UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

BRUNO SAVEGNAGO FAJARDO

Uma Extensão Nativa do SQL para Mineração de Trajetórias Semânticas

Trabalho de Graduação.

Prof. Dra. Vania Bogorny
Orientadora

Porto Alegre, novembro de 2008.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ANSI - American National Standards Institute
ATLaS - Aggregate & Table Language and System
BDG - Banco de Dados Geográfico
BNF - Backus-Naur Form
CB-SMoT - Clustering-Based Stops and Moves of Trajectories
DDL - Data Definition Language
DML - Data Manipulation Language
GPS - Global Positioning System
ISO - International Organization for Standardization
SGBD - Sistema Gerenciador de Banco de Dados
SMoT - Stops and Moves of Trajectories
SQL - Structured Query Language
TB - Trajetórias Brutas
TS - Trajetórias Semânticas
UDA - User Defined Aggregates
WEKA - Waikato Environment for Knowledge Analysis
WEKA-GDPM - WEKA-Geographic Data Preprocessing Module

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O objetivo principal deste trabalho é desenvolver uma extensão nativa do SQL para a mineração de trajetórias de objetos móveis. As trajetórias são representadas por *stops* e *moves*, e o processo contempla as etapas de pré-processamento, extração de padrões seqüenciais e pós-processamento. Através do método proposto, baseado em uma árvore de prefixos para a estruturação das seqüências de itens, o usuário pode definir diferentes granularidades espaciais e temporais. Essa transformação é fundamental no processo de descoberta de conhecimento sobre trajetórias, pois permite que diferentes padrões sejam extraídos a partir da mesma base de dados. Os padrões gerados são armazenados no próprio banco de dados, possibilitando que filtros sejam aplicados, visando o refinamento dos resultados obtidos. O processo de mineração pode ser realimentado e executado diversas vezes, convergindo para atingir os objetivos do usuário. A integração do algoritmo desenvolvido neste trabalho com os métodos de pré-processamento de trajetórias apresentados em (CHIECHELSKI, BOGORNY, 2008) resulta em uma linguagem completa para a mineração de dados espaço-temporais, mais especificamente sobre trajetórias semânticas. Tal abordagem não é encontrada na maioria das soluções existentes atualmente.

Palavras-chave: mineração de dados espaço-temporais, padrões seqüenciais, dados espaço-temporais, trajetórias semânticas, granularidade espacial, granularidade temporal, SQL, armazenamento de padrões, *stops* e *moves*.

A Native Extension of SQL for Semantic Trajectory Data Mining

ABSTRACT

The main purpose of this work is to develop a native extension of SQL for mining trajectories of moving objects. The trajectories are modeled as stops and moves, and the whole process covers the tasks of data preprocessing, sequential pattern extraction and post-processing. Through the proposed method, based on a prefix trie to represent sequences, the user can define different spatial and temporal granularities. Different granularities are fundamental in the spatio-temporal knowledge discovery process, because they provide a way to extract different patterns from the same data. The generated patterns are stored in the database, allowing the user to apply filters over the patterns for the refinement of the results. The mining process can be feeded and executed several times, according to the user's needs. The integration of the algorithm presented in this paper with the trajectories preprocessing methods implemented in (CHIECHELSKI, BOGORNY, 2008) results in a complete language for mining spatio-temporal data, more specifically for semantic trajectories. Such approach is not commonly found in most of the existing solutions.

Keywords: spatio-temporal data mining, sequential patterns, spatio-temporal data, semantic trajectories, space granularity, time granularity, SQL, patterns storing, stops and moves.

1. Introdução e Motivação

Dados sobre o deslocamento de objetos móveis têm se tornado muito comuns ultimamente, devido ao crescente uso de dispositivos eletrônicos capazes de capturar estas informações. Quantidades enormes de dados sobre trajetórias são armazenados por empresas com os mais diversos objetivos: estudo da migração de pássaros, tráfego urbano, comportamento de pessoas, etc.

Estas informações, apesar de abundantes, são pouco utilizadas para a obtenção de conhecimento útil. A mineração de dados espaço-temporais visa à extração de conhecimento sobre trajetórias para fins estratégicos, trabalhando sobre agrupamentos de dados para gerar padrões. Os dados espaço-temporais são mais complexos do que os dados tradicionais, pois se baseiam em medidas de espaço e tempo. Esta é uma área ainda pouco explorada, e conseqüentemente dispõe de poucas ferramentas adequadas.

Grande parte dos trabalhos na área de mineração de dados espaço-temporais propõem métodos baseados em dados brutos, geralmente representados na forma de pontos no espaço e instantes no tempo. Além disso, os métodos existentes focam somente na etapa de mineração, não cobrindo as etapas de pré-processamento e pós-processamento, necessárias para a extração de padrões interessantes e compreensíveis pelo usuário.

As trajetórias brutas possuem pouca ou nenhuma semântica agregada, tornando o processo de mineração e a interpretação dos padrões gerados para o usuário mais difíceis e demorados. Os dados brutos são utilizados somente para detectar padrões geométricos, como áreas densas ou similaridades entre trajetórias. O lado esquerdo da figura 1.1 ilustra um padrão de convergência, onde diversas trajetórias rumam para o mesmo ponto *A*. Caso estas trajetórias tivessem passado por um processo de agregação de semântica geográfica, seria possível detectar que a origem das trajetórias representa colégios de uma cidade, e que o ponto *A* representa um cinema, como ilustrado no lado direito da mesma figura. Desta forma, as trajetórias semânticas permitiriam extrair conhecimento não existente na simples representação geométrica das trajetórias.

Existem diversas maneiras de agregar semântica às trajetórias brutas. Geralmente, este processo é realizado na etapa de pré-processamento de uma mineração de dados. Em (CHIECHELSKI, BOGORNY, 2008), são implementados dois métodos que agregam semântica às trajetórias. Estes métodos foram definidos a partir das especificações teóricas existentes em (ALVARES *et al.*, 2007a) e (PALMA *et al.*, 2008).

O objetivo deste trabalho é elaborar um algoritmo para a extração de padrões seqüenciais sobre trajetórias semânticas, através de uma linguagem capaz de executar todas as tarefas necessárias para a mineração. Os métodos definidos em (CHIECHELSKI, BOGORNY, 2008) são utilizados para a geração das trajetórias semânticas, que servem como entrada do algoritmo implementado neste trabalho. A linguagem implementada é uma extensão nativa do SQL, onde é possível ao usuário configurar parâmetros que definam as granularidades dos dados a serem minerados,

bem como quais aspectos devem ser considerados na mineração.

Até o momento, existem somente dois trabalhos que permitem a extração de padrões de trajetórias em diferentes níveis de granularidade: o trabalho de Bogorny, Kuijpers e Alvares (BOGORNY, KUIJPERS, ALVARES, 2008), implementado na plataforma WEKA-GDPM (BOGORNY *et al.*, 2006) (Site do WEKA), e este trabalho de conclusão, implementado em SQL para dar suporte a bancos de dados espaciais.

Os padrões gerados pelo algoritmo são armazenados no banco de dados, possibilitando ao usuário refinar o processo de mineração diversas vezes, gerando, armazenando e consultando os padrões a cada ciclo. Esta característica praticamente não é encontrada em outras linguagens de mineração de dados, que geralmente resumem-se em apresentar os resultados para o usuário, não permitindo consultas sobre os padrões obtidos.

Os padrões seqüenciais representam informações sobre uma seqüência de fatos, permitindo definir relações entre pontos no espaço e no tempo. Por este motivo, estes padrões têm sido amplamente utilizados em vários domínios de aplicação. Vários algoritmos foram propostos para a geração dos padrões seqüenciais, como (AGRAWAL, SRIKANT, 1995), (PEI *et al.*, 2001) e (AYRES *et al.*, 2002), mas nenhum deles para trajetórias. Além disso, a maioria destes algoritmos é implementada em linguagens como C ou Java, ou de forma não padronizada.

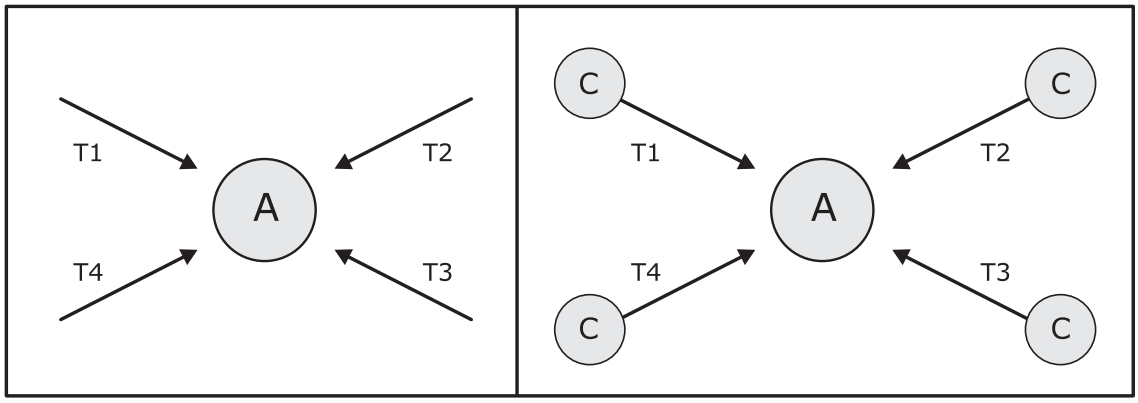

Figura 1.1: Lado esquerdo: um padrão de convergência geométrica. Lado direito: o mesmo padrão de convergência com trajetórias semânticas

Este trabalho foi inspirado por (WANG, ZANIOLO, 2003) e (BOGORNY, KUIJPERS, ALVARES, 2008). Zaniolo (WANG, ZANIOLO, 2003) mostrou que é possível utilizar SQL para implementar regras de associação. Em (BOGORNY, KUIJPERS, ALVARES, 2008), foi proposta uma linguagem de mineração onde padrões seqüenciais são implementados em uma linguagem qualquer e acoplados ao SQL. O método proposto neste trabalho é totalmente implementado em SQL, mais especificamente na sua revisão SQL:1999. Isto traz como benefício a compatibilidade com uma enormidade de sistemas e plataformas, visto que o SQL é um padrão reconhecido e amplamente utilizado em SGBDs comerciais e de código aberto.

O trabalho está estruturado da seguinte forma: o capítulo 2 apresenta os

conceitos básicos necessários para o entendimento do processo. O capítulo 3 introduz conceitualmente o algoritmo utilizado, definido totalmente em SQL, além de apresentar uma simulação de mineração de dados espaço-temporais. O capítulo 4 mostra alguns experimentos e testes diversos realizados com o algoritmo, além de mostrar como os padrões armazenados podem ser consultados, refinando o processo de mineração. O capítulo 5 conclui este trabalho.

2. Conceitos Básicos

2.1 Mineração de dados e descoberta de conhecimento

Atualmente, devido à queda no custo do armazenamento dos dados e a rápida automatização das empresas, a quantidade de dados armazenados em bases de dados cresce a uma velocidade muito grande. Estes dados estão presentes nos mais diversos ramos, como financeiro, médico, industrial, comercial, científico, etc.

Através das informações coletadas, é possível obter conhecimento significativo para o negócio ou aplicação em questão. Por exemplo, uma loja poderia descobrir os perfis de compras dos seus clientes, e utilizar esta informação em promoções futuras; ou um hospital poderia prever diagnósticos e antecipar tratamentos baseado no histórico de seus pacientes. Infelizmente, tais informações não são obtidas diretamente, visto que o ser humano não é capaz de analisar, processar e extrair relações significativas de tamanha quantidade de informações. Na maioria das vezes, padrões valiosos são desperdiçados, pois as organizações não sabem ou não conseguem extrair conhecimento útil de seus bancos de dados.

A área de Mineração de Dados surgiu no final da década de 80 justamente para tratar esses problemas, focando na extração de conhecimento a partir de grandes volumes de dados, utilizando meios computacionais. A área é multidisciplinar, tendo estreitas relações com as áreas de Banco de Dados, Aprendizado de Máquina, Estatística, Recuperação de Informação e Computação Paralela e Distribuída. Uma definição mais abrangente é dada abaixo:

"Mineração de Dados é o processo de explorar grandes quantidades de dados à procura de padrões consistentes, como regras de associação ou seqüências temporais, para detectar relacionamentos sistemáticos entre variáveis, detectando assim novos subconjuntos de dados" (Mineração de Dados na Wikipédia).

2.1.1 Dados, informação e conhecimento

Os conceitos de dado, informação e conhecimento estão interligados, e estão presentes em qualquer operação de mineração de dados.

Segundo (REZENDE, PUGLIESI, MELANDA, 2003), os *dados* são elementos puros, quantificáveis sobre um determinado evento. Dados são fatos, números, texto ou qualquer mídia que possa ser processada pelo computador. Hoje em dia, as organizações estão acumulando vastas e crescentes quantidades de dados em diferentes formatos e em diferentes tipos de repositórios. O dado, por si só, não oferece embasamento para o entendimento da situação.

A *informação* consiste no dado analisado e contextualizado. Envolve a interpretação de um conjunto de dados, ou seja, a informação é constituída por associações ou relações que todos aqueles dados acumulados podem proporcionar. O

grande desafio da década de 80 foi migrar os dados para informações, por meio do desenvolvimento de sistemas de informação.

A informação pode gerar *conhecimento* que ajude na análise de padrões históricos para prever casos futuros, pelo menos no contexto das variáveis que estão sendo envolvidas na análise. Como um exemplo, podemos considerar as informações adquiridas a partir de dados de transações comerciais, que podem ser analisadas a fim de serem estabelecidas ações mais lucrativas a serem executadas no futuro.

Enquanto que a informação é descritiva, o conhecimento é utilizado fundamentalmente para fornecer uma base de previsão com um determinado grau de certeza. O conhecimento refere-se à habilidade de criar um modelo mental que descreva o objeto e indique as ações a implementar e as decisões a tomar (REZENDE, PUGLIESI, MELANDA, 2003). Uma decisão é o uso explícito de um conhecimento.

A compreensão, análise e síntese, necessárias para a tomada de decisões inteligentes, são realizadas a partir do nível do conhecimento. Assim, é fundamental que se mantenha a coerência dos dados que estão armazenados nos diferentes repositórios e das informações nos diferentes níveis. A figura 2.1 representa os conceitos discutidos nesta seção.

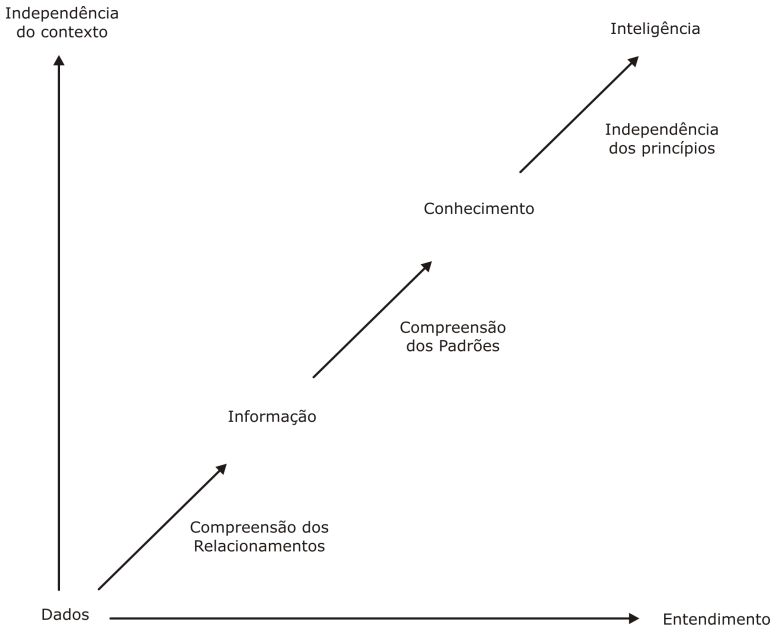

Figura 2.1: Relação de dados, informação e conhecimento. Imagem retirada de (REZENDE, PUGLIESI, MELANDA, 2003)

2.1.2 O processo de descoberta de conhecimento

O processo de extrair conhecimento a partir de bases de dados não é automático, mas sim interativo, e requer a intervenção de um humano a cada ciclo para conduzir a mineração até que os padrões apareçam. Todo o processo pode ser separado em três grandes etapas: pré-processamento, extração de padrões e pós-processamento. Além disso, pode-se incluir uma etapa anterior à mineração propriamente dita, constituída do

conhecimento do domínio e a identificação do problema, e uma etapa posterior, que se refere à utilização do conhecimento obtido. A figura 2.2 descreve este processo.

O processo inicia com o conhecimento do domínio da aplicação (I), considerando quais os objetivos da mineração e os dados a serem utilizados. O sucesso do processo de extração de conhecimento depende, em parte, da participação dos especialistas do domínio da aplicação no fornecimento de conhecimento sobre o domínio e apoio aos analistas em sua tarefa de encontrar os padrões.

A etapa de pré-processamento (II) é responsável pelo tratamento, limpeza e redução do volume de dados, visto que na maioria das vezes os dados não se encontram prontos para a extração do conhecimento. Os dados devem ser preparados de forma a atender os requisitos da aplicação.

A extração de padrões (III) compreende a escolha dos recursos de mineração de dados a serem utilizados, de modo a cumprir os objetivos estabelecidos quando da identificação do problema. É preciso escolher o algoritmo a ser utilizado, e repetir o processo quantas vezes forem necessárias, até que os parâmetros sejam apropriados para a correta geração dos padrões.

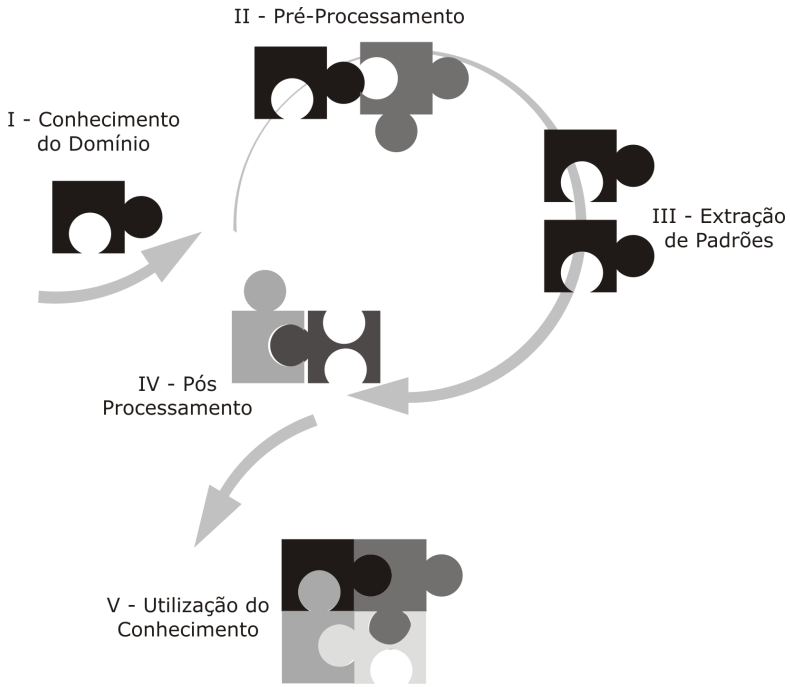

Figura 2.2: Representação do processo de extração de conhecimento. Imagem inspirada em (REZENDE, PUGLIESI, MELANDA, 2003)

Depois da obtenção dos primeiros padrões, entra em cena a etapa de pós-processamento (IV). O conhecimento obtido pode ser utilizado para a resolução de problemas (V). Se o resultado não foi satisfatório, o especialista faz novas consultas sobre os padrões ou reinicia todo o processo de mineração.

2.2 Mineração de dados espaço-temporais

Diferentemente da mineração de dados tradicional, que trabalha com dados convencionais, como registros de compras em uma loja, a mineração de dados espaço-temporais tem como objetivo extrair relações sobre dados envolvendo espaço e tempo. Os recentes avanços na tecnologia de satélites e dispositivos móveis, como aparelhos de GPS, telefones celulares e redes de sensores, têm facilitado a coleta de grandes quantidades de dados espaço-temporais. Estes dados geralmente são representados em forma de trajetórias, formadas por diversos pontos na forma (tid, x, y, t), onde tid é o identificador da trajetória e (x, y) são coordenadas geográficas que correspondem a um lugar no espaço num instante de tempo t.

Um dos grandes problemas envolvendo mineração e extração de conhecimento de trajetórias diz respeito ao alto nível de complexidade dos dados envolvidos, tornando o entendimento difícil para o usuário. Como as trajetórias são originalmente dados brutos, contendo pouca ou nenhuma semântica, os algoritmos atuais conseguem somente gerar padrões geométricos, que podem não ter muita utilidade em aplicações reais (BOGORNY, KUIJPERS, ALVARES, 2008) (ALVARES *et al.*, 2007a). Na seção 2.2.1, são apresentados mais detalhes sobre trajetórias e padrões geométricos.

Como discutido em (ALVARES *et al.*, 2007b), a transformação de trajetórias brutas em trajetórias semânticas, que consiste no processo de adição de informações geográficas às trajetórias, é essencial para a extração de conhecimento. Além disso, deve ser possibilitado ao usuário a definição de diferentes níveis de granularidade (BOGORNY, KUIJPERS, ALVARES, 2008) dos dados, conforme será discutido na seção 2.2.4. Estas transformações fazem parte da etapa de pré-processamento de dados espaço-temporais.

2.2.1 Trajetórias

Uma trajetória consiste no caminho percorrido por um dado objeto no espaço durante seu movimento, ou, de forma mais aplicada a este trabalho, consiste de uma seqüência de pontos, definidos em termos de coordenadas no espaço (x, y) e no tempo (t), que descrevem o percurso de alguma pessoa ou artefato.

O movimento de um objeto qualquer não necessariamente é classificado como uma trajetória, pois para tal, o objeto precisa deslocar-se de um ponto inicial a um ponto final com um objetivo definido. Também é preciso que os horários de início e fim da trajetória sejam demarcados, de modo a delimitar o trecho de interesse. Em outras palavras, uma trajetória é o segmento espaço-temporal do caminho percorrido por um objeto (SPACCAPIETRA *et al.*, 2007) (SPACCAPIETRA *et al.*, 2008).

Uma pessoa saindo de casa para mais um dia de trabalho, possivelmente com um telefone celular ou com qualquer outro dispositivo capaz de localizar-se, o vôo de um avião, o fluxo de carros em uma cidade grande, enfim, todos são exemplos reais de eventos do cotidiano que geram trajetórias. Tais trajetórias são coletadas há anos, de diversas formas, na sua grande maioria em formatos brutos, ou seja, que não agregam qualquer informação semântica aos dados, tornando seu uso extremamente difícil.

2.2.1.1 Trajetórias Brutas

As trajetórias brutas são os dados comuns, crus, coletados diretamente de dispositivos eletrônicos. O formato geral de uma trajetória bruta resume-se a uma seqüência de pontos de amostra, definidos em termos de coordenadas (x, y, t), que definem o comportamento de uma entidade durante certo intervalo de tempo. A figura 2.3 representa, de forma geral, uma trajetória ao longo do tempo.

Figura 2.3: Trajetória bruta representada por diversos pontos no espaço

O grande problema de trajetórias desse tipo diz respeito à sua falta de significado prático para as pessoas, pois a semântica agregada é praticamente nula. Além disso, os processos de consulta e de mineração sobre estes dados tornam-se extremamente complexos, visto que operações de intersecção espaciais e temporais são necessárias em qualquer tipo de análise.

De qualquer forma, a grande maioria dos métodos e algoritmos de mineração de dados espaciais ou espaço-temporais existentes atualmente utilizam as trajetórias brutas, gerando padrões geométricos como resultado. O trabalho de Laube, em 2002, foi um dos pioneiros na área de mineração de dados espaço-temporais gerados por dispositivos móveis (LAUBE, IMFELD, 2002), e que tem sido a base de muitos outros trabalhos nesta área. Ele define um padrão para trajetórias de comportamento similar, baseado na direção do movimento e na mudança de direção. Um padrão deve conter um número mínimo de trajetórias que se movimentam na mesma direção.

Outro tipo de padrão de trajetórias, introduzido por (HWANG *et al.*, 2005), considera um conjunto de trajetórias próximas umas às outras independente de direção. Em outras palavras, a distância entre as trajetórias precisa ser inferior a uma dada distância *minDist* e por um intervalo mínimo de tempo *minTime*. Os padrões freqüentes são gerados da mesma forma como no algoritmo Apriori (AGRAWAL, SRIKANT, 1994) (Algoritmo Apriori na Wikipédia), um dos mais conhecidos algoritmos para mineração de regras de associação.

2.2.1.2 Trajetórias Semânticas

Tendo em vista as desvantagens referentes às trajetórias brutas citadas anteriormente, as trajetórias semânticas apresentam a vantagem de serem mais facilmente compreendidas, pois as coordenadas comuns são acompanhadas de informações geográficas sobre cada ponto da amostra. A figura 2.4 mostra a mesma trajetória da imagem 2.3, mas desta vez com informações semânticas agregadas.

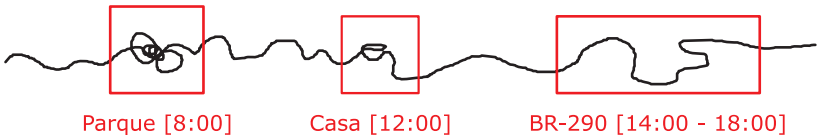

Figura 2.4: Trajetória semântica, evidenciando o movimento de uma pessoa ao longo do dia

Naturalmente, a forma de consultar uma base de dados a fim de recuperar dados relevantes sobre trajetórias torna-se muito mais simples com trajetórias semânticas, uma vez que o usuário não precisa mais especificar quais pontos são de seu interesse e executar uma operação de intersecção entre eles, basta que ele pesquise diretamente sobre os dados mais abstratos (BOGORNY, KUIJPERS, ALVARES, 2008). A tabela 2.1 mostra dois exemplos de consultas, uma sobre dados brutos e outra sobre dados contendo semântica. As duas consultas têm como objetivo selecionar os pontos importantes pelos quais uma determinada trajetória *A* passou, considerando hotéis e restaurantes de uma cidade como exemplo.

É possível notar que a consulta sobre trajetórias semânticas torna-se extremamente simples e de entendimento mais fácil por parte do usuário. Além da facilidade em consultar os dados, o processo de mineração também é facilitado. Com trajetórias semânticas, é possível extrair conhecimento que seria impossível caso dados brutos fossem utilizados. O exemplo utilizado na motivação deste trabalho, no capítulo 1, representa bem esta vantagem das trajetórias semânticas sobre as trajetórias brutas.

Tabela 2.1: Exemplos de consultas arbitrárias sobre trajetórias brutas e trajetórias semânticas. Retirado de (BOGORNY, KUIJPERS, ALVARES, 2008) e (ALVARES *et al.*, 2007a)

Consulta sobre trajetórias brutas

```
SELECT h.name FROM trajectory t, hotel h WHERE t.id = 'A' AND
intersects(t.movingpoint.geometry, h.geometry)
UNION
SELECT r.name FROM trajectory t, restaurant r WHERE t.id = 'A' AND
intersects(t.movingpoint.geometry, r.geometry)
```

Consulta sobre trajetórias semânticas

```
SELECT place FROM semanticTrajectory WHERE id = 'A'
```

2.2.2 Stops e Moves

O conceito de *stops* e *moves*, introduzido em (SPACCAPIETRA *et al.*, 2007) e (SPACCAPIETRA *et al.*, 2008), é a base para a agregação de semântica às trajetórias brutas discutidas na seção anterior, facilitando funções de pré-processamento e a extração de padrões significativos. Basicamente, *stops* são pontos importantes da trajetória do ponto de vista da aplicação, onde o objeto permanece por um determinado intervalo de tempo. Os *moves* são sub-trajetórias entre dois *stops* consecutivos.

De acordo com o trabalho de Spaccapietra (SPACCAPIETRA *et al.*, 2008), os

stops são definidos de acordo com as seguintes características:

> - Um *stop* [*tbeginstopx*; *tendstopx*] é parte de uma trajetória, de modo que o usuário tenha explicitamente definido o ponto de início *tbeginstopx* e de término *tendstopx*.
>
> - A duração do *stop* é um intervalo de tempo não nulo, e o objeto da trajetória não se desloca para fora dos limites do *stop*.
>
> - Todos os *stops* são temporalmente disjuntos.

Para uma aplicação que pretende monitorar o deslocamento de uma pessoa ao longo de seu dia, os *stops* podem ser a casa da pessoa, o seu local de trabalho, ou o restaurante onde almoça. Já para uma aplicação de monitoramento da migração de aves, por exemplo, os *stops* podem ser regiões de um país ou de um continente, onde o pássaro pára para se alimentar. Conseqüentemente, os *stops* podem assumir diferentes formas, dependendo do contexto da aplicação que irá utilizar os dados. Estas diferentes formas são chamadas de *spatial features*, e geralmente são armazenadas em arquivos separados ou em diferentes tabelas de um banco de dados.

É importante notar que um determinado ponto de parada em uma trajetória não necessariamente configura um *stop*, ou seja, um *stop* físico nem sempre é um *stop* conceitual. Os *stops* são definidos pela aplicação, e podem assumir diferentes granularidades espaciais. Por exemplo, quando um vendedor pára em uma lanchonete para tomar um café, este *stop* não é relevante para sua empresa, e sim os pontos em que ele permanece parado visitando clientes. De forma similar, a definição para *move* é introduzida no trabalho (SPACCAPIETRA *et al.*, 2008):

> - Um *move* é parte de uma trajetória, e é delimitado por duas extremidades que representam *stops*, ou pelo ponto inicial da trajetória e o primeiro *stop*, ou pelo último *stop* e pelo ponto final da trajetória, ou por toda a extensão da trajetória.
>
> - A duração do *move* é um intervalo de tempo não nulo, e o *move* é composto somente por uma linha espaço-temporal, definida pela trajetória e pelos seus instantes de começo e fim.

Atualmente, existem dois métodos principais para a geração de *stops* e *moves*. Estes métodos são aplicados às trajetórias brutas para agregar semântica geográfica às mesmas como uma etapa de pré-processamento, podendo então estes dados semânticos ser utilizados em aplicações de mineração de dados.

O primeiro método, chamado de SMoT (*Stops and Moves of Trajectories*), definido em (ALVARES *et al.*, 2007a), verifica se cada ponto de uma trajetória *T* intercepta a área geométrica de um candidato a *stop*, que é um objeto geográfico qualquer. Em caso afirmativo, o algoritmo verifica a duração da intersecção, que deve ser igual ou superior ao limite de tempo especificado para que um trecho da trajetória seja considerado um *stop*. Caso o ponto satisfaça as duas condições anteriores, ele é armazenado como um novo *stop*. Os *moves* são derivados naturalmente, pois basicamente são os trechos existentes entre dois *stops*.

O segundo método é uma alternativa ao primeiro, chamado de CB-SMoT (*Clustering-Based Stops and Moves of Trajectories*), e foi inicialmente proposto em (PALMA *et al.*, 2008). Enquanto o algoritmo anterior procura por intersecções entre pontos da trajetória e as áreas de interesse da aplicação (*stops* candidatos), este método é baseado em clusterização espaço-temporal, levando em consideração a velocidade das trajetórias. Dito em outras palavras, o algoritmo considera pontos interessantes de uma trajetória os trechos em que a velocidade da mesma é reduzida. Como exemplo, podemos citar uma aplicação de controle de tráfego urbano, em que a velocidade dos automóveis próximos a semáforos ou congestionamentos seria reduzida, fazendo com que o algoritmo identificasse possíveis *stops* nestas áreas.

As trajetórias semânticas utilizadas neste trabalho são geradas a partir dos algoritmos SMoT e CB-SMoT, implementados em (CHIECHELSKI, BOGORNY, 2008), para o banco de dados PostGIS, que também é utilizado neste trabalho.

2.2.3 Tipos de padrões de trajetórias

Existem diversos tipos de padrões que podem ser extraídos de uma base de dados de trajetórias, potencialmente em aplicações diferentes. Desde regras de associação simples, padrões geométricos, padrões freqüentes e padrões seqüenciais, sendo os últimos o foco deste trabalho.

As regras de associação possuem uma forma geral X => Y, indicando que caso ocorra o item X em uma transação (ou trajetória), muito provavelmente o item Y também ocorra. Uma regra possui um suporte, definido pelo *número de transações com X e Y* dividido pelo *número total de transações*, e uma confiança, definida pelo *número de transações com X e Y* dividido pelo *número de registros com X*. Este conceito foi inicialmente proposto em (AGRAWAL, IMIELINSKI, SWAMI, 1993) para a descoberta de relações entre produtos vendidos em uma loja, sendo base para a tomada de decisão em empresas deste ramo. Um exemplo deste tipo de regra de associação seria a relação {cebola, batata} => {bife}, indicando que as pessoas que compram cebolas e batatas provavelmente também compram bifes. Em trajetórias, uma regra seria a relação {trabalho [manhã], restaurante [noite]} => {casa [noite]}.

Especificamente em bases de dados de trajetórias, padrões geométricos são possíveis de ser obtidos, como o exemplo da figura 1.1, que demonstra um padrão de convergência entre diferentes trajetórias. Os padrões geométricos são normalmente extraídos com base no conceito de regiões densas ou similaridade de trajetórias (BOGORNY, KUIJPERS, ALVARES, 2008). Outro tipo de padrão geométrico é o padrão de recorrência, onde diversas trajetórias passam pelo mesmo ponto *A* no espaço. A figura 2.5 ilustra este tipo de padrão.

Os padrões freqüentes e seqüenciais são similares, e muito comuns em aplicações envolvendo dados espaço-temporais. Um padrão freqüente indica que um conjunto de lugares é encontrado em pelo menos *n* trajetórias, independente da ordem dos lugares, tendo assim suporte mínimo. Já um padrão seqüencial considera a ordem entre os lugares do conjunto. Uma seqüência do tipo {1, 2, 3} é um padrão seqüencial somente se os elementos do conjunto aparecem nesta ordem em pelo menos *n* trajetórias, independentemente da contigüidade dos mesmos.

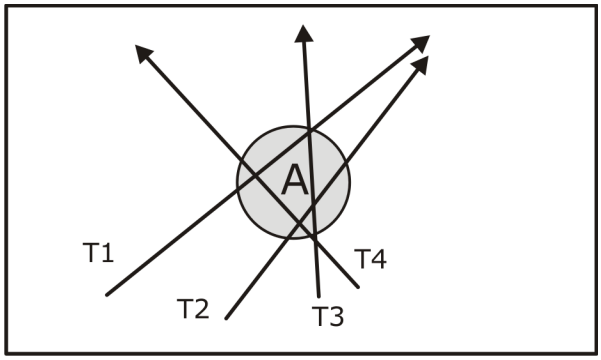

Figura 2.5: Padrão geométrico de recorrência

Os padrões seqüenciais gerados pelo processo de mineração descrito neste trabalho são compostos por lugares, chamados de “itens”, que representam trechos das trajetórias sendo mineradas. A formação dos itens depende dos parâmetros fornecidos pelo usuário, como detalhado no capítulo 3, mas em geral são definidos por granularidades temporais e espaciais. Desta forma, itens poderiam ser *hospitais*, *bairros* ou *cidades* inteiras. Considerando-se o tempo, possíveis itens seriam *escolas ao meio-dia*, ou *shopping center Iguatemi das 17h às 21h*.

Uma seqüência de itens que ocorrem em um número mínimo de trajetórias definido pelo usuário, ou seja, que possua um suporte mínimo, são considerados padrões seqüenciais. Na tabela 2.2, são apresentados três exemplos de trajetórias de pessoas ao longo de um dia.

Tabela 2.2: Exemplos de trajetórias de pessoas ao longo de um dia

Trajetória	**Seqüência de itens**
A	Bairro Partenon, Campus UFRGS, Bairro Centro, Parque da Redenção
B	Escola, Campus UFRGS, Parque da Redenção, Shopping Iguatemi
C	Bairro Passo D'areia, Campus UFRGS, Shopping Iguatemi, Shopping Total

Considerando-se as trajetórias da tabela 2.2 e um suporte mínimo de 50%, é possível dizer que um padrão seqüencial ocorre na seqüência de itens *Campus UFRGS* e *Parque da Redenção*, pois pelo menos 50% das trajetórias apresentam esta seqüência de itens, especificamente neste caso as trajetórias *A* e *B*. O mesmo acontece com as trajetórias *B* e *C*, que apresentam o padrão *Campus UFRGS* para *Shopping Iguatemi*.

O trabalho de Zaniolo (WANG, ZANIOLO, 2003) apresenta o problema de minerar padrões freqüentes sobre uma base de dados de transações, de forma similar ao trabalho de Agrawal (AGRAWAL, SRIKANT, 1995), que apresenta o problema de minerar padrões seqüenciais sobre transações. Apesar de não lidarem com dados espaço-temporais, os métodos e idéias descritos pelos autores serviram de inspiração para o algoritmo desenvolvido neste trabalho, apresentado no capítulo 3.

2.2.4 Níveis de granularidade e Hierarquia de Conceito

Tratando-se de etapas de pré-processamento e transformação de dados para um processo de mineração, um aspecto importante diz respeito à granularidade da informação a ser considerada. O modelo de dados para trajetórias utilizado neste trabalho, baseado em *stops* e *moves*, por sua natureza, pode ser parametrizado em termos de granularidades temporais e granularidades espaciais.

A granularidade baseada no tempo objetiva a seleção dos pontos de interesse com base em intervalos de tempo definidos pelo usuário. Pode ser do interesse da aplicação minerar padrões em trajetórias que ocorreram durante o turno da tarde, ou especificamente em termos de horários, dentro de um intervalo iniciando às 8h e encerrando às 10h15. Estes exemplos são mostrados na tabela 2.3.

Diferentes granularidades espaciais para *stops* também podem ser definidas, permitindo que o processo de mineração trabalhe sobre diferentes visões do espaço, conforme a necessidade do usuário. Existem dois níveis de granularidades espaciais que são pré-definidos: *feature instance* e *feature type*, ou, instâncias e tipos. Ao alterar-se a granularidade dos *stops* para uma determinada consulta, automaticamente são alteradas as granularidades dos *moves*, pois um *move* é formado por dois *stops* consecutivos.

Tabela 2.3: Exemplos de diferentes granularidades temporais

Intervalo das 8h às 10h15 [08:00-10:15]
Turno da tarde [12:00-18:00]

Feature instances são ocorrências específicas de locais, como o Hotel Sheraton, ou o Parque Moinhos de Vento. *Feature types* são locais genéricos, ou tipos de locais, como Hotéis ou Parques. A partir desses dois níveis básicos, é possível criar regras para obter a granularidade desejada na consulta. Alguns exemplos são mostrados na tabela 2.4.

Tabela 2.4: Exemplos de diferentes granularidades espaciais, agindo sobre *stops*

Bairros de uma cidade TYPE = 'bairro'
Um determinado restaurante TYPE = 'restaurante' AND INSTANCE = 'Churrascaria Na Brasa'
Rios ou lagos de um território TYPE = 'rio' OR TYPE = 'lago'

Estes parâmetros são configuráveis pelo usuário, que molda a execução da mineração às suas necessidades. As granularidades não agem como restrições, eliminando dos resultados da mineração os itens que não se enquadram nos seus limites, mas sim, servem para agregar itens em diferentes proporções de espaço e tempo. A seção 3.3 apresenta detalhadamente a forma de utilização das granularidades no método desenvolvido neste trabalho.

Outra forma de agregar os *stops* no espaço é através da utilização de uma hierarquia de conceito, como demonstrado na figura 2.6.

Figura 2.6: Hierarquia de conceito

Numa hierarquia desse tipo, o nível mais alto inicia em 0 (zero). Conseqüentemente, as folhas da estrutura representam o nível mais baixo. Na figura 2.6, o nível 0 (zero) representa o *feature type* da granularidade, enquanto o nível *n* representa as *feature instances*. Através desta estrutura, diversas granularidades espaciais podem ser utilizadas em conjunto. Um exemplo de utilização deste recurso é minerar uma base de dados com granularidade espacial padrão configurada como *feature type*, e informar que todos os *stops* do tipo *Road* utilizarão o nível 1 (um) da hierarquia de conceito, ou seja, serão todos considerados como instâncias de *Road*.

As hierarquias de conceito são apresentadas como recursos possíveis de serem utilizados em uma mineração de dados espaço-temporais, embora não sejam utilizadas neste trabalho.

2.3 SQL

O SQL, ou *Structured Query Language* (Linguagem de Consulta Estruturada), é uma linguagem de consulta declarativa para bancos de dados relacionais, fortemente inspirada na álgebra relacional (SQL na Wikipédia). A linguagem nasceu dentro dos laboratórios da IBM no início dos anos 70, e devido a sua simplicidade e objetividade ganhou popularidade no meio, tendo sido originadas diversas versões e extensões por fabricantes de SGBDs e de software em geral. Em 1986, a ANSI reuniu estas diversas extensões da linguagem e desenvolveu um padrão, sendo seguida pela ISO em 1987. Depois da padronização, a linguagem sofreu revisões em 1989 (SQL-89), 1992 (SQL-92), 1999 (SQL:1999 ou SQL3), 2003 (SQL:2003), e mais recentemente em 2006 (SQL:2006), tendo sido adicionadas diversas novas construções e recursos à linguagem ao longo dos anos (DATE, 2003).

O SQL, no contexto deste trabalho, desempenha um papel fundamental. Todo o algoritmo de mineração de padrões seqüenciais foi implementado utilizando-se esta linguagem, mais especificamente em sua versão SQL3, no banco de dados PostgreSQL (Site oficial do banco de dados PostgreSQL). Devido à padronização da linguagem, o algoritmo implementado neste trabalho pode ser executado sob qualquer banco de dados que suporte a linguagem SQL, proporcionando portabilidade entre praticamente todas as plataformas e soluções existentes hoje no mercado.

O SQL3 foi desenvolvido e projetado durante 7 anos (a previsão inicial girava em torno de 3 a 4 anos) para ser o sucessor do SQL-92, padrão utilizado na época (EISENBERG, MELTON, 1999). Nesta nova versão da linguagem, aprimorada e estendida, o foco principal resumiu-se na incorporação de recursos de orientação a objetos, que futuramente tornariam-se a base de diversos SGBDs Objeto-Relacionais, como o ORACLE8, Informix Universal Server, DB2 Universal Database da IBM, etc.

Apesar do foco em orientação a objetos, diversos recursos considerados de natureza "relacional" também foram adicionados ao padrão. Novos tipos de dados (*Blob*, *Boolean*, *Array* e *Row*), novos conceitos semânticos (atualizações de *views*, consultas recursivas) e o conceito de *Role* (papel, perfil), possibilitando uma maior segurança dos dados foram adicionados. Mas um dos mais importantes conceitos introduzidos pelo padrão são os *User Defined Aggregates* (UDA), descritos em detalhes na próxima seção. O conceito de UDA acabou ficando fora da versão final do padrão SQL3, mas estava presente em propostas iniciais da linguagem, além de ser suportada por diversos sistemas comerciais da atualidade.

2.3.1 Agregações Definidas pelo Usuário

O conceito de UDA, *User Defined Aggregates*, representa a base do algoritmo desenvolvido neste trabalho. Diferentemente das funções normais do SQL (sejam definidas pelo próprio usuário ou não), que geram uma saída para cada linha retornada do resultado da consulta, as funções de agregação tem a capacidade de retornar uma única saída, operando sobre diversas linhas agrupadas do resultado. Existem diversas funções de agregação que são padrões na maioria dos SGBDs, como SUM, COUNT, AVG, MIN, MAX, entre outras.

De certa forma, uma operação de agregação pode ser considerada como um laço, iterando sobre diversas linhas de uma tabela. Antes da concepção deste recurso, implementar algoritmos ou funções mais complexas em SQL era uma tarefa difícil, em muitos casos impossível.

As funções de agregação definidas pelo usuário possuem uma estrutura comum, sendo compostas de uma etapa de *Inicialização*, a etapa de *Iteração* e a etapa de *Finalização*. Em grande parte dos sistemas, as etapas de Inicialização e de Finalização são opcionais, fazendo com que o usuário precise especificar somente as operações da etapa de Iteração.

O funcionamento básico de uma agregação, após ser executada a etapa de Inicialização, onde valores de variáveis são inicializados e as estruturas necessárias são criadas, consiste em operar sobre determinados campos a cada linha do resultado,

armazenando a saída temporária em memória até que a última linha do grupo seja processada. Neste momento, o resultado final é retornado, podendo sofrer adaptações e conversões definidas na etapa de Finalização. É importante salientar que a agregação pode ser executada diversas vezes em uma mesma consulta, caso esta possua agrupamentos gerados pela cláusula GROUP BY do SQL.

Neste trabalho, as agregações são representadas em pseudocódigo, baseado no formato descrito no trabalho de Wang, Zaniolo e Luo (WANG, ZANIOLO, LUO, 2003), sendo estas divididas em três blocos, denominados INITIALIZE, ITERATE e TERMINATE, representando as etapas de Inicialização, Iteração e Finalização, respectivamente. Dentro destes blocos, podem ser executados quaisquer comandos de DDL ou DML do SQL.

A tabela 2.5 ilustra um exemplo do formato utilizado para descrever agregações. A função descrita é a conhecida *avg* (média aritmética), que utiliza uma tabela temporária *state* para armazenar os resultados intermediários. A cada entrada, a soma total e o contador de elementos são atualizados. Ao final, o resultado da divisão do total pelo número de elementos é retornado.

Tabela 2.5: Exemplo da agregação *avg* descrita em pseudocódigo

```
AGGREGATE myavg(next integer) : real {
    INITIALIZE: {
        TABLE state {
            tsum integer,
            cnt integer
        }
        INSERT INTO state VALUES (next, 1);
    }

    ITERATE: {
        UPDATE state SET tsum = tsum + next, cnt = cnt + 1;
    }

    TERMINATE: {
        INSERT INTO RETURN SELECT tsum / cnt FROM state;
    }
}
```

2.3.2 SQL e Mineração de Dados

Aplicações de mineração de dados são pouco suportadas pelos atuais sistemas de banco de dados, tendo como principais razões a limitação e o fraco poder de extensão da linguagem SQL. Mesmo que ao longo dos anos diversos recursos interessantes tenham sido adicionados à linguagem, os sistemas atuais não estão preparados para atender aplicações de suporte à tomada de decisão. Os *User Defined Aggregates*, apresentados na seção anterior, podem ser utilizados para contornar este problema.

Em (WANG, ZANIOLO, 2003), foi definida uma extensão nativa ao SQL chamada ATLaS, com suporte a UDAs e outros recursos, que possibilita a implementação de operações de mineração de dados. Os autores apresentaram a

implementação de um algoritmo de mineração de padrões freqüentes, tendo como base o algoritmo Apriori (AGRAWAL, SRIKANT, 1994) (Algoritmo Apriori na Wikipédia). Algumas outras aplicações também são sugeridas no artigo, demonstrando a variedade de aplicações para a extensão.

As idéias apresentadas em (WANG, ZANIOLO, 2003) demonstram que é possível implementar operações de mineração de dados em SQL, considerando que o suporte à UDAs existe em sistemas atuais. O principal problema do ATLaS consiste no fato de que o usuário precisa deste sistema para fazer mineração em bancos de dados convencionais. O objetivo deste trabalho não é criar mais um sistema, e sim utilizar o poder do SQL para enriquecer ainda mais esta linguagem que já é suportada por vários SGBDs.

3. Mineração de Padrões Seqüenciais em Trajetórias Semânticas

O objetivo deste trabalho é definir uma linguagem capaz de implementar as diferentes etapas do processo de descoberta de conhecimento sobre trajetórias, abrangendo operações de pré-processamento (notadamente a definição de granularidades espaciais e temporais), algoritmos para a extração de padrões seqüenciais, e o armazenamento dos padrões gerados para consultas posteriores, possibilitando a etapa de pós-processamento.

A figura 3.1 ilustra o processo completo implementado neste trabalho. O banco de dados contendo as trajetórias brutas (TB) é pré-processado juntamente com o banco de dados geográfico (BDG), resultando em trajetórias semânticas (TS). Esta etapa é realizada a partir dos métodos apresentados em (CHIECHELSKI, BOGORNY, 2008). As trajetórias semânticas passam por uma etapa de transformação, onde são aplicadas as granularidades fornecidas pelo usuário, para então serem submetidas à etapa de mineração propriamente dita. Os padrões gerados são armazenados no banco de dados para consultas posteriores. Na figura 3.1, todas as diferentes fontes de dados são representadas como existindo em bancos de dados separados, mas na prática, todos estes dados encontram-se em um mesmo banco, separados por tabelas.

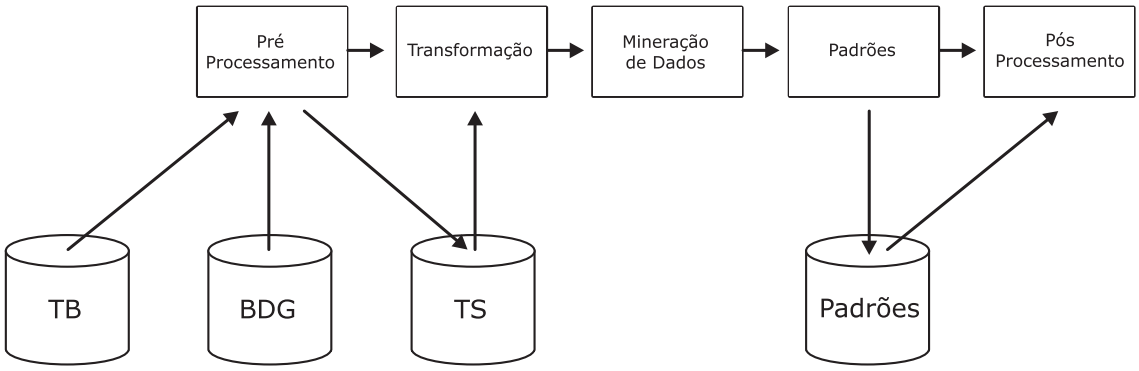

Figura 3.1: Processo de descoberta de conhecimento sobre dados espaço-temporais

Todo o processo é executado através de um algoritmo implementado unicamente em SQL. O usuário pode configurar a execução do algoritmo, informando diferentes valores para a granularidade do item a ser considerado na mineração. Além destes parâmetros, o usuário configura o suporte mínimo que os padrões gerados devem suportar.

Todos os padrões gerados por este processo são armazenados, como dados, no próprio banco de dados, permitindo a utilização dos mesmos em consultas futuras, refinando o processo de descoberta de conhecimento na etapa de pós-processamento. O algoritmo desenvolvido e apresentado no restante deste capítulo foi baseado nos trabalhos (WANG, ZANIOLO, 2003) e no trabalho (BOGORNY, KUIJPERS, ALVARES, 2008).

3.1 Visão geral

O fluxograma apresentado na figura 3.2 define, de forma geral, o algoritmo desenvolvido neste trabalho. O algoritmo pode ser dividido em três grandes blocos: *Pré-processamento*, *Mineração de Dados* e *Pós-processamento*. Cada um destes blocos é formado por uma seqüência de operações, que no SQL são implementadas na forma de diversas funções, agregações e tabelas. Na prática, todo o processo é executado a partir de uma função do SQL, denominada *sequentialStops*.

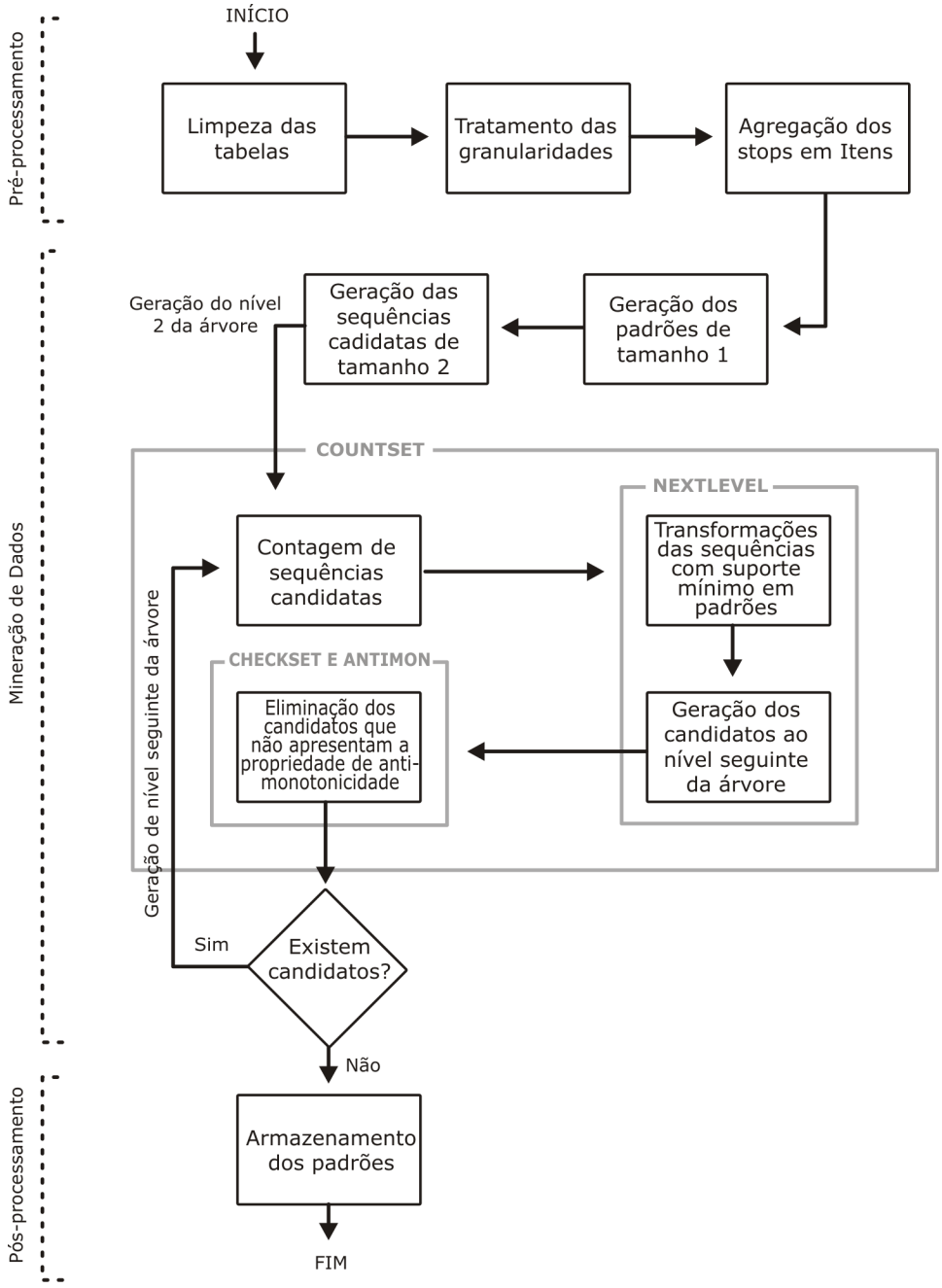

Figura 3.2: Visão geral do algoritmo

Na etapa de Pré-processamento, são realizadas operações de limpeza e

preparação dos dados. Como o algoritmo pode ser executado diversas vezes, as tabelas auxiliares envolvidas precisam ser esvaziadas no começo do processo. Além disso, os *stops* a serem minerados são agregados conforme as granularidades definidas pelo usuário, tornando-se itens.

As principais operações do algoritmo ocorrem na etapa de Mineração de Dados, e são implementadas por quatro agregações principais: *countset*, *nextlevel*, *checkset* e *antimon*. A lógica do algoritmo é fundamentalmente baseada em uma árvore de prefixos (WANG, ZANIOLO, 2003), ou *trie*, que é utilizada para estruturar todas as seqüências candidatas a padrões seqüenciais. A raiz desta árvore é um elemento nulo, sem pai, contendo um valor simbólico 0 (zero). Os demais nodos da árvore representam itens, dispostos em níveis. Realizando uma varredura em profundidade na árvore, cada caminho diferente da raiz até um nodo representa uma seqüência independente. A figura 3.3 contém uma representação da *trie*.

Cada nodo do nível *n* da *trie* define uma seqüência de tamanho igual a *n*. Os elementos que compõem esta seqüência são os nodos existentes no caminho da raiz até este nodo, inclusive. Como exemplo, as duas seqüências destacadas na árvore da figura 3.3 são {2, 3, 4} e {4, 5}.

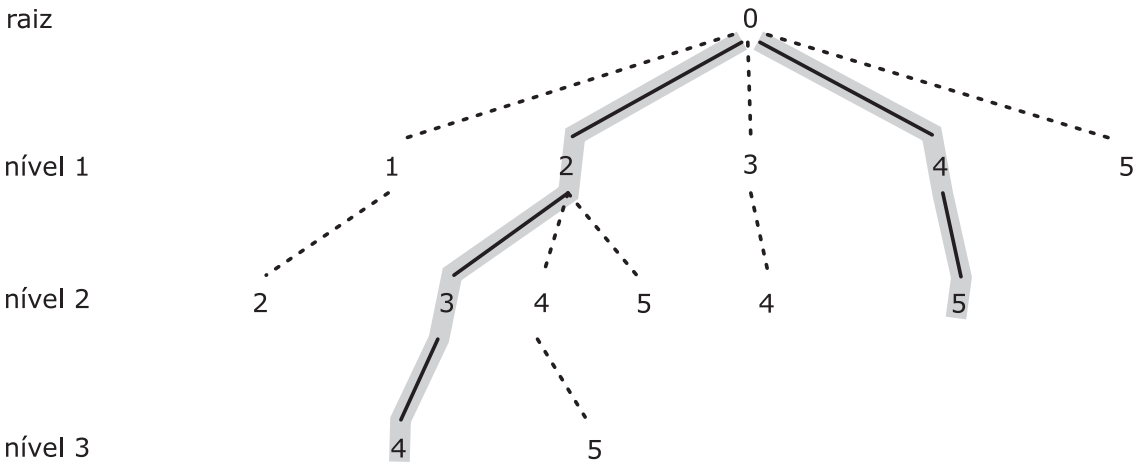

Figura 3.3: Representação da *trie* utilizada no algoritmo

A árvore é inicializada com o primeiro nível formado por todas as seqüências de tamanho 1 (um) que atingem o suporte mínimo (são padrões de tamanho 1), ou seja, todos os itens que aparecem em um número mínimo de trajetórias. Também são definidos os candidatos ao segundo nível da árvore, de forma que cada nodo do primeiro nível possui todos os outros nodos do primeiro nível como candidatos. A árvore então é expandida nível a nível pela agregação *countset*, que é executada recursivamente.

A agregação *countset* lê todas as trajetórias do banco de dados e conta as ocorrências de todas as possíveis seqüências de itens formadas por estas trajetórias. Cada nodo da árvore possui um contador indicando quantas trajetórias suportam a seqüência definida por ele. Após esta contagem, a agregação *nextlevel* transforma as seqüências que atingem o suporte mínimo em padrões, e descarta as demais, removendo-as da árvore. Os candidatos ao próximo nível da árvore também são gerados

por *nextlevel*.

Os novos candidatos são então submetidos ao teste da anti-monotonicidade, que se baseia na premissa de que uma seqüência candidata deve ter suporte mínimo para todas as suas subseqüências. Esta avaliação é realizada pelo conjunto das agregações *checkset* e *antimon.* A árvore é expandida através da agregação *countset* até que não existam mais novos candidatos.

A etapa de Pós-processamento, que é executada após a árvore estar completa, percorre todos os caminhos da *trie*, que neste ponto é composta somente por padrões seqüenciais. Estes padrões são armazenados no banco de dados, possibilitando que sejam utilizados para refinamentos futuros.

3.2 Organização dos dados

A principal fonte de dados necessária para a mineração dos padrões seqüenciais são as informações sobre as trajetórias, no caso deste trabalho, os *stops* que compõem as trajetórias. Estes *stops*, que geralmente são fornecidos pela aplicação ou pelo usuário, são armazenados em uma tabela do banco de dados, descrita na tabela 3.1.

O campo *stopId*, que é a chave primária da tabela stop, representa uma numeração seqüencial para cada *stop*, identificando-os de forma única. O *tid* indica a qual trajetória o *stop* pertence, sendo esta informação fundamental para a avaliação do suporte mínimo de um padrão. Os campos *startT*, *endT* e *duration* referem-se ao aspecto temporal do *stop*, indicando os horários de início e fim de sua ocorrência, bem como a sua duração total. A descrição espacial do *stop* é feita pelos campos *stopName*, *stopGid* e *stopTheGeom*, este último contendo a informação geográfica sobre o *stop*. O campo *stopName* indica o tipo do *stop* (bairro, rua, cidade) enquanto o campo *stopGid* identifica instâncias de um mesmo tipo (Bairro Menino Deus, Rua Andradas, etc.).

Tabela 3.1: Tabela stop

```
TABLE stop {
    stopId serial,
    tid integer,
    startT timestamp,
    endT timestamp,
    duration interval,
    stopName string,
    stopGid integer,
    stopTheGeom geometry
}
```

Os *stops* não são utilizados diretamente na mineração, pois existe a etapa de pré-processamento destes dados, transformando-os em itens a serem minerados, com base nos parâmetros informados pelo usuário. As funções de pré-processamento são detalhadas na seção 3.3. Após este tratamento, os *stops* agregados são armazenados em uma tabela chamada *item*, mostrada na tabela 3.2, que conterá efetivamente os dados utilizados ao longo da mineração.

Tabela 3.2: Tabela item

```
TABLE item {
    stopId integer,
    tid integer,
    item string,
    itemId integer,
    itemName string,
    itemGid integer,
    itemStart string,
    itemEnd string
}
```

Basicamente, existe uma entrada na tabela *item* para cada entrada na tabela *stop*. Portanto, os campos *stopId* e *tid* são os mesmos nas duas tabelas. Os campos *item* e *itemId* são elementos importantíssimos no algoritmo, pois identificam o que será minerado. O campo *itemId* é uma representação numérica do valor do campo *item*, para fins de desempenho do algoritmo. Os campos *itemName*, *itemGid*, *itemStart* e *itemEnd* representam os aspectos espaciais e temporais individuais de cada item. Estas quatro colunas juntas formam o valor armazenado na coluna *item*. A tabela *item* possui um índice sobre a coluna *itemId* e outro sobre a coluna *tid*, pois estas colunas são pesquisadas com muita freqüência.

As demais tabelas envolvidas no algoritmo desempenham papéis secundários, sendo utilizadas como tabelas auxiliares ao longo do processamento. A tabela 3.3 apresenta em detalhes a estrutura destas tabelas.

A tabela *trie* é responsável por armazenar a árvore utilizada no algoritmo. Cada registro da tabela representa um nodo da árvore. A tabela *cand*, juntamente com a tabela *previous*, tem como objetivo o armazenamento e contagem de freqüência dos candidatos ao nível seguinte da árvore. Os candidatos que não apresentam uma contagem satisfatória, ou seja, que não atingem o suporte mínimo desejado pelo usuário, são descartados.

Ao final do processamento de um nível da árvore e da exclusão dos candidatos que não atingem o suporte desejado, são gerados os candidatos ao nível seguinte da árvore, armazenados na tabela *nextcand*. Estes novos candidatos são submetidos à checagem de anti-monotonicidade. A tabela *subitem* é utilizada para este propósito.

Na etapa de pré-processamento, as granularidades temporais fornecidas pelo usuário são armazenadas na tabela *timeg*, para serem utilizadas na agregação de *stops* em itens.

Tabela 3.3: Tabelas auxiliares

TABLE trie {
id serial,
father integer,
itemId integer
}
	TABLE cand {
	id serial,
	father integer,
	count integer,
	itemId integer
	}
		TABLE nextcand {
		id serial,
		father integer,
		count integer,
		itemId integer
		}

TABLE previous {
id serial,
node integer,
level integer,
tid integer
}
	TABLE subitem {
	id serial,
	node integer,
	level integer
	}
		TABLE timeg {
		id serial,
		startT time,
		endT time,
		label string
		}

Tabela 3.4: Tabelas utilizadas no armazenamento dos padrões gerados

TABLE pattern {
id serial,
size integer,
pid integer
}
	TABLE pattern_item {
	id serial,
	itemId integer,
	itemName string,
	itemGid integer,
	itemStart string,
	itemEnd string,
	pid integer
	}

Finalmente, existem duas tabelas utilizadas somente na etapa de armazenamento dos padrões gerados, designadas para armazenar cada um dos padrões e seus respectivos itens. As tabelas são apresentadas na tabela 3.4 e o processo de armazenamento dos padrões é explicado na seção 3.5.

Existem índices nestas tabelas sobre as colunas mais pesquisadas, como as colunas *itemId* e *father* das tabelas *trie*, *cand* e *nextcand*, a coluna *node* das tabelas *previous* e *subitem* e a coluna *pid* da tabela *pattern_item*.

3.3 Pré-processamento

3.3.1 Tratamento das granularidades temporais

A primeira operação efetiva de pré-processamento executada pelo algoritmo é o tratamento das granularidades temporais fornecidas pelo usuário. Como pode ser visto na tabela 3.6, as granularidades são fornecidas em uma string, separadas por um *pipe* ("|"). As três partes obrigatórias de uma granularidade temporal são separadas por vírgulas (","). Cada granularidade é tratada e armazenada na tabela *timeg*, para ser utilizada nas etapas seguintes do algoritmo.

A string é tratada por um conjunto de funções, descritas na tabela 3.5. A tabela 3.6 contém um exemplo de utilização destas funções, que precisam ser chamadas a cada

execução do algoritmo de mineração.

Nas funções descritas na tabela 3.5, a função *split* executa uma repartição de uma string a partir de um separador informado pelo usuário, retornando um array de strings. A função *unpack* tem como objetivo converter um array de dados em um conjunto, ou seja, uma estrutura que pode ser considerada como uma tabela do banco de dados. Estas funções não são detalhadas, pois são implementadas de forma diferente em cada SGBD.

Tabela 3.5: Funções responsáveis pelo tratamento das granularidades temporais

```
FUNCTION packTimeG(array string[]) {
    SELECT ROW(array[1]::time, array[2]::time, array[3]);
}

FUNCTION splitTimeG(gran string) {
    SELECT packTimeG(split(g, ',')) FROM unpack(split(gran, '|')) AS
"g";
}

FUNCTION saveTimeG(timeG row) {
    INSERT INTO timeg (startT, endT, label) VALUES(timeG[1], timeG[2],
timeG[3]);
}
```

A função *splitTimeG* executa a separação das granularidades, que são repassadas para a função *packTimeG*, que tem como objetivo retornar uma estrutura representando uma linha da tabela contendo os dados corretos para o armazenamento em *timeg*. As strings que representam os horários precisam ser transformadas para um tipo *time*, dependendo do ambiente do banco de dados. Todas estas linhas geradas são passadas para a função *saveTimeG*, que tem como único trabalho realizar a inserção dos dados no banco.

Tabela 3.6: Exemplo de utilização das funções para tratamento das granularidades temporais

```
SELECT saveTimeG(t) FROM
splitTimeG('15:30,16:00,intervalo|18:30,22:00,happy_hour') AS "t";
```

3.3.2 Agregação de stops em itens

Antes da mineração propriamente dita, os *stops* fornecidos pelo usuário sofrem um processo de agregação, tornando-se itens. Isso é necessário porque *stops* são objetos complexos que envolvem as dimensões de tempo e espaço, essenciais para o processo de mineração. Três fatores definem quais itens serão formados a partir dos *stops*: o tipo do item, a granularidade espacial e a granularidade temporal.

A granularidade espacial pode assumir dois valores distintos: TYPE e INSTANCE. O tipo TYPE indica que os itens serão gerados considerando-se somente

os tipos dos *stops*. Dois *stops* definidos em bairros diferentes, por exemplo, constituiriam o mesmo nome de item: bairro. O segundo valor que a granularidade espacial pode assumir é INSTANCE, que ao contrário de TYPE, diferencia os *stops* por seus tipos e instâncias individualmente. No caso do exemplo anterior, os dois bairros constituiriam itens diferentes, bairro_A e bairro_B.

As granularidades temporais são opcionais, mas podem ser definidas em qualquer dimensão. Uma granularidade temporal consiste de um horário de início, um horário de término e de um rótulo identificador. Por exemplo, caso o usuário queira especificar granularidades para os três turnos do dia, poderia fornecer as seguintes estruturas: [08:00, 12:00, morning], [12:01, 18:00, afternoon], [18:01, 23:59, night]. Os itens que se enquadrarem dentro de alguma destas faixas, serão rotulados com os identificadores correspondentes.

O item é o atributo que define o que vai ser considerado na mineração, e conseqüentemente, é o elemento que irá compor os padrões gerados. No contexto de mineração de dados espaço-temporais, especificamente de trajetórias, um item representa um conjunto de informações, ao contrário da mineração de dados convencional, onde um item é um atributo simples, como produtos de uma loja.

O item de uma trajetória é construído de acordo com um dos tipos listados abaixo, podendo ser definido com base em diferentes granularidades:

- **Name**: é o nome do *stop*. Caso o usuário esteja utilizando uma granularidade espacial em tipos, ou seja, uma granularidade que considera os *spatial feature types*, possíveis itens seriam Hotel, ou Aeroporto. Caso o usuário utilize a granularidade espacial por instâncias, ou *spatial feature instances*, os itens Hotel_Mercury e Aeroporto_Salgado_Filho poderiam ser gerados.

- **NameStart**: é o nome do *stop* juntamente com o seu horário de início. Os horários de início podem ser classificados de acordo com as granularidades temporais fornecidas pelo usuário. Supondo que o usuário definiu duas granularidades temporais, [08:00, 12:00, morning] e [12:01, 18:00, afternoon], possíveis itens seriam Hotel_morning, Hotel_Mercury_afternoon, ou Aeroporto_Salgado_Filho_afternoon. Caso o horário de início de um *stop* não se enquadre em nenhuma granularidade temporal, o nome *other* é adicionado ao item, como em Supermercado_other.

- **NameEnd**: é similar ao tipo de item *NameStart*, com a diferença que o horário considerado é o de término do *stop*.

- **NameStartEnd**: este tipo de item é formado pelo nome do *stop*, o seu horário de início e também seu horário de término. Universidade_morning_afternoon e Museu_Louvre_afternoon_other são exemplos deste tipo de item.

Tabela 3.7: Exemplo de utilização das funções de agregação de *stops* em itens

```
/* pre-process stops (aggregate by granularities) */
INSERT INTO item (stopId, tid, item, itemName, itemGid, itemStart,
itemEnd) SELECT stop.stopId, stop.tid, itemAgg(itemSpace('INSTANCE',
stop.stopGid, stop.stopName), itemStart('NameStartEnd', stop.startT),
itemEnd('NameStartEnd', stop.endT)) AS "item", stop.stopName,
stop.stopGid, itemStart('NameStartEnd', stop.startT),
itemEnd('NameStartEnd', stop.endT) FROM stop;

/* give numbers to each item */
SELECT itemNumber(item.item) FROM (SELECT DISTINCT item.item FROM
item);
```

Os parâmetros acima são necessários para a transformação de *stops* em itens, sendo estes definidos pelo usuário. Na implementação, os aspectos espaciais e temporais são tratados separadamente, por funções próprias. A idéia básica é criar uma string que representará o item em si, sendo que a string pode conter até três partes, dependendo da configuração do item fornecido pelo usuário: nome, horário de início e horário de término. A tabela 3.7 apresenta um exemplo em SQL desta transformação.

As funções itemSpace, itemStart e itemEnd geram as três possíveis configurações da string do item. Caso alguma delas não seja necessária, como a parte referente ao horário de início não é necessária quando os tipos de itens utilizados são *Name* ou *NameEnd*, por exemplo, uma string vazia é retornada. Desta forma, as três funções são sempre executadas, independente do tipo de item desejado. A função itemAgg tem o objetivo de concatenar corretamente estas três possíveis configurações, separando cada uma delas com o caractere "_". As funções são detalhadas na tabela 3.8, onde o operador "||" representa uma concatenação de strings.

Tabela 3.8: Funções para a agregação de *stops* em itens

```
FUNCTION itemSpace(stopG string, stopGid integer, stopName string) {
    SELECT CASE WHEN stopG = 'TYPE' THEN stopName WHEN stopG =
'INSTANCE' THEN stopName || '_' || stopGid END;
}

FUNCTION itemStart(item string, startT timestamp) {
    SELECT CASE WHEN (item = 'NameStart' OR item = 'NameStartEnd')
THEN (SELECT tr.label FROM timeg AS tr WHERE startT >= tr.startT ORDER
BY tr.startT DESC LIMIT 1) END;
}

FUNCTION itemEnd(item string, endT timestamp) {
    SELECT CASE WHEN (item = 'NameEnd' OR item = 'NameStartEnd') THEN
(SELECT tr.label FROM timeg AS tr WHERE endT <= tr.endT ORDER BY
tr.endT ASC LIMIT 1) END;
}

FUNCTION itemAgg(iSpace string, iStart string, iEnd string) {
    SELECT iSpace || CASE WHEN iStart ISNULL THEN '' ELSE '_' ||
iStart END || CASE WHEN iEnd ISNULL THEN '' ELSE '_' || iEnd END;
}
```

Após a geração dos itens, uma segunda etapa é realizada, associando um número inteiro, iniciando em 1 (um), a cada item diferente. Itens iguais terão os mesmos números. Este procedimento, descrito na tabela 3.9, tem como objetivo aumentar o desempenho das comparações entre itens, visto que trabalhar com inteiros é mais eficiente do que trabalhar com strings, principalmente tratando-se de comparações de valores.

Tabela 3.9: Associação de números aos itens gerados

```
AGGREGATE itemNumber(item string) {
    INITIALIZE: {
        integer i = 1;
    }
    ITERATE: {
        UPDATE item SET item.itemId = i WHERE item.item = item;
        i++;
    }
}
```

3.3.3 Limpeza das tabelas

Tabela 3.10: Limpeza das tabelas auxiliares

```
TRUNCATE TABLE item;
TRUNCATE TABLE trie;
TRUNCATE TABLE cand;
TRUNCATE TABLE previous;
TRUNCATE TABLE nextcand;
TRUNCATE TABLE subitem;
TRUNCATE TABLE pattern;
TRUNCATE TABLE pattern_item;
TRUNCATE TABLE timeg;
```

Outro procedimento fundamental para a correta execução do algoritmo é a limpeza das tabelas auxiliares. Caso algum dado permaneça nestas tabelas entre uma execução e outra da mineração, a saída gerada não será correta. Esta limpeza deve ser executada antes de qualquer operação de pré-processamento, e sua implementação é detalhada na tabela 3.10.

3.4 O algoritmo

A mineração dos itens é realizada a partir de uma função principal, denominada *sequentialStops*, invocada pelo usuário através de uma consulta SQL. Esta função recebe como parâmetros os valores para as granularidades, bem como o suporte mínimo. A tabela 3.11 representa a BNF desta função, enquanto a tabela 3.12 exibe o formato geral de utilização da função, juntamente com um exemplo. Neste exemplo, os itens são agregados pelo nome de cada instância dos *stops*, utilizando duas granularidades temporais, *morning* e *afternoon*, além do suporte mínimo de 40%. A tabela 3.13 apresenta a implementação da função *sequentialStops*.

Ao longo da explicação, serão inseridas referências às etapas de cada função envolvida no algoritmo. A referência será formada pelo nome da tabela e por um valor numérico, identificando o trecho em questão.

Tabela 3.11: BNF da função sequentialStops

```
  <mining_function> ::= <pattern_type> <left_par> <mining_parameters>
                        <right_par>
     <pattern_type> ::= SEQUENTIALSTOPS
<mining_parameters> ::= <item> <coma> <stopG> <coma> <timeG> <coma>
                        <minSup>
             <item> ::= NAME | NAMESTART | NAMEEND | NAMESTARTEND
            <stopG> ::= TYPE | INSTANCE
            <timeG> ::= <startTime> <coma> <endTime> <coma> <label>
                        [{<pipe> <startTime> <coma> <endTime> <coma>
                        <label>]}...]
        <startTime> ::= <Time>
          <endTime> ::= <Time>
             <Time> ::= <hour>:<minute>
            <label> ::= string
           <minSup> ::= real
             <hour> ::= 00 | 02 | 03 | ... | 23
           <minute> ::= 01 | 02 | 03 | ... | 59 | 60
         <left_par> ::= (
        <right_par> ::= )
             <coma> ::= ,
             <pipe> ::= |
```

Como discutido anteriormente, antes da mineração propriamente dita, são executadas as operações de pré-processamento, que consistem na limpeza das tabelas auxiliares (3.13 I), no tratamento das granularidades (3.13 II) e na agregação dos *stops* em itens (3.13 III e 3.13 IV).

Tabela 3.12: Formato geral da função sequentialStops e exemplo de utilização no SQL

Formato geral

```
SELECT sequentialStops(item string, stopG string, timeG string,
support float);
```

Exemplo de utilização

```
SELECT sequentialStops('Name', 'INSTANCE',
'08:00,12:00,morning|12:01,18:00,afternoon', 0.4);
```

Assim que os itens são definidos, a etapa de mineração inicia. Os padrões de tamanho 1 (um) são gerados e inseridos na tabela *trie* (3.13 V). A seguir, é feito um *join* dos padrões de tamanho 1, (um) gerando seqüências candidatas de tamanho 2 (dois), inseridas na tabela *cand* (3.13 VI). É garantido que cada padrão e cada candidato apareça uma única vez na árvore, não contendo caminhos diferentes que formem a mesma seqüência de itens.

Em diversas etapas do algoritmo, a função *hasSupport* é utilizada. Esta função,

cuja implementação é mostrada na tabela 3.14, tem como objetivo testar se um dado número de trajetórias é suficiente para atingir o suporte mínimo, definido como um número real entre 0 e 1. Para tal, o número de trajetórias diferentes existentes na base de dados sendo minerada é utilizado como divisor da operação.

Tabela 3.13: Função sequentialStops

```
FUNCTION sequentialStops(item string, stopG string, timeG string,
support float) {
       /* clean tables and reset sequences */
(I)    SELECT cleanTables();

       /* save time granularities in a table */
(II)   SELECT saveTimeG(t) FROM splitTimeG(timeG) AS "t";

       /* pre-process stops (aggregate by granularities) */
(III)  INSERT INTO item (stopId, tid, item, itemName, itemGid,
itemStart, itemEnd) SELECT stop.stopId, stop.tid,
itemAgg(itemSpace('INSTANCE', stop.stopGid, stop.stopName),
itemStart('NameStartEnd', stop.startT), itemEnd('NameStartEnd',
stop.endT)) AS "item",  stop.stopName, stop.stopGid,
itemStart('NameStartEnd', stop.startT), itemEnd('NameStartEnd',
stop.endT) FROM stop;

       /* give numbers to each item */
(IV)   SELECT itemNumber(item.item) FROM (SELECT DISTINCT item.item
FROM item);

       /* generate frequent 1-itemsets */
(V)    INSERT INTO trie (itemId, father) SELECT item.itemId, 0 FROM
item GROUP BY item.itemId HAVING hasSupport(COUNT(DISTINCT item.tid),
support);

       /* self-join frequent 1-itemsets to candidates 2-itemsets */
(VI)   INSERT INTO cand (itemId, father, count) SELECT t1.itemId,
t2.id, 0 FROM trie AS t1, trie AS t2 WHERE t1.itemId != t2.itemId;

       /* generate (k+1)-itemsets from k-itemsets (k=2, ...) */
(VII)  SELECT countset(item.itemId, item.tid, 2, support) FROM (SELECT
item.itemId, item.tid FROM item ORDER BY item.tid, item.stopId);

       /* calculate the patterns */
(VIII) SELECT pattern(trie.itemId, trie.father) FROM (SELECT
trie.itemId, trie.father FROM trie WHERE trie.id NOT IN (SELECT
trie.father FROM trie GROUP BY trie.father));
}
```

Tabela 3.14: Função hasSupport

```
FUNCTION hasSupport(total integer, support float) {
    SELECT (total / (SELECT COUNT(DISTINCT item.tid) FROM item)) >=
support;
}
```

Tabela 3.15: Agregação countset

```
AGGREGATE countset(itemId integer, tid integer, level integer, support
float) {
    INITIALIZE: ITERATE: {
        /* initialize previous for a new trajectory */
(I)     DELETE FROM previous WHERE tid NOT IN (SELECT previous.tid
FROM previous GROUP BY previous.tid);
(II)    INSERT INTO previous (node, level, tid) SELECT 0, 0, tid WHERE
tid NOT IN (SELECT previous.tid FROM previous GROUP BY previous.tid);

        /* store supported frequent itemsets in previous */
(III)   INSERT INTO previous (node, level, tid) SELECT trie.id,
p.level + 1, tid FROM previous AS p, trie WHERE trie.itemId = itemId
AND trie.father = p.node AND p.level < level - 1;

        /* count candidates that appear in the transaction */
(IV)    UPDATE cand SET cand.count = cand.count + 1 WHERE cand.itemId
= itemId AND cand.id = ANY(SELECT c.id FROM previous AS p, cand AS c
WHERE p.level = level - 1 AND c.father = p.node);
    }
    TERMINATE: {
        /* derive trie on level j and candidates on level j+1 */
(V)     SELECT nextlevel(cand.itemId, cand.father, (SELECT COUNT(*)
FROM cand AS c WHERE hasSupport(c.count, support) AND c.father =
cand.father), support) FROM cand WHERE hasSupport(cand.count, support)
GROUP BY cand.father;

        /* eliminates candidates by the anti-monotonicity */
(VI)    INSERT INTO subitem (node, level) VALUES (0, 0);
(VII)   SELECT checkset(nextcand.itemId, nextcand.father),
antimon(nextcand.itemId, nextcand.father, level) FROM (SELECT
nextcand.itemId, nextcand.father FROM nextcand ORDER BY
nextcand.father ASC);

        /* copy table nextcand to cand */
(VIII)  DELETE FROM cand;
(IX)    INSERT INTO cand (itemId, father, count) SELECT
nextcand.itemId, nextcand.father, nextcand.count FROM nextcand;

        /* ascend to level j+1 if cands not empty */
(X)     SELECT ascendToNextLevel(level, support) FROM (SELECT COUNT(*)
AS size FROM nextcand) WHERE size > 0;
    }
}
```

Tendo o primeiro nível da árvore e os candidatos ao segundo nível definidos, a agregação *countset* é executada (3.13 VII), tendo como objetivo inicial construir o segundo nível da árvore a partir dos candidatos existentes. Esta agregação constrói um nível da árvore a cada execução, sendo chamada recursivamente para expandir a árvore até que todos os padrões tenham sido gerados. Por último, os padrões são armazenados (3.13 VIII).

A agregação *countset*, ilustrada na tabela 3.15, recebe um fluxo contínuo de itens a cada execução, ordenados por suas respectivas trajetórias. Como as trajetórias são

processadas separadamente, a tabela *previous* é inicializada contendo somente o nodo raiz a cada nova trajetória (3.15 I e 3.15 II). Todas as seqüências que são suportadas pela trajetória têm suas referências armazenadas na tabela *previous* (3.15 III), para que no comando seguinte, tenham seus contadores de freqüência incrementados (3.15 IV). Desta forma, após cada trajetória ter sido processada, todas as seqüências derivadas de seus itens que possam ser encontradas na árvore (entre os itens já existentes na árvore e os candidatos ao próximo nível) são contabilizadas.

Após todas as trajetórias terem sido processadas por *countset* e suas respectivas seqüências contadas, a agregação *nextlevel* é invocada (3.15 V). *Nextlevel* é responsável por adicionar na árvore (3.16 I), mais especificamente no nível sendo construído, todos os candidatos que atingiram o suporte mínimo necessário, definidos por seus contadores de freqüência, incrementados na etapa anterior. Os candidatos não selecionados são descartados. A tabela 3.16 apresenta a implementação desta agregação.

Tabela 3.16: Agregação nextlevel

```
AGGREGATE nextlevel(itemId integer, father integer, count integer,
support float) {
    INITIALIZE: ITERATE: {
        /* insert candidate with minSup in the trie */
(I)     INSERT INTO trie (itemId, father) VALUES (itemId, father);

        /* generate candidates on level j+1 */
(II)    INSERT INTO nextcand (itemId, father, count) SELECT
item.itemId, (SELECT trie.id FROM trie WHERE trie.itemId = itemId AND
trie.father = father), 0 FROM item WHERE item.itemId != itemId AND
count > 1 GROUP BY item.itemId HAVING hasSupport(COUNT(*), support);
    }
}
```

Após a definição do novo nível da árvore, a agregação *nextlevel* verifica se os novos nodos possuem possíveis candidatos ao próximo nível da árvore. Um nodo recém inserido pode ter filhos na árvore caso o seu nodo pai tenha mais do que um filho, ou seja, caso o nodo recém inserido tenha irmãos do mesmo nodo pai. Estes novos candidatos são inseridos na tabela *nextcand* (3.16 II).

Mesmo que novos candidatos tenham sido obtidos (seqüências de tamanho *level + 1* candidatas a tornarem-se padrões), estas novas seqüências precisam passar pelo teste da anti-monotonicidade. Esta avaliação é realizada pelo conjunto das agregações *checkset* e *antimon* (3.15 VI e 3.15 VII), definidas nas tabelas 3.17 e 3.18, respectivamente.

Para a execução do teste de anti-monotonicidade, é utilizada a tabela *subitem*, responsável por armazenar as subseqüências de todas as seqüências candidatas de *nextcand*. A agregação *checkset* é executada recursivamente (3.17 I) para preencher a tabela *subitem* (3.17 II), partindo da raiz até as folhas da árvore.

Assim que a tabela *subitem* é preenchida com os dados de uma seqüência candidata, a agregação *antimon* elimina da tabela *nextcand* (3.18 I) todas as seqüências

de tamanho *n + 1* que não possuem no mínimo *n + 1* subseqüências de tamanho *n*. A tabela *subitem* é então reinicializada (3.18 II) para armazenar as subseqüências da candidata seguinte.

Tabela 3.17: Agregação checkset

```
AGGREGATE checkset(itemId integer, father integer) {
    INITIALIZE: ITERATE: {
        /* call checkset recursively */
(I)     SELECT checkset(trie.itemId, trie.father) FROM trie WHERE
father != 0 AND  trie.id = father;

        /* as we exit the recursion, we expand subitem */
(II)    INSERT INTO subitem (node, level) SELECT trie.id, s.level + 1
FROM subitem AS s, trie WHERE trie.itemId = itemId AND trie.father =
s.node;
    }
}
```

Tabela 3.18: Agregação antimon

```
AGGREGATE antimon(itemId integer, father integer, level integer) {
    INITIALIZE: ITERATE: {
        /* prune the itemset from nextcand if it don't apply for the
anti-monotonic property */
(I)     DELETE FROM nextcand WHERE nextcand.itemId = itemId AND
nextcand.father = father AND level + 1 > (SELECT COUNT(*) FROM subitem
WHERE subitem.level = level);

        /* clean subitem table for next itemset */
(II)    DELETE FROM subitem WHERE subitem.node != 0;
    }
}
```

Tabela 3.19: Função ascendToNextLevel

```
FUNCTION ascendToNextLevel(level integer, support float) {
      /* clean aux tables */
(I)   DELETE FROM nextcand;
(II)  DELETE FROM subitem;

      /* call the countset aggregate to build the j+1 level of the
trie */
(III) SELECT countset(item.itemId, item.tid, level + 1, support) FROM
(SELECT item.itemId, item.tid FROM item ORDER BY item.tid,
item.stopId);
}
```

Passado este ponto, as seqüências que respeitam a propriedade da anti-monotonicidade são copiadas da tabela *nextcand* para a tabela *cand* (3.15 VIII e 3.15 IX), tornando-se candidatas prontas para o próximo nível da árvore. Caso a tabela

nextcand não esteja vazia, ou seja, caso alguma seqüência candidata válida tenha sobrevivido ao processo, a função *ascendToNextLevel* é executada (3.15 X). Esta função, apresentada na tabela 3.19, é a responsável por limpar as tabelas auxiliares necessárias para o reinício do processo (3.19 I e 3.19 II), e executar a agregação *countset* novamente (3.19 III), desta vez para o nível seguinte da árvore.

O processo de armazenamento dos padrões, executado assim que a árvore é totalmente construída (3.13 VIII), é apresentado na seção 3.5. A seção 3.6 apresenta uma simulação da execução deste método para um conjunto limitado de *stops*.

3.5 Armazenamento dos padrões

Poucas linguagens de mineração de dados armazenam os padrões gerados para serem utilizados em refinamentos posteriores. Nestes métodos, os filtros utilizados são manuais. Os processos de descoberta de conhecimento não são considerados satisfatórios com apenas uma execução, mas sim são processos repetitivos, onde os resultados gerados nas etapas anteriores servem de entrada para etapas seguintes.

As tabelas *pattern* e *pattern_item*, apresentadas na tabela 3.4, são utilizadas para o armazenamento dos padrões gerados. A tabela *pattern* reúne referências a todos os padrões gerados, onde cada registro contém o identificador do padrão, ou *pid* (*pattern id*), bem como o seu tamanho, ou seja, o número de itens que o compõem.

A tabela *pattern_item* armazena todos os itens presentes nos padrões, agrupados pelo *pid*. A coluna *item* contém o item agregado, em formato de string. As colunas *itemName*, *itemGid*, *itemStart* e *itemEnd* armazenam a mesma informação da coluna *item*, sendo que cada uma dessas colunas armazena um pedaço do item somente. Em outras palavras, isto representa a desagregação do item, fazendo do padrão um dado que pode ser consultado ou minerado. Exemplos de consultas sobre padrões serão apresentados no capítulo 4.

Ao final do processo de mineração, a função *sequentialStops* executa o trecho em SQL descrito na tabela 3.20. Este comando repassa para a agregação *pattern*, descrita na tabela 3.21, todas as folhas da árvore.

Tabela 3.20: Comando SQL para armazenar os padrões gerados

```
SELECT pattern(trie.itemId, trie.father) FROM (SELECT trie.itemId,
trie.father FROM trie WHERE trie.id NOT IN (SELECT trie.father FROM
trie GROUP BY trie.father));
```

Tabela 3.21: Agregação pattern

```
AGGREGATE pattern(itemId integer, father integer) {
    INITIALIZE: {
        integer i = 1;
    }
    ITERATE: {
        /* traverse the trie to get each item of the sequence */
(I)     SELECT patternItem(itemId, father, i);

        /* register the pattern */
(II)    INSERT INTO pattern (size, pid) VALUES((SELECT COUNT(*) FROM
pattern_item WHERE pid = i), i);

        i++;
    }
}
```

A agregação *pattern* inicialmente repassa a folha para outra agregação (3.21 I), *patternItem*, apresentada na tabela 3.22. O *pid* deste novo padrão é definido na própria agregação *pattern*, iniciando em 1 (um). A agregação *patternItem* percorre a árvore recursivamente (3.22 I), da folha até a raiz, de forma que os itens são inseridos na tabela *pattern_item* conforme a recursão vai sendo finalizada (3.22 II).

Ao final da inserção dos itens de cada padrão, suas referências são armazenadas na tabela *pattern* (3.21 II), ficando disponíveis para consultas futuras.

Tabela 3.22: Agregação patternItem

```
AGGREGATE patternItem(itemId integer, father integer, pid integer) {
    INITIALIZE: ITERATE: {
        /* call patternItem recursively */
(I)     SELECT patternItem(trie.itemId, trie.father, pid) FROM trie
WHERE father != 0 AND trie.id = father;

        /* as we exit the recursion we expand the sequence */
(II)    INSERT INTO pattern_item (itemId, itemName, itemGid,
itemStart, itemEnd, pid) SELECT itemId, item.itemName, item.itemGid,
item.itemStart, item.itemEnd, pid FROM item WHERE item.itemId = itemId
LIMIT 1;
    }
}
```

3.6 Simulação

Para exemplificar o processo de mineração de padrões seqüenciais sobre trajetórias descrito neste trabalho, será realizada uma simulação passo a passo, considerando-se um conjunto pequeno de trajetórias e *stops*. O tipo de item considerado na simulação será *Name*, a granularidade espacial INSTANCE, e o suporte mínimo ajustado em 40%. Não serão utilizadas granularidades temporais nesta simulação. O comando SQL utilizado para executar esta simulação é exibido na tabela 3.23.

Tabela 3.23: Comando SQL para mineração dos padrões

```
SELECT sequentialStops('Name', 'INSTANCE', '', 0.4);
```

O primeiro passo no processo é transformar os *stops* em itens, considerando os parâmetros fornecidos pelo usuário. Os *stops* e os itens gerados são apresentados na tabela 3.24.

O primeiro nível da árvore é formado pelos itens que atingem o suporte mínimo de 40%, sendo estes armazenados na tabela *trie*. Como existem 5 (cinco) trajetórias diferentes sendo mineradas, o suporte mínimo de 40% indica que é preciso que pelo menos 2 (duas) trajetórias suportem a seqüência candidata. O primeiro nível da árvore irá conter seqüências de tamanho 1 (um) que atingiram o suporte mínimo necessário. Na figura 3.4, o 1º nível da árvore contém os identificadores dos diferentes bairros (1, 2, 3, 4 e 5).

Tabela 3.24: Trajetórias e s*tops* para simulação

tid	**stopName**	**stopGid**	**startT**	**endT**	**item**
1	bairro	2	08:00	08:30	bairro_2
1	bairro	3	09:00	09:30	bairro_3
1	bairro	4	10:00	10:30	bairro_4
2	bairro	1	11:00	11:30	bairro_1
2	bairro	2	12:00	12:30	bairro_2
2	bairro	3	13:00	13:30	bairro_3
2	bairro	4	14:00	14:30	bairro_4
3	bairro	3	15:00	15:30	bairro_3
3	bairro	4	16:00	16:30	bairro_4
3	bairro	5	17:00	17:30	bairro_5
4	bairro	1	18:00	18:30	bairro_1
4	bairro	2	19:00	19:30	bairro_2
4	bairro	5	20:00	20:30	bairro_5
5	bairro	2	21:00	21:30	bairro_2
5	bairro	4	22:00	22:30	bairro_4
5	bairro	5	23:00	23:30	bairro_5

A tabela *cand* é inicializada com os candidatos ao segundo nível da árvore, sendo que cada item da árvore possui como próximos candidatos todos os outros itens diferentes existentes no primeiro nível da árvore. Nesta etapa, os candidatos ainda não fazem parte da *trie*, somente são representados juntos na estrutura para melhor

visualização. A figura 3.4 exibe a estrutura gerada até este ponto, onde cada item do tipo bairro_1 é representado somente por seu *stopGid*, neste caso, 1.

Na representação gráfica, cada item candidato é desenhado logo abaixo de seu nodo pai, que já pertence à árvore. Desta forma, caso a árvore seja percorrida a partir da raiz, todas as seqüências candidatas de tamanho *j + 1* podem ser derivadas, sendo *j* o tamanho atual da árvore. Neste ponto, considerando-se somente o item bairro_1, as seqüências candidatas de tamanho 2 (dois) seriam {bairro_1, bairro_2}, {bairro_1, bairro_3}, {bairro_1, bairro_4} e {bairro_1, bairro_5}.

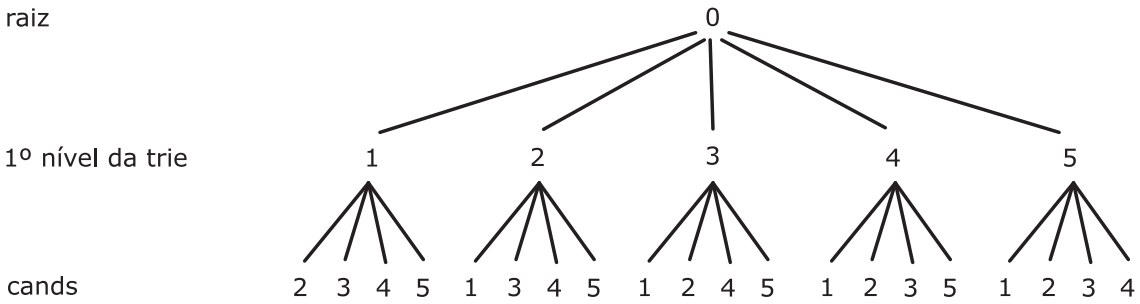

Figura 3.4: Primeiro nível da *trie* e candidatos ao segundo nível

A partir deste ponto, a árvore é expandida nível a nível pela agregação *countset*. Cada execução de *countset* expande a árvore em um nível, e recebe como parâmetros todos os itens das trajetórias, como em um fluxo ordenado. Para cada trajetória, todas as suas subseqüências são contabilizadas na tabela *cand*. Tomando como exemplo a trajetória 1 (um), que contém os itens {bairro_2, bairro_3, bairro_4}, as subseqüências {bairro_2, bairro_3}, {bairro_2, bairro_4} e {bairro_3, bairro_4} seriam incrementadas. Estes contadores são vinculados às folhas da estrutura, ou seja, aos candidatos.

Ao final desta etapa inicial, os candidatos existentes que atingiram suporte mínimo, ou seja, os candidatos que possuem um contador não inferior a 2 (dois), são incorporados à *trie*. Ao mesmo tempo, os candidatos do próximo nível são gerados (neste ponto, o terceiro nível da árvore). A agregação *nextlevel* é responsável por esta expansão. Para cada novo item *x* inserido na *trie*, somente são inseridos candidatos no nível seguinte da árvore para este nodo caso o nodo pai de *x* possua mais do que um nodo filho, ou seja, somente no caso em que *x* possua outros itens no mesmo nível da árvore, pertencendo ao mesmo nodo pai. A figura 3.5 mostra a estrutura resultante após a geração do segundo nível da árvore e da obtenção dos candidatos ao terceiro nível.

O item bairro_2, anexado à árvore no segundo nível como filho de bairro_1, indica que a seqüência {bairro_1, bairro_2} é suportada por pelo menos duas trajetórias. Mas o item bairro_2 não possui filhos candidatos ao próximo nível, pois ele é o único filho de bairro_1. Em outras palavras, não poderia existir uma seqüência {bairro_1, bairro_2, bairro_N} sem existir a subseqüência {bairro_1, bairro_N}.

Os candidatos ao nível seguinte da árvore, armazenados na tabela *nextcand*, sofrem um processo de seleção, considerando-se a regra da anti-monotonicidade. As agregações *checkset* e *antimon* realizam este trabalho. A regra de anti-monotonicidade

considera como seqüências válidas somente aquelas que possuem suporte mínimo em todas as suas subseqüências. Por exemplo, a seqüência {bairro_2, bairro_3, bairro_5} é candidata ao terceiro nível da árvore, mas é eliminada porque a subseqüência {bairro_3, bairro_5} não possui suporte mínimo. A figura 3.6 exibe a estrutura após a execução desta etapa.

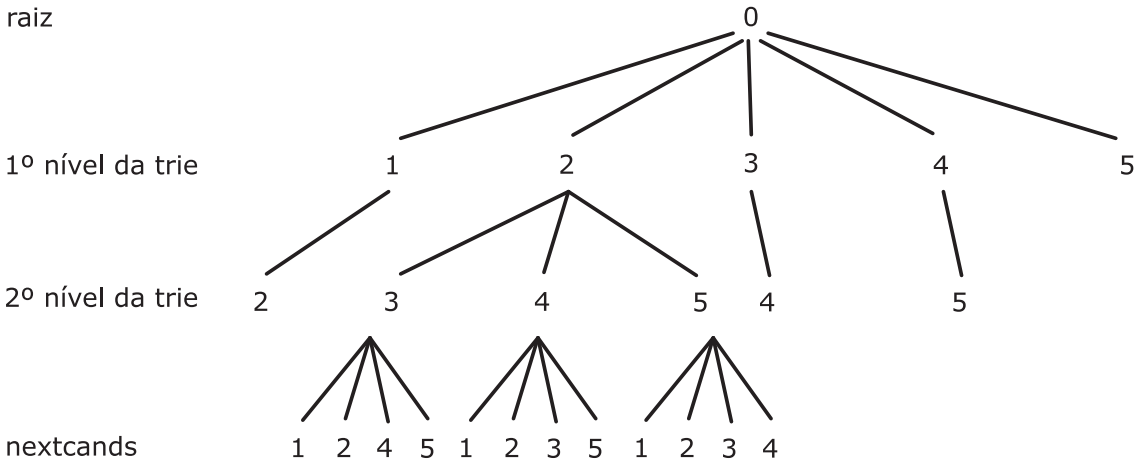

Figura 3.5: Segundo nível da *trie* e candidatos ao terceiro nível

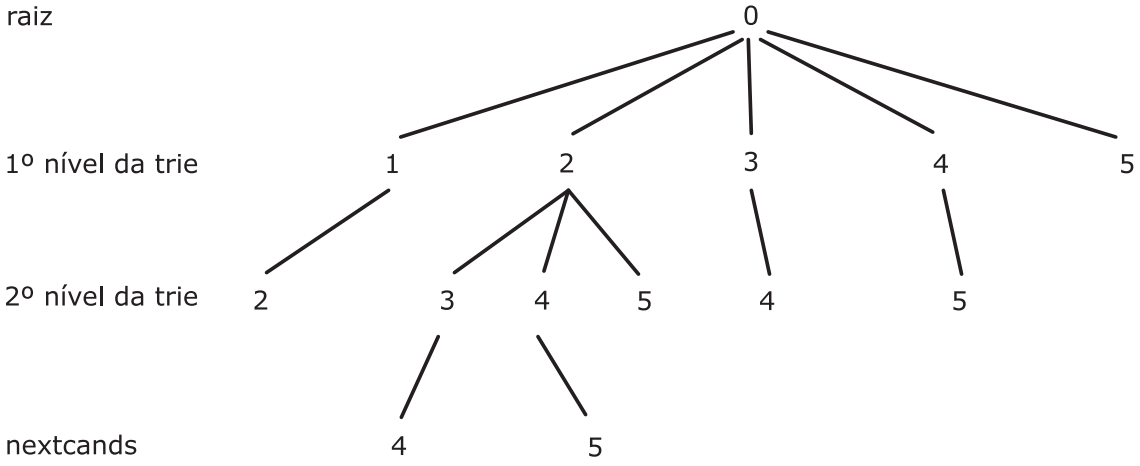

Figura 3.6: Novos candidatos que passaram pelo teste da anti-monotonicidade

Tendo sido gerado o segundo nível da árvore e seus respectivos candidatos, a agregação *countset* é executada novamente, desta vez para construir o terceiro nível da árvore. O mesmo processo de contagem de candidatos é realizado, sendo que somente são adicionados à *trie* as seqüências candidatas que atingiram o suporte mínimo (a seqüência {bairro_2, bairro_4, bairro_5} não atingiu o suporte mínimo, e por isso não foi adicionada ao terceiro nível da *trie*). A figura 3.7 exibe a estrutura da árvore até este ponto. Após a inserção dos itens no terceiro nível, não existem candidatos ao quarto nível da árvore (tabela *nextcand* vazia), e desta forma o processo é finalizado. A *trie* está completa, contendo todos os padrões seqüenciais gerados.

Na tabela 3.25, são apresentados alguns dos padrões gerados pelo processo,

ordenados pelo tamanho. Na verdade, somente são exibidos os padrões que terminam em folhas da árvore. Desta forma, o padrão {bairro_2, bairro_3} não aparece na saída do processo, pois o padrão mais abrangente {bairro_2, bairro_3, bairro_4} já possui esta informação implícita. De forma análoga, o padrão {bairro_5} não precisaria aparecer como resultado, pois o padrão {bairro_4, bairro_5} já contém esta informação, mas este padrão aparece na saída por estar terminando em uma folha da árvore.

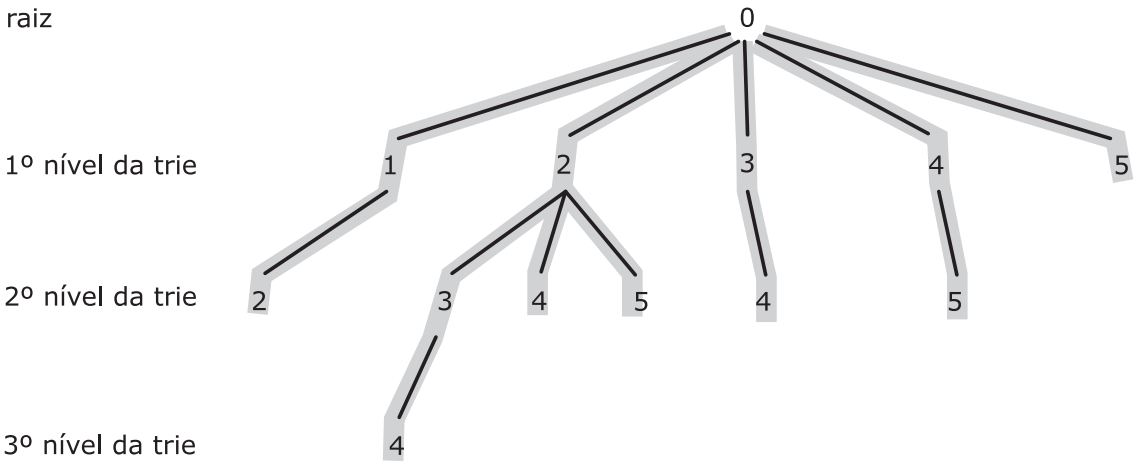

Figura 3.7: Terceiro nível da árvore, sem candidatos ao nível seguinte

Tabela 3.25: Padrões seqüenciais gerados

pid	**tamanho**	**itens**
1	1	{bairro_5}
2	2	{bairro_3, bairro_4}
3	2	{bairro_1, bairro_2}
4	2	{bairro_2, bairro_4}
5	2	{bairro_2, bairro_5}
6	2	{bairro_4, bairro_5}
7	3	{bairro_2, bairro_3, bairro_4}

Os padrões descritos na tabela 3.25 são armazenados nas tabelas *pattern* e *pattern_item*. A tabela *pattern* contém referências aos sete padrões encontrados pelo algoritmo, enquanto a tabela *pattern_item* armazena todos os itens de cada padrão. As imagens 3.8 e 3.9 mostram estas duas tabelas com seus respectivos dados.

Row	id (int4)	size (int4)	pid (int8)
1	1	1	1
2	2	2	2
3	3	2	3
4	4	2	4
5	5	2	5
6	6	2	6
7	7	3	7

Figura 3.8: Tabela pattern após a execução da simulação

Row	Pid (int8)	Size (int4)	Item (varchar)	itemName (varchar)	itemGid (int8)	itemStart (varchar)	itemEnd (varchar)
1	1	1	bairro_5	bairro	5		
2	2	2	bairro_3	bairro	3		
3	2	2	bairro_4	bairro	4		
4	3	2	bairro_1	bairro	1		
5	3	2	bairro_2	bairro	2		
6	4	2	bairro_2	bairro	2		
7	4	2	bairro_4	bairro	4		
8	5	2	bairro_2	bairro	2		
9	5	2	bairro_5	bairro	5		
10	6	2	bairro_4	bairro	4		
11	6	2	bairro_5	bairro	5		
12	7	3	bairro_2	bairro	2		
13	7	3	bairro_3	bairro	3		
14	7	3	bairro_4	bairro	4		

Figura 3.9: Tabela pattern_item após a execução da simulação

4. Experimentos

Os experimentos apresentados neste capítulo foram feitos com base em trajetórias obtidas de um carro sonda, equipado com um aparelho de GPS, que circulou e coletou dados durante três anos pela cidade do Rio de Janeiro. O objetivo deste carro sonda era detectar pontos de congestionamentos pela cidade. Através do método apresentado neste trabalho, é possível extrair relações mais complexas e úteis a respeito do comportamento do tráfego urbano de automóveis a partir dos dados coletados.

Os dados obtidos do carro sonda são trajetórias brutas, formadas por milhões de pontos no espaço. Além disso, é registrado em cada ponto, a data e a hora de sua coleta. Estes dados, como discutido anteriormente no trabalho, têm pouca capacidade de expressão de conhecimento, pois são difíceis de serem interpretados por humanos. Soma-se a isso o custo computacional elevado para processá-los. Para obter melhores resultados, os dados passaram por uma etapa de pré-processamento.

Foi utilizado o método CB-SMoT, apresentado e implementado em (CHIECHELSKI, BOGORNY, 2008), para o pré-processamento das trajetórias. Com o processo de agregação de semântica, elas passaram de trajetórias brutas a trajetórias semânticas.

4.1 Favelas do Rio de Janeiro

Para o primeiro experimento, foi utilizado um conjunto de 500 trajetórias, totalizando 1.888.076 pontos. Isto significa que a tabela com 500 trajetórias continha 1.888.076 registros. Estes dados foram transformados em trajetórias semânticas, com a representação de favelas do Rio de Janeiro. Desta forma, os *stops* das trajetórias que passam por favelas são identificados como tal, enquanto que os demais são marcados como *UNKNOWN*, desconhecidos para a aplicação.

A figura 4.1 apresenta um subconjunto de padrões seqüenciais interessantes obtidos após a mineração dos dados. Os itens que compõem os padrões são no formato *nome do objeto* (neste caso favela) mais o *identificador único da instância*.

Os itens foram agregados pelas instâncias de *stops*, ou seja, pelos nomes das favelas, sem granularidades temporais. O suporte mínimo necessário para os padrões é de 1%, ou 5 trajetórias. O comando utilizado para esta operação é mostrado na tabela 4.1.

Tabela 4.1: Consulta para minerar padrões em favelas do Rio com suporte mínimo de 1%

```
SELECT sequentialStops('Name', 'INSTANCE', '', 0.01);
```

O padrão 3 parte da favela Parque Jardim Beira-Mar (itemGid 321) para a favela

Parque São José de Barros Filho / Gleba I (itemGid 338). Como não foram utilizadas granularidades temporais nesta mineração de dados, estas paradas podem ter acontecido em qualquer horário do dia, até mesmo em dias diferentes. O que se pode saber é que pelo menos 1% das trajetórias passou de uma favela para outra.

Além destes padrões, existem outros dois que indicam a mesma movimentação de chegada na favela Nova Holanda (itemGid 293), mas partindo de um ponto desconhecido para a aplicação e passando por favelas diferentes. O padrão 31 indica que existiram trajetórias passando pela favela Parque União (itemGid 343), enquanto que o padrão 32 demonstra rotas através da favela Rubens Vaz (itemGid 395), ambas dirigindo-se para Nova Holanda. Com estas informações, é possível que existam áreas de tráfego lento nos acessos à favela Nova Holanda.

Row	Pid (int8)	Size (int4)	Item (varchar)
1	3	2	favela_321
2	3	2	favela_338
3	31	3	UNKNOWN_1
4	31	3	favela_343
5	31	3	favela_293
6	32	3	UNKNOWN_1
7	32	3	favela_395
8	32	3	favela_293

Figura 4.1: Padrões gerados considerando-se favelas do Rio de Janeiro e suporte mínimo de 1%

Para obter maiores detalhes sobre as movimentações urbanas entre favelas, pode-se utilizar granularidades temporais para saber em que período do dia ocorreram os *stops*. Esta mineração foi realizada com o comando exibido na tabela 4.2.

Tabela 4.2: Consulta para minerar padrões em favelas do Rio com suporte mínimo de 1%, considerando o período do dia em que cada *stop* da trajetória iniciou

```
SELECT sequentialStops('NameStart', 'INSTANCE',
'07:00,12:00,manha|12:00,18:00,tarde|18:00,23:00,noite', 0.01);
```

Foram utilizados três intervalos de tempo, definindo os três turnos do dia: manhã, tarde e noite. Os itens presentes nos padrões agora possuem o formato *nome do objeto* mais o *identificador único da instância* mais o *período do dia em que ocorreu*. Exemplos de padrões obtidos podem ser visualizados na figura 4.2.

Row	Pid (int8)	Size (int4)	Item (varchar)
1	18	2	favela_321_manha
2	18	2	UNKNOWN_1_manha
3	21	2	favela_293_tarde
4	21	2	UNKNOWN_1_noite
5	34	3	favela_293_tarde
6	34	3	favela_481_tarde
7	34	3	UNKNOWN_1_noite

Figura 4.2: Padrões gerados em favelas do Rio de Janeiro com suporte mínimo de 1%, considerando o período do dia em que cada *stop* da trajetória iniciou

Como no exemplo anterior, podem-se analisar alguns padrões interessantes. O padrão 21 denota o deslocamento mais intenso de saída da favela Nova Holanda (itemGid 293) pelo período da tarde, em direção a outras regiões desconhecidas ou à favela Vila Nova Esperança (itemGid 481), como mostra o padrão 34. O padrão 18 captura o fluxo no período da manhã saindo de Parque Jardim Beira-Mar (itemGid 321), possivelmente indicando engarrafamentos nas saídas próximas.

4.2 Bairros do Rio de Janeiro

Neste experimento, será possível analisar o fluxo das trajetórias entre os bairros da cidade durante os diferentes períodos do dia. Para a agregação da semântica de bairros às trajetórias, foram utilizadas as mesmas 500 trajetórias de amostra dos experimentos da seção 4.1. A consulta apresentada na tabela 4.3 foi utilizada para a mineração dos padrões seqüenciais, considerando itens formados pelo *nome dos bairros* mais o *período do dia em que iniciaram*.

Tabela 4.3: Consulta para minerar padrões em bairros do Rio com suporte mínimo de 10%, considerando o período do dia em que cada *stop* da trajetória iniciou

```
SELECT sequentialStops('NameStart', 'INSTANCE',
'07:00,12:00,manha|12:00,18:00,tarde|18:00,23:00,noite', 0.1);
```

Row	Pid (int8)	Size (int4)	Item (varchar)
1	21	2	bairro_139_tarde
2	21	2	bairro_140_tarde
3	32	2	bairro_139_tarde
4	32	2	bairro_140_noite
5	52	3	bairro_139_tarde
6	52	3	bairro_36_tarde
7	52	3	bairro_140_noite

Figura 4.3: Padrões gerados em bairros do Rio de Janeiro com suporte mínimo de 10%, considerando o período do dia em que cada *stop* da trajetória iniciou

Os padrões gerados por esta consulta, apresentados parcialmente na figura 4.3,

representam comportamentos das movimentações entre bairros, agrupados por períodos do dia.

Pode-se notar um intenso fluxo de saída da Cidade Universitária (itemGid 139). Pela tarde, no mínimo 10% das pessoas que passam por este bairro dirigem-se para a Lagoa (itemGid 36), deslocando-se posteriormente para a Barra da Tijuca (itemGid 140) à noite, como mostrado no padrão 52. O padrão 21 mostra o mesmo movimento da Cidade Universitária para a Barra da Tijuca pela tarde, enquanto que o deslocamento entre a tarde e a noite é detectado pelo padrão 32.

De forma análoga ao experimento feito com favelas, será feito um detalhamento mais minucioso dos padrões seqüenciais entre bairros, utilizando-se itens formados também pelo período do dia em que terminaram os *stops*. A tabela 4.4 exibe a consulta utilizada para esta mineração.

Tabela 4.4: Consulta para minerar padrões em bairros do Rio com suporte mínimo de 10%, considerando o período do dia em que cada *stop* da trajetória iniciou e terminou

```
SELECT sequentialStops('NameStartEnd', 'INSTANCE',
'07:00,12:00,manha|12:00,18:00,tarde|18:00,23:00,noite', 0.1);
```

Com esta consulta, é possível detectar com maior precisão os movimentos de um bairro para outro, pois também são observados os horários de entrada e saída de cada bairro (*NameStartEnd*), como se pode ver através da figura 4.4, que exibe alguns resultados interessantes da mineração.

Row	Pid (int8)	Size (int4)	Item (varchar)
1	18	2	bairro_126_manha_manha
2	18	2	bairro_140_manha_manha
3	34	2	bairro_134_tarde_tarde
4	34	2	bairro_140_tarde_tarde
5	94	4	UNKNOWN_1_tarde_tarde
6	94	4	bairro_134_noite_noite
7	94	4	bairro_151_noite_noite
8	94	4	bairro_140_noite_noite

Figura 4.4: Padrões gerados em bairros do Rio de Janeiro com suporte mínimo de 10%, considerando o período do dia em que cada *stop* da trajetória iniciou e terminou

Pela parte da noite, existe um fluxo partindo da Gávea (itemGid 134), passando por São Conrado (itemGid 151) e chegando à Barra da Tijuca (itemGid 140). Este fluxo partiu de um ponto desconhecido, de tarde, como demonstra o padrão 94. Pela forma do item, contendo períodos de início e fim iguais (*noite_noite*), toda trajetória que suporta este padrão passa por estes bairros antes entre as 18h e 23h. O padrão 18 indica uma movimentação de São Cristóvão (itemGid 126) para a Barra da Tijuca somente pela manhã, e pela tarde, existe um fluxo da Gávea para a Barra da Tijuca, como demonstrado pelo padrão 34.

A mineração executada através do comando exibido na tabela 4.5 tem como objetivo obter os padrões de congestionamentos somente na *hora do rush*, entre as 7h e 9h e também entre as 17h30 e 19h. Foi utilizado um suporte mínimo de 20% para reforçar pontos de grande parada no trânsito da cidade.

Tabela 4.5: Consulta para minerar padrões em bairros do Rio com suporte mínimo de 20%, considerando a *hora do rush* da cidade

```
SELECT sequentialStops('NameStart', 'INSTANCE',
'07:00,09:00,horaRushManha|17:30,19:00,horaRushTarde', 0.2);
```

Row	Pid (int8)	Size (int4)	Item (varchar)
1	6	2	bairro_36_horaRushManha
2	6	2	bairro_140_horaRushManha
3	8	2	bairro_36_horaRushManha
4	8	2	bairro_140_horaRushTarde
5	35	3	bairro_41_horaRushTarde
6	35	3	bairro_151_horaRushTarde
7	35	3	bairro_140_horaRushTarde

Figura 4.5: Padrões gerados em bairros do Rio de Janeiro com suporte mínimo de 20%, considerando a *hora do rush* da cidade

A figura 4.5 exibe alguns padrões interessantes obtidos deste experimento. Pela manhã, existe congestionamento entre a Lagoa (itemGid 36) e a Barra da Tijuca (itemGid 140), como demonstra o padrão 6. O padrão 8 vai além, e indica que o trajeto entre os dois bairros é congestionado durante todo o dia, da manhã até a tarde, para pelo menos 20% das trajetórias coletadas pelo carro sonda. Pela tarde, existe um fluxo intenso partindo do Leblon (itemGid 41), passando por São Conrado (itemGid 151) e chegando à Barra da Tijuca.

4.3 Refinando resultados

Para demonstrar as vantagens do armazenamento dos padrões gerados, serão realizadas consultas sobre os resultados obtidos nos experimentos realizados neste capítulo, refinando o processo de mineração de dados.

4.3.1 Consulta de padrões sobre instâncias específicas

Considerando-se os padrões obtidos através da mineração executada na tabela 4.1, um usuário poderia estar interessado em obter somente os padrões que aconteceram entre as favelas Rubens Vaz (itemGid 395) ou Nova Holanda (itemGid 293), por exemplo. Para isso, seria realizada uma consulta sobre os padrões armazenados, selecionando aqueles que envolvem os identificadores de instâncias correspondentes às favelas em questão. Esta consulta é exibida na tabela 4.6. Um subconjunto dos padrões é retornado, como mostrado na figura 4.6.

Row	Pid (int8)	Item (varchar)
1	14	favela_395
2	14	UNKNOWN_1
3	28	UNKNOWN_1
4	28	favela_293
5	28	favela_481
6	29	favela_293
7	29	UNKNOWN_1
8	29	favela_481
9	30	favela_293
10	30	favela_481
11	30	UNKNOWN_1
12	31	UNKNOWN_1
13	31	favela_343
14	31	favela_293
15	32	UNKNOWN_1
16	32	favela_395
17	32	favela_293
18	33	favela_395
19	33	favela_293
20	33	UNKNOWN_1
21	34	favela_343
22	34	favela_293
23	34	UNKNOWN_1

Figura 4.6: Subconjunto dos padrões obtidos depois do refinamento espacial

Estes padrões obtidos através do refinamento definido pelo usuário poderiam ser utilizados como entrada para um novo ciclo de mineração, pois todas as informações a respeito dos itens existem na própria representação do padrão na tabela *pattern_item*. Na consulta da tabela 4.6 e também nas consultas seguintes desta seção, é feito um *join* com a tabela *item* para facilitar a exibição do nome de cada item, não sendo necessário realizar a concatenação de cada uma das partes. Portanto, esta operação é opcional para os refinamentos propostos.

Tabela 4.6: Consulta SQL para filtrar resultados, obtendo somente os padrões que envolvem instâncias específicas

```
SELECT pi.pid AS "Pid", "itens".item AS "Item" FROM pattern_item pi
LEFT JOIN (SELECT item, "itemId" FROM item GROUP BY item, "itemId") AS
"itens" ON pi."itemId" = "itens"."itemId" WHERE pi.pid IN (SELECT pid
FROM pattern_item WHERE "itemName" = 'favela' AND "itemGid" IN (395,
293) GROUP BY pid) ORDER BY pi.id;
```

4.3.2 Consulta de padrões sobre intervalos de tempo específicos

De forma similar à seção anterior, padrões específicos podem ser obtidos através da definição de restrições temporais. No exemplo mostrado na tabela 4.7, são recuperados todos os padrões em que pelo menos um de seus itens inicia no turno da manhã. Os padrões filtrados são os obtidos através da mineração realizada pelo

comando descrito na tabela 4.3, portanto, instâncias de bairros serão consideradas. O subconjunto de padrões obtidos através do filtro aplicado nos padrões de bairros é mostrado na figura 4.7.

Tabela 4.7: Consulta SQL para filtrar resultados, obtendo somente os padrões que iniciam no turno da manhã

```
SELECT pi.pid AS "Pid", "itens".item AS "Item" FROM pattern_item pi
LEFT JOIN (SELECT item, "itemId" FROM item GROUP BY item, "itemId") AS
"itens" ON pi."itemId" = "itens"."itemId" WHERE pi.pid IN (SELECT pid
FROM pattern_item WHERE "itemStart" = 'manha' GROUP BY pid) ORDER BY
pi.id;
```

Estes exemplos mostram que a análise de padrões é facilitada. Os filtros dos padrões podem ser aplicados sobre qualquer parte do item. Isto reforça a importância do armazenamento dos padrões em um formato flexível, capaz de abstrair as diferenças entre a natureza dos dados.

Outro refinamento possível seria obter, a partir dos padrões gerados pela consulta da tabela 4.4, todos os padrões que envolvam itens iniciando pela manhã e terminando à tarde, visto que cada item é formado por seu período de início e fim neste exemplo.

Row	Pid (int8)	Item (varchar)
1	1	bairro_140_manha
2	4	bairro_39_manha
3	16	UNKNOWN_1_manha
4	16	bairro_126_manha
5	17	UNKNOWN_1_manha
6	17	bairro_140_manha
7	18	bairro_126_manha
8	18	bairro_140_manha
9	19	bairro_36_manha
10	19	bairro_140_manha
11	20	bairro_126_manha
12	20	UNKNOWN_1_manha

Figura 4.7: Subconjunto dos padrões obtidos depois do refinamento temporal

4.3.3 Consulta de padrões usando atributos não-espaciais como filtros

No momento em que as trajetórias brutas são transformadas em trajetórias semânticas, através dos métodos apresentados em (CHIECHELSKI, BOGORNY, 2008) e utilizados neste trabalho, são utilizadas tabelas que representam os objetos relevantes para a aplicação. No caso dos experimentos realizados neste capítulo, foram utilizadas tabelas contendo todas as instâncias de favelas e bairros do Rio de Janeiro.

Row	Pid (int8)	Item (varchar)
1	3	favela_321
2	3	favela_338
3	13	favela_343
4	13	UNKNOWN_1
5	15	favela_321
6	15	favela_495
7	15	UNKNOWN_1
8	16	UNKNOWN_1
9	16	favela_321
10	16	favela_338

Figura 4.8: Subconjunto dos padrões obtidos depois do refinamento feito através de um atributo específico da tabela *favela*

Nestas tabelas, além das informações geográficas utilizadas pelos algoritmos de pré-processamento para a geração dos *stops*, existem diversos atributos não-espaciais que descrevem e detalham estes objetos. Tais atributos também podem ser utilizados para o refinamento dos resultados obtidos pela mineração.

Tabela 4.8: Consulta SQL para filtrar resultados, obtendo todos os padrões que contenham pelo menos uma favela representada por mais de 10.000 pessoas na aplicação

```
SELECT pi.pid AS "Pid", "itens".item AS "Item" FROM pattern_item pi
LEFT JOIN (SELECT item, "itemId" FROM item GROUP BY item, "itemId") AS
"itens" ON pi."itemId" = "itens"."itemId" WHERE pi.pid IN (SELECT pid
FROM pattern_item WHERE "itemName" = 'favela' AND "itemGid" IN (SELECT
gid FROM favela WHERE totalpesso >= 10000) GROUP BY pid) ORDER BY
pi.id;
```

Como exemplo, será utilizado um campo da tabela *favela*, chamado *totalpesso*, que contabiliza o número de pessoas de cada favela relevantes para a aplicação. Para o filtro, serão considerados os padrões gerados pela consulta da tabela 4.1 que envolvam pelo menos uma favela contendo mais do que 10.000 pessoas. A tabela 4.8 exibe a consulta utilizada para este refinamento, enquanto que a figura 4.8 mostra uma parcela dos padrões obtidos.

5. Conclusão

A área de mineração de dados espaço-temporais não dispõe de uma solução completa e indiscutível para a mineração de trajetórias. Na verdade, grande parte do problema reside no fato de que a maioria dos trabalhos existentes considera somente trajetórias brutas nos seus processos de extração de conhecimento, ou seja, operam somente sobre pontos no espaço, dispostos em intervalos de tempo. Estes dados brutos, além de possuírem pouca capacidade de expressão para os usuários, são mais trabalhosos de manipular.

Trabalhos realizados no grupo de mineração de dados espaço-temporais da UFRGS (BOGORNY, KUIJPERS, ALVARES, 2008) (ALVARES *et al.*, 2007a) (PALMA *et al.*, 2008) mostraram que a semântica tem papel fundamental na mineração de trajetórias. Baseado nestas constatações, este trabalho realiza a mineração sobre trajetórias semânticas, que são geradas a partir dos dois métodos descritos em (CHIECHELSKI, BOGORNY, 2008).

Através da integração do algoritmo desenvolvido neste trabalho de conclusão com os algoritmos de agregação de semântica às trajetórias de (CHIECHELSKI, BOGORNY, 2008), foi construída uma linguagem completa para a mineração de dados espaço-temporais. Esta linguagem visa ajudar a preencher a lacuna existente nesta área, uma vez que existem poucas ferramentas dedicadas a este propósito.

O WEKA (Site do WEKA) possui uma extensão denominada WEKA-GDPM, proposta em (BOGORNY *et al.*, 2006), que representa uma solução alternativa para a mineração de trajetórias semânticas. Esta ferramenta, apesar de apropriada para esta tarefa, não deixa de ser um módulo externo, obrigatório para o usuário que pretende minerar dados utilizando esta abordagem. O cenário piora quando estes dados encontram-se em bancos de dados remotos, necessitando de uma comunicação através de uma rede para transferência e utilização dos mesmos.

Uma das principais contribuições deste trabalho refere-se à implementação do processo de mineração exclusivamente em SQL, permitindo que a extração de padrões seja realizada no próprio banco de dados, sem a necessidade de ferramentas externas. Os dados e o módulo de mineração encontram-se no mesmo local, facilitando a execução do processo e também a sua realimentação. Além disso, devido à padronização e maturidade do SQL, este algoritmo pode ser incorporado na maioria dos SGBDs atuais.

Outro aspecto interessante deste trabalho é a possibilidade de mineração dos dados em diversos níveis de granularidade espacial e temporal. Trajetórias podem ser capturadas em uma infinidade de situações e para aplicações diferentes. Um usuário interessado em minerar dados sobre o tráfego urbano de uma cidade provavelmente seleciona elementos comuns a este ambiente, como ruas, bairros, praças, hospitais, etc. Além disso, padrões interessantes são obtidos ao analisar comportamentos em diferentes dias da semana, ou horários de pico. Já um cientista analisando o ciclo de vida de uma

espécie de aves está interessado em padrões envolvendo meses, anos, até mesmo décadas. Os aspectos espaciais também são diferentes, considerando características do solo e clima de regiões, países ou continentes.

Como as trajetórias são formadas por elementos de espaço e tempo, a mineração sobre estes dados deve ser flexível ao ponto de permitir que o usuário configure o tipo dos padrões que espera obter. Além deste trabalho de conclusão, existe somente um trabalho (BOGORNY, KUIJPERS, ALVARES, 2008) que permite tais ajustes por parte do usuário, até o presente momento.

A geração e a apresentação dos padrões geralmente é a última etapa do processo de extração de conhecimento dos métodos convencionais, como o WEKA. Este trabalho permite a realização de uma etapa de pós-processamento, através do armazenamento dos padrões no próprio banco de dados e da possibilidade de execução de consultas sobre estes padrões, permitindo o refinamento dos resultados. Em alguns ciclos de mineração, através da realimentação dos dados a serem minerados a partir de subconjuntos dos padrões gerados nas iterações anteriores, é possível obter padrões extremamente precisos acerca dos aspectos buscados pelo usuário.

O trabalho desenvolvido apresentou um desempenho satisfatório nos diversos testes realizados com dados reais. Mesmo assim, melhorias podem ser feitas em nível de banco de dados, através da criação de índices específicos nas tabelas mais acessadas pelo algoritmo, do armazenamento dos dados auxiliares em tabelas temporárias e da utilização de funções específicas de SGBDs para o aumento da eficiência. Para trabalhos futuros, sugere-se também o suporte a diferentes tipos de granularidades temporais como dias da semana ou meses, aumentando a liberdade do usuário para trabalhar com agrupamentos de tempo, e a incorporação de granularidades espaciais em múltiplos níveis, implementando hierarquias de conceito.

REFERÊNCIAS BIBLIOGRÁFICAS

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A., **Mining association rules between sets of items in large databases**. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 207-216, Washington, D.C., May, 1993.

AGRAWAL, R.; SRIKANT, R., 1994, **Fast algorithms for mining association rules in large databases**. In VLDB, pages 487–499.

AGRAWAL, R.; SRIKANT, R., 1995, **Mining Sequential Patterns**. In *Proceedings of the ICDE*, P.S. Yu and A.L.P. Chen (Eds) (IEEE Computer Society), pp. 3-14.

ALVARES, L. O.; BOGORNY, V.; KUIJPERS, B.; MACEDO, J. A. F.; MOELANS, B.; VAISMAN, A., 2007a, **A Model for Enriching Trajectories with Semantic Geographical Information**. In: Proc. of the ACM 15th International Symposium on Advances in Geographic Information Systems (ACM-GIS'07), Seattle, Washington, November (2007). pp. 162-169.

ALVARES, L. O.; BOGORNY, V.; MACEDO, J. F. de; MOELANS, B., 2007b, **Dynamic Modeling of Trajectory Patterns using Data Mining and Reverse Engineering**. In Proceedings of the Twenty-Sixth International Conference on Conceptual Modeling - ER2007 - Tutorials, Posters, Panels and Industrial Contributions, 83, Auckland, New Zealand, November (CRPIT), pp. 149-154.

AYRES, J.; FLANNICK, J.; GEHRKE, J.; YIU, T., **Sequential PAttern mining using a bitmap representation**. Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 429--435, 2002.

BOGORNY, V.; KUIJPERS, B.; ALVARES, L.O., **ST-DMQL: A Semantic Trajectory Data Mining Query Language**. In: International Journal of Geographical Information Science (2008). Taylor and Francis (in press).

BOGORNY, V.; PALMA, A. T.; ENGEL, P.; ALVARES, L. O., 2006, **Weka-GDPM: Integrating Classical Data Mining Toolkit to Geographic Information Systems**. In Proceedings of the WAAMD Workshop, Florianopolis, Brazil (SBC), pp. 9-16.

CHIECHELSKI, G. O; BOGORNY, V., **Uma Extensão do PostGIS para a Geração Automática de Trajetórias Semânticas**. Instituto de Informática da UFRGS, Porto Alegre, Brasil. Trabalho de conclusão de curso, pp.61, 2008.

DATE, C. J., **Introdução a sistemas de banco de dados**; tradução de Daniel Vieira. - Rio de Janeiro: Elsevier, 2003 - 4ª Reimpressão.

EISENBERG, A.; MELTON, J., **"SQL:1999, Formerly Known as SQL3"**, ACM SIGMOD Record 28, Número 1 (março de 1999).

HWANG, S.-Y.; LIU, Y.-H.; CHIU, J.-K.; LIM, E.-P., 2005, **Mining mobile group patterns: A trajectory-based approach**. In Ho, T. B., Cheung, D. W.-L., and Liu, H., editors, PAKDD, volume 3518 of Lecture Notes in Computer Science, pages 713–718. Springer.

LAUBE, P.; IMFELD, S., 2002, **Analyzing relative motion within groups of trackable moving point objects**. In Egenhofer, M. J. and Mark, D. M., editors, GIScience, volume 2478 of Lecture Notes in Computer Science, pages 132–144.

Springer.

PALMA, A.; BOGORNY, V.; KUIJPERS, B.; ALVARES, L.O., **A clustering-based approach for finding interesting places in trajectories**. In: 23rd Annual ACM Symposium on Applied Computing, 2008, Fortaleza. Proceedings of the 2008 ACM Symposium on Applied Computing. New York: ACM Press, 2008. p. 863-868.

PEI, J.; HAN, J.; MORTAZAVI-ASL, B.; PINTO, H.; CHEN, Q.; DAYAL, U.; HSU, M. C., **PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth**, icde, pp.0215, 17th International Conference on Data Engineering (ICDE'01), 2001.

REZENDE, S. O.; PUGLIESI, J. B.; MELANDA, E. A.; *et al.*; 2003. **Mineração de Dados**. In: REZENDE, S. O.; *Sistemas Inteligentes - Fundamentos e Aplicações*. 1ª Edição. Barueri, SP: Manoele, 2003. p. 307-335.

SPACCAPIETRA, S.; DAMIANI, M. L.; MACEDO, J. A. F. de; PARENT, C.; PORTO, F.; VANGENOT, C., 2007, **A Conceptual View on Trajectories**. Technical report, Ecole Polytechnique Federal de Lausanne.

SPACCAPIETRA, S.; PARENT, C.; DAMIANI, M. L.; MACEDO, J. A. de; PORTO, F.; VANGENOT, C.; **A conceptual view on trajectories**, Data & Knowledge Engineering, v.65 n.1, p.126-146, April, 2008

WANG, H.; ZANIOLO, C., 2003, **ATLaS: A Native Extension of SQL for Data Mining**. In Proceedings of the SDM, D. Barbará and C. Kamath (Eds) (SIAM).

WANG, H.; ZANIOLO, C.; LUO, C. R., 2003, **ATLAS: a small but complete SQL extension for data mining and data streams**. Proceedings of the 29th international conference on Very large data bases-Volume 29, pages 1113—1116.

Referências na Web

Algoritmo Apriori na Wikipédia: http://en.wikipedia.org/wiki/Apriori_algorithm

Mineração de Dados na Wikipédia: http://pt.wikipedia.org/wiki/Minera%C3%A7%C3%A3o_de_dados

Regras de Associação na Wikipédia: http://en.wikipedia.org/wiki/Association_rules

Site do WEKA: http://www.cs.waikato.ac.nz/~ml/weka/

Site oficial do banco de dados PostgreSQL: http://www.postgresql.org/

SQL na Wikipédia: http://pt.wikipedia.org/wiki/SQL