obtidos em um período de 5 semanas cobrindo diferentes tipos de territórios. Tal conjunto de observações de GPS pode ser visualizado na Figura 7.1.

Para os experimentos desta seção, os algoritmos foram desenvolvidos em linguagem Java utilizando a biblioteca XXL (eXtensible and fleXible Library) [2].

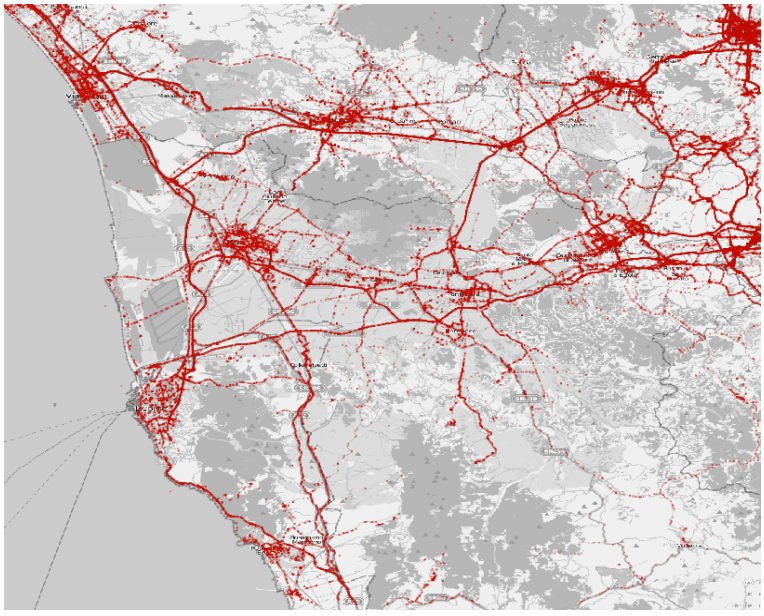


Figura 7.1: Conjunto de dados de observações de trajetórias que geraram a rede projetada.

Embora a rede original de Pisa contenha 359.569 arestas, apenas 154.014 foram utilizadas nos experimentos, porque não havia qualquer observação de GPS sobre as vias restantes.

[1] http://www-kdd.isti.cnr.it/

[2] XXL é uma biblioteca desenvolvida em Java que fornece infraestrutura para implementar funcionalidades de processamento de consultas. A biblioteca oferece componentes para acesso ao disco e um *framework* para indexação, utilizados para o desenvolvimento da solução, além de outras funcionalidades de alto-nível. A documentação da biblioteca XXL encontra-se disponível em http://code.google.com/p/xxl/.

Em outras palavras, a rede obtida para experimentação foi aquela resultante da projeção das observações das trajetórias sobre as vias. Esta projeção foi conseguida através da aplicação da técnica apresentada no Capítulo 2. A rede obtida com a projeção dos pontos tem um total de 80.961 vértices com distribuição de grau como apresentado no histograma da Figura 7.2.

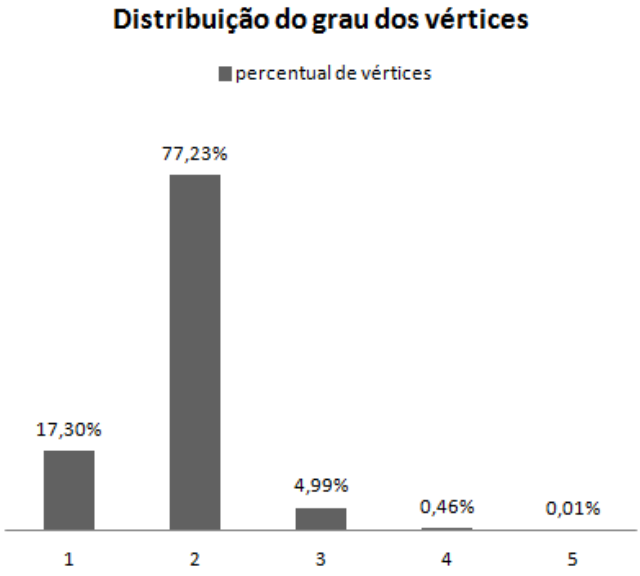


Figura 7.2: Distribuição do grau dos vértices da rede gerada.

O tamanho dos intervalos temporais foi de 15 minutos. Entretanto, o período do dia das observações capturadas variou para cada via, gerando períodos em que não há informações do tráfego para tais vias. Para estes casos, a velocidade média atribuída foi a velocidade referente ao período mais próximo cuja informação foi diretamente obtida dos dados de trajetórias. Métodos para estimar e inferir tais valores não foram o foco deste trabalho.

Pode-se observar na Figura 7.1 que a rede projetada é formada por agrupamentos de vértices (*clusters*) que internamente têm uma certa conectividade, mas existem poucos caminhos para ir de um agrupamento para o outro. Este fato também pode ser averiguado pela distribuição do grau dos nós. A grande maioria dos vértices tem apenas dois vizinhos, muitos outros apenas um e menos de 5% tem grau maior ou igual a três.

A falta de informação sobre o tráfego de certas vias em determinados intervalos de tempo, aliada à falta de informação sobre o tráfego de vias inteiras (não projetadas) acarretou os seguintes problemas: (i) pouca variabilidade do tráfego no decorrer do dia, que resultou por sua vez em sub-redes estacionárias; (ii) poucos caminhos possíveis entre uma grande quantidade de nós, que implica numa pequena quantidade de caminhos a serem descartados e (iii) desconectividade da rede, que possibilita a certos pontos de consulta não alcançar todos os possíveis pontos de interesse.

7.2 Resultados

Nos experimentos realizados 5% dos vértices da rede, em média, foram escolhidos como pontos de interesse. Os pontos de interesse foram escolhidos randomicamente, com distribuição de probabilidade uniforme. O valor de k variou entre 1 e 5. As escolhas destes parâmetros se deram pelos seguintes motivos. Uma menor densidade de pontos de interesse permite fazer com que a distância na rede entre um vértice e o seu ponto de interesse mais próximo aumente e que surja uma maior quantidade de caminhos possíveis entre estes. Com uma maior quantidade de caminhos aumenta a probabilidade de que o menor caminho passe por arestas com uma maior variabilidade no tráfego. Além disso, observou-se que para valores maiores de k e pontos de consulta escolhidos aleatoriamente, alguns pontos de consulta pertenciam a uma sub-rede desconexa, fazendo com que tais pontos não alcançassem k de pontos de interesse necessários para responder a consulta.

Os resultados apresentados mostram que o comportamento do algoritmo pode variar de acordo com o ponto de consulta. Mais precisamente, o comportamento do algoritmo varia de acordo com a região da rede em que o ponto de consulta está. Basicamente, o algoritmo TD-NE-A* funciona de maneira a excluir caminhos da rede. Entretanto, se não há muitas alternativas para os possíveis caminhos de um vértice ao próximo ponto de interesse, então não há caminho a ser excluído e a busca A* não minimiza o percentual da rede expandido. Cabe ressaltar que a rede projetada não apresenta a mesma estrutura da rede real.

Os gráficos das Figuras 7.3, 7.4, 7.5, 7.6 e 7.7 apresentam a distribuição do percentual de ganho, em relação ao número de páginas de disco acessadas, para cada valor de k. Para cada intervalo de percentual de ganho em operações de leitura (barras dos diagramas) é apresentado o percentual de execuções em que o algoritmo apresentou tal ganho (eixo y). Lembrando que cada execução diz respeito a um ponto de consulta randomicamente escolhido entre os vértices da rede.

Em todos os experimentos, foram executadas 1000 consultas para cada valor de k, somando um total de 5000 execuções por experimento. Para $k = 1$, o algoritmo TD-NE-A* ganhou em número de operações E/S em mais de 85% das execuções. Para esse mesmo valor de k, o algoritmo TD-NE-A* superou o concorrente executando 20% de operações E/S a menos em mais de 60% das execuções. A distribuição do percentual de ganho para k=1 pode ser vista na Figura 7.3. Para $k = 2$ o algoritmo apresentou ganho em mais de 75% das execuções, sendo este ganho maior que 20% em mais de 60% das execuções. A distribuição do percentual de ganho para k=2 pode ser vista na Figura 7.4. Algo semelhante ocorre com $k = 3$ (Figura 7.5). Para $k = 4$ e $k = 5$ (Figuras 7.6 e 7.7, respectivamente) o percentual de execuções em que se obteve ganho foi reduzido, mas ainda foi maior que 70%.

7.3 Conclusão

Neste capítulo foram apresentados os resultados dos experimentos feitos com dados reais. Os dados de tráfego foram obtidos a partir de um conjunto de observações de trajetórias.

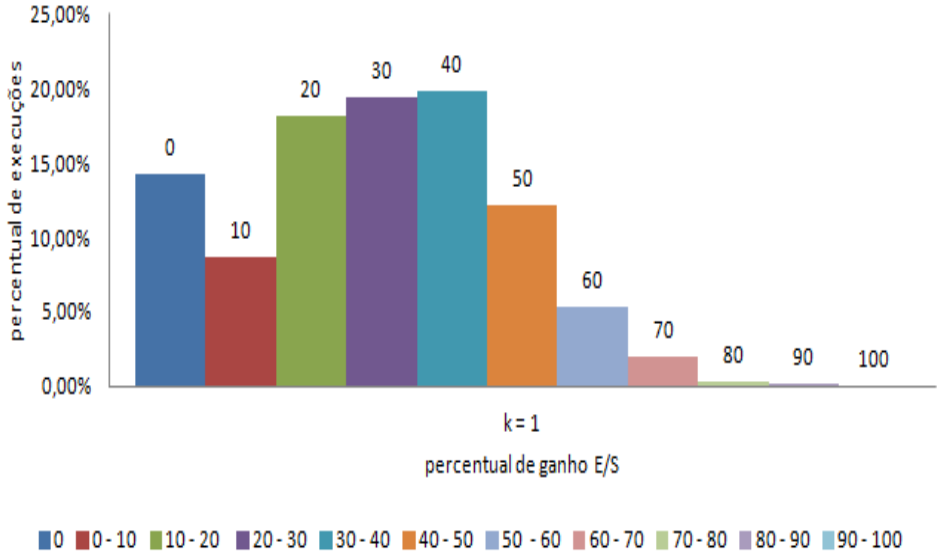


Figura 7.3: Distribuição do percentual de ganho para $k = 1$.

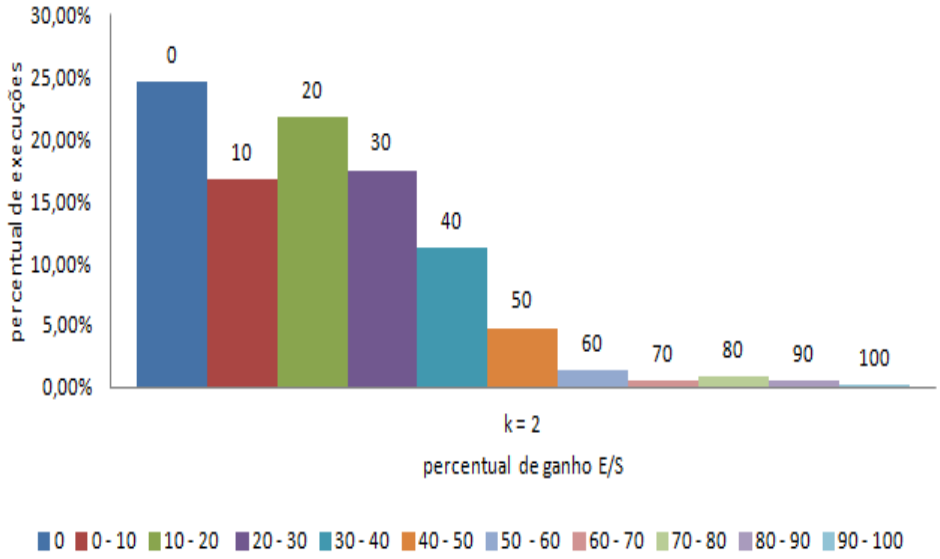


Figura 7.4: Distribuição do percentual de ganho para $k = 2$.

A rede utilizada foi parte da rede de Pisa onde estas trajetórias estavam projetadas. Os experimentos mostraram que para mais de 60% dos casos o algoritmo TD-NE-A* reduz o número de operações de E/S em mais de 20%, mesmo em redes com onde o grau dos vértices é pequeno.

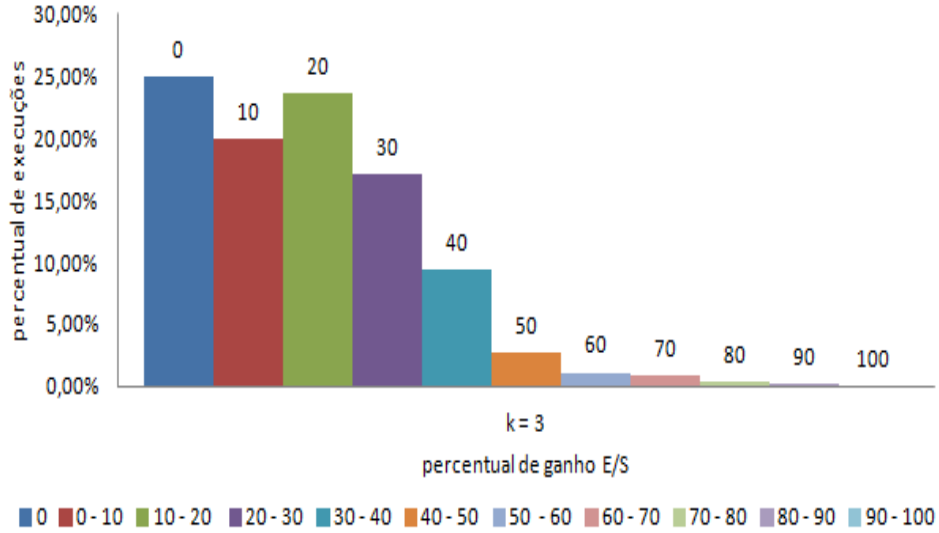


Figura 7.5: Distribuição do percentual de ganho para $k = 3$.

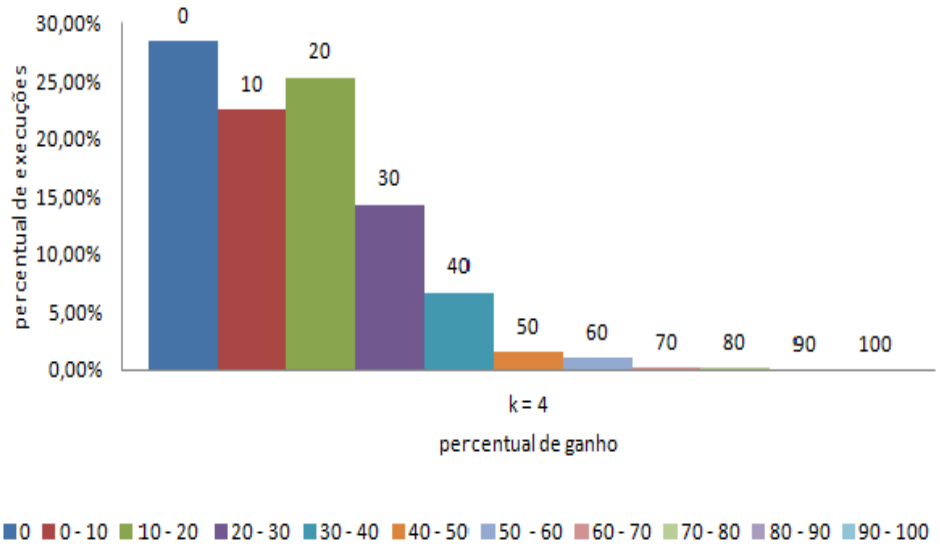


Figura 7.6: Distribuição do percentual de ganho para $k = 4$.

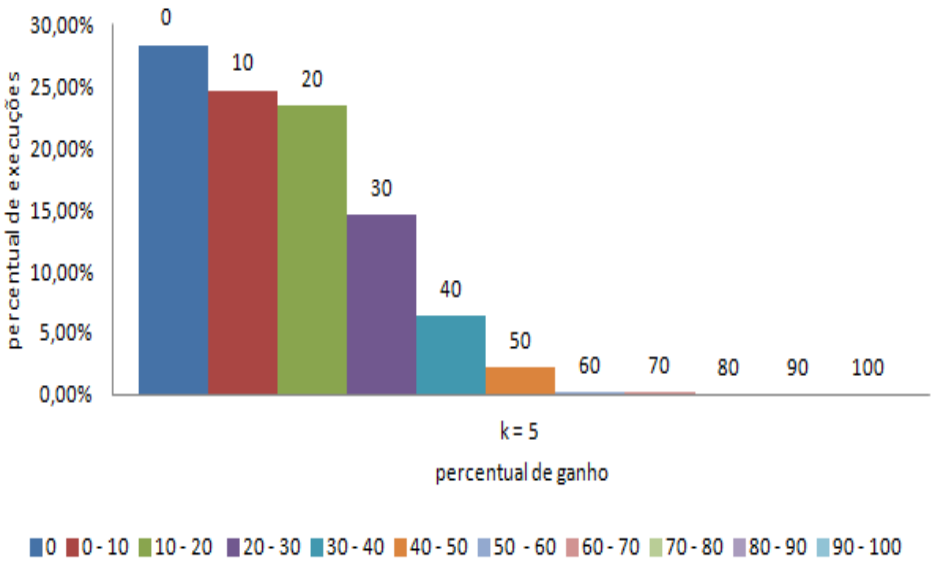


Figura 7.7: Distribuição do percentual de ganho para $k = 5$.

8 CONSIDERAÇÕES FINAIS

8.1 Conclusão

Nesta dissertação foi apresentada uma solução para o problema de processamento de consultas *k*NN em redes dependentes do tempo. A solução apresentada baseou-se na estratégia de busca A* para direcionar a expansão da rede para regiões que contêm pontos de interesse. A função heurística utilizada para o cálculo dos potenciais dos vértices é baseada na construção do grafo de limites inferiores, chamado $\underline{G}$. Para o cálculo da heurística, uma etapa pré-processamento é executada sobre o grafo $\underline{G}$. A etapa de pré-processamento executa em tempo polinomial.

Para dar suporte à execução do algoritmo, foi proposto um método de acesso e armazenamento. O método de acesso proposto fornece mecanismo para consultar a adjacência de um ponto da rede (vértice) e informações de tráfego das vias adjacentes. A estrutura do método de acesso é divida em três níveis: um nível para acesso do tempo, um para acesso do grafo e um nível para o armazenamento dos dados.

Foram feitos experimentos utilizando dados sintéticos e reais. Os experimentos mostraram que o algoritmo TD-NE-A* pode reduzir em até 50% o número de páginas de dados acessadas em relação ao algoritmo TD-NE. Além disso, mesmo em redes reais com pouca conectividade, o algoritmo mostrou reduzir o número de acessos em mais de 20% em relação ao algoritmo TD-NE para mais de 60% dos casos.

8.2 Trabalhos futuros

Como trabalhos futuros podem ser apontadas soluções para outras consultas em redes dependentes do tempo. Por exemplo, pode-se analisar o caso em que a consulta é processada utilizando, além dos dados históricos de tráfego, também as informações do tráfego atual. Dessa forma, o histórico de tráfego apontado pelo TDG serviria como um guia para os possíveis tempos de viagem e as informações do tempo real ponto a ponto, do caminho já percorrido, ajudariam a ajustar os valores estimados.

Analisar e explorar as preferências dos usuários no resultado da consulta também é um caminho a ser investigado. Por exemplo, os pontos de interesse que não se enquadram nas preferências do usuário poderiam ser descartados do resultado da consulta.

Outro problema a ser investigado é a análise dos dados de trajetória para inferir o tempo de viagem e construir redes dependentes do tempo levando em consideração problemas como o erro das observações de GPS e a falta de observações em alguns intervalos de tempo.

Por fim, em relação ao acesso aos dados, podem ser investigadas diferentes políticas de cache para o método de acesso proposto, além de outras estratégias de alocação dos dados nas páginas de disco.

REFERÊNCIAS BIBLIOGRÁFICAS

CHEN, D. et al. Approximate map matching with respect to the fréchet distance. In: *ALENEX*. [S.l.: s.n.], 2011. p. 75–83.

CINTRA P. ; TRASARTI, R. . C. L. . F. C. . M. J. A. F. A gravity model for speed estimation over road network. In: MOBILE DATA MANAGEMENT. *Proceedings of HuMoComp Workshop (in press)*. [S.l.], 2013.

COOKE, K. L.; HALSEY, E. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, v. 14, n. 3, p. 493–498, 1966. ISSN 0022-247X. Disponível em: <http://www.sciencedirect.com/science/article/pii/0022247X66900096>.

CRUZ, L. A.; NASCIMENTO, M. A.; MACÊDO, J. A. F. de. k-nearest neighbors queries in time-dependent road networks. *Journal of Information and Data Management*, v. 3, n. 3, p. 211–226, 2012.

DEMIRYUREK, U.; BANAEI-KASHANI, F.; SHAHABI, C. Efficient k-nearest neighbor search in time-dependent spatial networks. In: SPRINGER. *Database and Expert Systems Applications*. [S.l.], 2010. p. 432–449.

DEMIRYUREK, U. et al. Online computation of fastest path in time-dependent spatial networks. In: PFOSER, D. et al. (Ed.). *Advances in Spatial and Temporal Databases*. Springer Berlin Heidelberg, 2011, (Lecture Notes in Computer Science, v. 6849). p. 92–111. ISBN 978-3-642-22921-3. Disponível em: <http://dx.doi.org/10.1007/978-3-642-22922-0_7>.

DEMIRYUREK, U.; KASHANI, F. B.; SHAHABI, C. Towards k-nearest neighbor search in time-dependent spatial network databases. In: *Databases in Networked Information Systems*. Berlin, Heidelberg: [s.n.], 2010. p. 296–310.

DIJKSTRA, E. A note on two problems in connexion with graphs. *Numerische Mathematik*, Springer-Verlag, v. 1, p. 269–271, 1959. ISSN 0029-599X. Disponível em: <http://dx.doi.org/10.1007/BF01386390>.

ERWIG, M. The graph voronoi diagram with applications. *Networks*, v. 36, n. 3.

GEORGE, B.; KIM, S.; SHEKHAR, S. Spatio-temporal network databases and routing algorithms: a summary of results. In: *Proceedings of the 10th International Conference on Advances in Spatial and Temporal Databases*. Berlin, Heidelberg: [s.n.], 2007. p. 460–477.

GOLDBERG, A. V.; HARRELSON, C. Computing the shortest path: A* search meets graph theory. In: *Proceedings of the Sixteenth Snnual ACM-SIAM Symposium on Discrete algorithms*. Philadelphia, PA, USA: [s.n.], 2005. p. 156–165.

HUANG, X.; JENSEN, C.; SALTENIS, S. The islands approach to nearest neighbor querying in spatial networks. In: MEDEIROS, C. B.; EGENHOFER, M.; BERTINO, E. (Ed.). *Advances in Spatial and Temporal Databases*. Springer Berlin Heidelberg, 2005, (Lecture Notes in Computer Science, v. 3633). p. 73–90. ISBN 978-3-540-28127-6. Disponível em: <http://dx.doi.org/10.1007/11535331_5>.

JUNGLAS, I. A.; WATSON, R. T. Location-based services. *Commun. ACM*, ACM, New York, NY, USA, v. 51, n. 3, p. 65–69, mar. 2008. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1325555.1325568>.

KOLAHDOUZAN, M.; SHAHABI, C. Voronoi-based k nearest neighbor search for spatial network databases. In: . [S.l.: s.n.].

KRIEGEL, H.-P. et al. Proximity queries in large traffic networks. In: *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*. New York, NY, USA: ACM, 2007. (GIS '07), p. 21:1–21:8. ISBN 978-1-59593-914-2. Disponível em: <http://doi.acm.org/10.1145/1341012.1341040>.

KRIEGEL, H.-P. et al. Proximity queries in time-dependent traffic networks using graph embeddings. In: *Proceedings of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science*. New York, NY, USA: ACM, 2011. (CTS '11), p. 45–54. ISBN 978-1-4503-1034-5. Disponível em: <http://doi.acm.org/10.1145/2068984.2068993>.

LOU, Y. et al. Map-matching for low-sampling-rate gps trajectories. In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. New York, NY, USA: ACM, 2009. (GIS '09), p. 352–361. ISBN 978-1-60558-649-6. Disponível em: <http://doi.acm.org/10.1145/1653771.1653820>.

NANNICINI, G. et al. Bidirectional A* search for time-dependent fast paths. In: MCGEOCH, C. (Ed.). *Experimental Algorithms*. Springer Berlin Heidelberg, 2008, (Lecture Notes in Computer Science, v. 5038). p. 334–346. ISBN 978-3-540-68548-7. Disponível em: <http://dx.doi.org/10.1007/978-3-540-68552-4_25>.

NANNICINI, G. et al. Bidirectional A* search on time-dependent road networks. *Networks*, Wiley Subscription Services, Inc., A Wiley Company, v. 59, n. 2, p. 240–251, 2012. ISSN 1097-0037. Disponível em: <http://dx.doi.org/10.1002/net.20438>.

ORDA, A.; ROM, R. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *J. ACM*, ACM, New York, NY, USA, v. 37, n. 3, p. 607–625, jul 1990. ISSN 0004-5411. Disponível em: <http://doi.acm.org/10.1145/79147.214078>.

PAPADIAS, D. et al. Query processing in spatial network databases. In: *Proceedings of the 29th international conference on Very large data bases - Volume 29*. VLDB Endowment, 2003. (VLDB '03), p. 802–813. ISBN 0-12-722442-4. Disponível em: <http://dl.acm.org/citation.cfm?id=1315451.1315520>.

WAGNER, D.; WILLHALM, T. Speed-up techniques for shortest-path computations. In: *Proceedings of the 24th Annual Conference on Theoretical Aspects of Computer Science*. Berlin, Heidelberg: [s.n.], 2007. p. 23–36.

YUAN, J. et al. An interactive-voting based map matching algorithm. In: *Proceedings of the 2010 Eleventh International Conference on Mobile Data Management*. Washington, DC, USA: IEEE Computer Society, 2010. (MDM '10), p. 43–52. ISBN 978-0-7695-4048-1. Disponível em: <http://dx.doi.org/10.1109/MDM.2010.14>.

ZHENG, K. et al. Reducing uncertainty of low-sampling-rate trajectories. In: *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*. Washington, DC,

USA: IEEE Computer Society, 2012. (ICDE '12), p. 1144–1155. ISBN 978-0-7695-4747-3. Disponível em: <http://dx.doi.org/10.1109/ICDE.2012.42>.