Universidade Estadual de Londrina

PEDRO SENA TANAKA

ALGORITMOS EFICIENTES PARA A DETECÇÃO ON-LINE DO PADRÃO FLOCO EM BANCOS DE DADOS DE TRAJETÓRIAS

Londrina
2016

PEDRO SENA TANAKA

ALGORITMOS EFICIENTES PARA A DETECÇÃO ON-LINE DO PADRÃO FLOCO EM BANCOS DE DADOS DE TRAJETÓRIAS

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Daniel dos Santos Kaster.

Coorientador: Dr. Marcos Rodrigues Vieira.

Londrina
2016

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Tanaka, Pedro Sena.
Algoritmos Eficientes para detecção on-line do padrão Floco em Bancos de dados de Trajetórias / Pedro Sena Tanaka. - Londrina, 2016.
85 f. : il.

Orientador: Daniel dos Santos Kaster.
Coorientador: Marcos Rodrigues Vieira.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2016.
Inclui bibliografia.

1. Banco de dados - Teses. 2. Detecção de Padrões - Teses. 3. Padrões Espaço-temporais - Teses. I. Kaster, Daniel dos Santos. II. Vieira, Marcos Rodrigues . III. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

PEDRO SENA TANAKA

ALGORITMOS EFICIENTES PARA A DETECÇÃO ON-LINE DO PADRÃO FLOCO EM BANCOS DE DADOS DE TRAJETÓRIAS

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

_____
Orientador: Prof. Dr. Daniel dos Santos Kaster
Universidade Estadual de Londrina - UEL

_____
Profa. Dra. Vania Bogorny
Universidade Federal de Santa Catarina- UFSC

_____
Prof. Dr. Sylvio Barbon Júnior
Universidade Estadual de Londrina – UEL

_____
Prof. Dr. Mario Lemes Proença Júnior
Universidade Estadual de Londrina - UEL

Londrina, 11 de maio de 2016.

Dedico este trabalho às crianças que não puderam ter uma educação de qualidade.
Àqueles que me apoiaram sempre e sonharam comigo até que mais esta etapa se realizasse.
Aos meus pais, minha esposa e meu filho pelo carinho e suporte incondicional.

AGRADECIMENTOS

Agradeço primeiramente a Deus, aquEle pelo qual eu vivo, por me permitir realizar mais este sonho, por me abençoar e me guiar através dos conselhos de cada uma das pessoas que ele colocou no meu caminho durante o programa.

Agradeço ao meu núcleo familiar mais próximo, minha esposa e meu filho, que a cada dia, não só auxiliam a me tornar a melhor versão de mim mesmo, mas também me apoiam de maneira incondicional. Agradeço meus pais, por sempre sonharem comigo e acreditarem em mim por toda a minha vida, independente do projeto que viesse a empreender.

Agradeço ao meu co-orientador, por ideias e *insights* sem os quais este trabalho não seria possível. Pela atenção e dedicação mesmo em meio fuso-horário, vida corrida de empresa e vida pessoal. Tenho imensa gratidão a ti, Marcos Vieira.

Agradeço ao meu orientador pela ajuda que foi além da esperada numa relação de orientador – orientando. Pelos conselhos sobre gerenciamento de tempo desde a época da graduação e por me fazer um melhor pesquisador. Pelos conselhos sobre carreira, família e tantos outros assuntos que me ajudaram a crescer como pessoa também. Muito obrigado Daniel Kaster.

Agradeço também cada um dos meus amigos e parceiros de mestrado, e em especial Guilherme Henrique que me ajudou com ideias e me ajudou a entender melhor a importância da qualidade em tudo que se faz, *principalmente se for código*. E aos parceiros Sean Alvarenga e Rodrigo Igawa que me incentivaram bastante, principalmente neste último semestre.

Agradeço à CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pela concessão da bolsa durante todo o período de realização deste mestrado.

Agradeço à Universidade Estadual de Londrina por me acolher como aluno desde o período da graduação.

E por fim, agradeço aos amigos e familiares de forma geral geral que me ajudaram e apoiaram durante a realização deste trabalho. Muito obrigado!

"Sua jornada moldou você para seu bem maior,
e foi exatamente o que precisava ser.
Não pense que você perdeu tempo.
Não existem atalhos para a vida.
Foi necessária cada e toda situação que você
encontrou para trazê-lo para o agora.
E agora é o momento certo."
Asha Tyson

TANAKA, Pedro Sena. **Algoritmos eficientes para a detecção on-line do padrão floco em bancos de dados de trajetórias**. 85f. 2016. Dissertação (Mestrado em Ciência da Computação) — Universidade Estadual de Londrina, Londrina, 2016.

RESUMO

A alta disponibilidade, baixo custo e crescente utilização de dispositivos de localização fez aumentar o interesse por pesquisas na área de padrões espaço-temporais. O principal objetivo em estudar tais padrões é descobrir relacionamentos espaciais entre objetos móveis e entender como eles se desenvolvem ao decorrer do tempo. Trabalhos recentes da literatura propuseram uma grande variedade destes padrões, entre eles está o padrão floco. Este padrão consiste em identificar se um dado número de entidades se movimentam por um certo período de tempo próximas entre si, isto é, se encontram-se em um disco de raio predefinido em instantes subsequentes de tempo. Exemplos típicos de aplicação incluem a monitoração e vigilância uma vez que ambas dependem da identificação ágil de grupos suspeitos formados por pessoas/veículos em fluxos volumosos de dados espaço-temporais. Trabalhos da literatura propuseram algoritmos que resolvem o problema do padrão floco com tempo fixo em tempo polinomial. Nesta dissertação é proposto um novo algoritmo on-line para a detecção de flocos com tempo fixo chamado PSI, que aplica a técnica de geometria computacional varredura de plano, além das técnicas de assinaturas binárias e a estrutura índice invertido. Além do método PSI, este trabalho propõe uma família de algoritmos, todos baseados no método proposto na literatura chamado BFE. Todos estes métodos foram testados de forma extensiva utilizando conjuntos de dados reais e sintéticos e os resultados obtidos comprovam que as técnicas propostas no trabalho geram ganhos consideráveis quando aplicadas aos algoritmos de descoberta de flocos. Para melhor entender os resultados e apontar os melhores e piores cenários para a aplicação das técnicas foi feita uma análise aprofundada da relação entre os resultados e a distribuição e outras características dos dados.

Palavras-chave: Bancos de dados espaço-temporais. Padrão floco. Detecção de padrões. Varredura de plano.

TANAKA, Pedro Sena. **Efficient algorithms for online discovery of Flock pattern in trajectorial databases.** 85p. 2016. Dissertation (Master in Science in Computer Science) — State University of Londrina, Londrina, 2016.

ABSTRACT

The high availability, low cost and increasing usage of location-aware devices have increased the interest in the research of spatiotemporal patterns. The main goal in studying such patterns is to discover spatial relationships over time between moving objects. Recent articles have proposed a wide variety of such patterns, among them is the flock pattern. This pattern is defined as a set of moving objects with minimum size that stay together within a maximum distance for a continuous period of time. Typical application examples are monitoring and surveillance that both rely on efficiently identifying groups of suspicious people/vehicles in large spatiotemporal streaming data. Previous works proposed polynomial-time algorithms to the flock pattern problem with fixed time duration. This master thesis proposes a new online method, called PSI, which is an improved base method to discover flock patterns that applies the plane sweeping computational geometry technique along with binary signatures and inverted indexes. In addition to PSI, this work proposes an algorithm family also based in the baseline algorithm proposed in literature, namely BFE, as well as variations based in heuristics. All the methods proposed in the work were extensively evaluated using real and synthetic datasets and the obtained results show that the techniques proposed in this work generate high performance gains when applied to the problem of finding flock patterns. Lastly, in order to better understand the results and to indicate a the best and worse scenarios for the utilization of the techniques this work also includes a profound analysis of the relationship between results and data distribution, and other data characteristics.

Keywords: Spatiotemporal databases. Flock pattern. Pattern detection. Plane sweep.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GPS	*Global Positioning System*
GSM	*Global System for Mobile Communications*
OID	*Object Identified*
MPF	Mineração de Padrões Frequentes
TED	Teste e Divisão
PCA	*Principal Component Analysis*
TWD	Transformada Wavelet Discreta
SQT	*Skip Quadtree*
MBR	*Minimum Bounding Rectangle*
SATA	*Serial AT Attachment*
RAM	*Random Access Memory*

SUMÁRIO

1 INTRODUÇÃO

Recentemente, dispositivos de localização com suporte ao GPS (*Global Positioning System*) como rastreadores, *tags* RFID e principalmente dispositivos móveis (*smartphones*) se popularizaram ao passo que os mesmos se tornaram mais baratos. Essa popularização em combinação com o uso ubíquo de serviços de localização como Swarm [3] (com mais de 50 milhões de usuários e mais de 8 bilhões de *check-ins* mensais) e Waze[1] está gerando um crescimento rápido e contínuo de conjunto de dados em formas de trajetórias. Uma trajetória representa uma sequência de posições gravadas durante um tempo para um objeto móvel específico.

Uma característica interessante desses dados é que eles não se limitam a grandes repositórios de dados históricos, mas também são encontrados no formato de fluxos espaço-temporais oferecidos por serviços *on-line*. Um exemplo de tal serviço é o AccuTracking[2], que ajuda grandes empresas do varejo e de entregas (por exemplo o Serviço Postal dos Estados Unidos com uma das maiores frotas veiculares do mundo) a gerenciar a sua logística [4].

Devido a essa alta disponibilidade de dados espaço-temporais existe uma crescente demanda por algoritmos eficientes que consigam analisar esses grandes repositórios de informação. Entretanto, a tarefa de descobrir padrões espaço-temporais em grandes volumes de dados é muito desafiadora. Geralmente quando se fala em detecção de padrões consegue-se fazer a detecção com métodos puramente numéricos, como aprendizado de máquina, algoritmos de inteligência artificial, entre outros. Estes algoritmos analisam variáveis que modelam os eventos no mundo real. Por exemplo, para a detecção de padrões de compra pode ser feita através da observação e análise dos itens comprados para cada cliente. E então consegue-se extrair, por exemplo, quais itens são comprados juntos com mais frequência e entre outros fatos [5].

Por outro lado, na detecção de padrões espaço-temporais o processo de extração de conhecimento é um pouco mais árduo, já que estes padrões são definidos através da modificação dos relacionamentos dos objetos móveis durante o tempo. Ainda que essa análise seja computacionalmente cara, ela pode revelar comportamentos comuns entre os objetos estudados em um dado período de tempo (por exemplo, padrões de migração de animais selvagens, padrões de tráfego em rede rodoviária e atividades suspeitas em áreas urbanas). Vários trabalhos na literatura propõem algoritmos que têm aplicações desde a busca por recorrência de eventos urbanos [6] até estudos de comportamento animal [7, 8].

Em se tratando de padrões de movimentação de grupos de entidades, os trabalhos

[1] <www.waze.com>
[2] <www.accutracking.com>

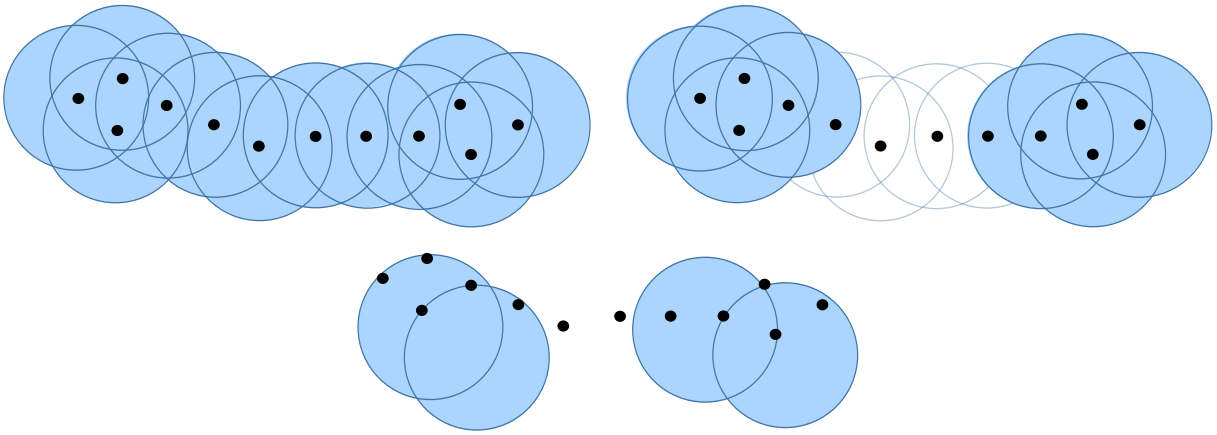


Figura 1 – Geração de clusters no algoritmo DBSCAN (parte superior) e geração de discos.

existentes podem ser classificados de uma forma simplificada como: padrões baseados em *cluster* e padrões baseados em distância/discos. Esta classificação diz respeito à forma como as entidades participantes de um certo padrão são agrupadas. Enquanto nos padrões baseados em *cluster* as entidades são agrupadas em grupos sem um formato rígido, nos padrões baseados em distância as entidades são agrupadas em discos com um raio bem definido.

Alguns trabalhos recentes em reconhecimento de padrões de movimentação focaram em padrões baseados em *cluster* (grupos [9, 10, 11], enxames [12], aglomerações [13]). Entre os motivos para este interesse está a simplicidade na geração de *clusters* [14] e a maior facilidade na detecção dos padrões, pois quando se usa a conectividade entre as entidades participantes do padrão não se exige que o grupo tenha um formato bem definido, em termos de distribuição no espaço, como é o caso dos padrões baseados em disco (ver Figura 1).

Entretanto, os padrões baseados em disco tem a propriedade particular que a distância máxima entre duas entidades quaisquer não excede o diâmetro do disco definido na busca. Isto é, os padrões baseados em disco estão interessados em entidades que satisfaçam uma dada relação de proximidade com todas as entidades pertencentes ao grupo e, por outro lado, os padrões baseados em *cluster* estão interessados em entidades que tenham relação de proximidade com um número de entidades do grupo. Desta forma, a semântica envolvida em cada um desses tipos de padrão atende a uma aplicação distinta.

Um dos padrões mais representativos da categoria de discos é o floco, que compreende um conjunto mínimo de objetos que estão espacialmente próximos por um intervalo de tempo. Dados um número de entidades μ, um raio de distância ϵ e um número de instantes de tempo δ, um floco é um conjunto de μ ou mais entidades que permanecem por pelo menos δ instantes de tempo localizadas no espaço definido por um disco de diâmetro ϵ. Algumas propostas da literatura focaram na descoberta de floco(s) com tamanho

(ou duração) máxima [15, 16], ou seja, conjunto(s) com no mínimo μ entidades dentro de um disco com diâmetro ϵ com a *maior duração no tempo possível*. Outros trabalhos propuseram algoritmos para encontrar flocos com uma duração fixa, isto é, a mesma definição acima, porém com *no mínimo* δ instantes de tempo. Assim como os algoritmos de padrão espaço-temporal são classificados como baseados em *cluster* e em distância, os métodos para descoberta de flocos com duração fixa podem, por sua vez, também sere classificados em duas categorias: **métodos *off-line***, por exemplo, [17, 18, 19], os quais demandam o conjunto de dados inteiro a ser processado de antemão; e **métodos *on-line***, que reportam resultados assim que eles são descobertos, possibilitando lidar com fluxos de dados.

A detecção de flocos de forma *on-line* é importante e tem uma grande variedade de aplicações, desde monitoração em tempo real de atividades suspeitas até observação de comportamento animal. Por exemplo, em um cenário onde um conjunto de carros está sendo rastreado, como parte de uma investigação de corridas ilegais, pelas forças policiais pode-se utilizar a detecção de flocos *on-line* para se lançar uma equipe de correção destes infratores. Como ilustrado na Figura 2, poderia se utilizar a detecção do padrão para reportar um agrupamento de carros investigados e assinalar com um alerta (entre os instantes de tempo t_1 e t_3). Isto lançaria a força policial, que, posteriormente poderia vir a interceptar os infratores. Existem outras aplicações tanto na área de segurança como na área de detecção de comportamento de condutores [20]. Entretanto, as aplicações não ficam restritas a área urbana, já que pode-se aplicadas a estudos de comportamento animal [21].

O trabalho de estado da arte em descoberta do padrão floco com duração fixa anterior à presente dissertação é [2], o qual apresenta um algoritmo básico chamado *Basic*

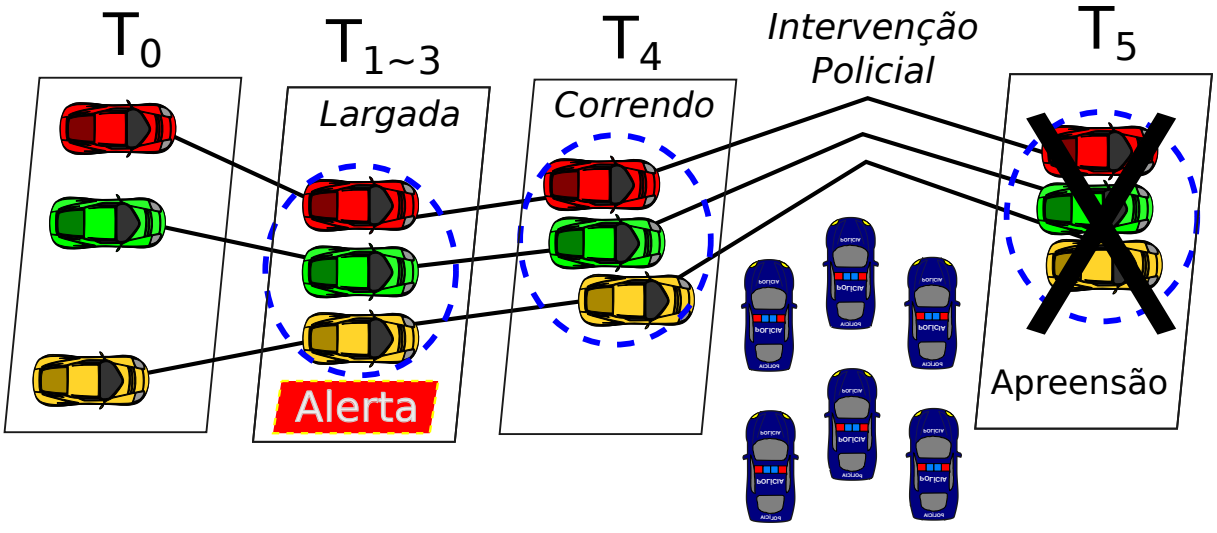


Figura 2 – Aplicação do floco on-line: lançamento automático de forças policiais para correção de infratores.

Flock Evaluation (BFE), e algumas variações com diferentes heurísticas para melhorá-lo. Esta dissertação aborda o problema de descoberta de flocos com duração fixa de forma *on-line*. A principal contribuição do trabalho foi a proposta do método chamado PSI – ***P****lane sweeping, binary* ***S****ignatures and* ***I****nverted indexes* – para descoberta *on-line* de flocos com tempo fixo. O PSI estende significativamente o algoritmo BFE aplicando um conjunto de técnicas/estruturas: **(1) varredura de plano (em inglês *plane sweeping*)** para detectar de forma rápida grupos de entidades que possam formar flocos em um determinado instante de tempo; **(2) assinaturas binárias** para reduzir o número de operações de interseção de conjuntos, e assim podar subconjuntos de candidatos em um dado instante de tempo; e **(3) índice invertido** para verificar se um dado floco da instância de tempo atual segue algum floco parcial da instância anterior.

Além deste método principal, este trabalho apresenta mais três variações de algoritmos para análise da aplicação isolada de cada uma das técnicas citadas no parágrafo anterior. Afim de investigar também o possível ganho de desempenho na aplicação das técnicas em outros métodos, foram desenvolvidos mais três variações de algoritmos baseados em heurísticas propostas em [2]. Por último, foi desenvolvida uma variação do método *LCM_Flock* [18], de modo a permitir o processamento *on-line* de dados, para que fosse comparado como um novo concorrente.

Para verificar a eficiência dos métodos propostos foram realizados testes em cinco conjuntos de dados (sendo quatro deles reais e um sintético). Durante os testes pôde se observar que as técnicas propostas oferecem alto ganho de desepmenho tanto se comparado ao algoritmo base (BFE), quanto se comparando ao algoritmo adaptado *LCM_Flock*. Para melhor entender os resultados colhidos foi feito também um estudo de obliquidade e dispersão dos conjuntos de dados. Sob a luz dessa análise aprofundada é possível apontar quando as técnicas obtêm os melhores resultados.

O restante desta dissertação está organizado da seguinte maneira. No Capítulo 2 apresenta-se os padrões de movimentação já propostos na literatura além de técnicas utilizadas para pré-processamento de dados em alguns algoritmos. O Capítulo 3 detalha os algoritmos propostos na literatura para detectar os padrões de movimentação citados no capítulo anterior além de introduzir algumas técnicas de computação geométrica e estruturas de dados para acelerar algoritmos espaço-temporais. O Capítulo 4 apresenta as principais contribuições do trabalho, a saber o método PSI e suas variações. Por fim, no Capítulo 5 são descritos os experimentos realizados para avaliar as propostas e no Capítulo 6 as conclusões da pesquisa e oportunidades de trabalhos futuros.

2 PADRÕES DE MOVIMENTAÇÃO

O crescimento da disponibilidade dos dados espaço-temporais fez aumentar o interesse e a importância da análise desse tipo de dado. Dados espaço-temporais têm fontes diversas advindas de vários domínios de aplicação como:

Biologia: movimentação animal, realocação animal e extinção [7, 8, 15];

Ecologia: rastreamento de incidentes de poluição e investigação de causas em mudanças ambientais[22, 6];

Transporte: planejamento de tráfego, rastreamento de veículos e monitoramento de tráfego [23, 24, 25, 26].

A compreensão de fenômenos espaço-temporais usualmente depende de processamento, análise e mineração de grande quantidade de dados espaço-temporais juntamente com atributos temporais e temáticos, em vários níveis de granularidade. A modelagem de acontecimentos espaço-temporais geralmente é árdua devido a dois fatores: o primeiro é que ocorrem mudanças, discretas e/ou contínuas, nos atributos espaciais e não espaciais dos objetos; o segundo é a influência de objetos vizinhos ao objeto principal do estudo. Por exemplo, considere-se a questão de como a chuva influencia em engarrafamentos em centros urbanos [7]. O posicionamento relativo de grupos de veículos define a ocorrência de engarrafamentos e suas características (extensão, duração, etc.). Por outro lado, a variação no índice pluviométrico no intervalo de tempo do estudo, bem como sua distribuição espacial, são dados importantes para a análise. Desta forma, é necessário fazer o cruzamento entre os dados pluviométricos e os padrões de movimentação dos veículos, considerando-se dados históricos, para obter-se resultados mais significativos.

A descoberta de conhecimento sobre dados espaço-temporais é um campo relativamente novo e é dedicado ao desenvolvimento e aplicação de técnicas para a análise de bancos de dados espaço-temporais. O objetivo principal é descobrir padrões ou relacionamentos existentes no banco de dados, mas que não estão armazenados de forma explícita [7, 27]. Este processo de descoberta envolve técnicas que lidam com ambos aspectos dos objetos espaço-temporais, o que torna a tarefa de mineração mais complicada do que quando utiliza-se apenas dados numéricos ou categóricos simples. Como existe uma grande correlação entre os dados espaço-temporais, possuindo complexidade de relacionamento e representação destes, técnicas clássicas de mineração de dados tendem a apresentar baixo desempenho quando aplicadas neste contexo visto a suposição de estacionalidade desses modelos. [28].

Os padrões de movimentação geralmente descrevem relacionamentos entre trajetórias em bancos espaço-temporais. Uma **trajetória** é uma sequência de pontos marcados com o local e tempo em que eles foram medidos, podendo ter outros atributos associados aos pontos. A partir de um conjunto de trajetórias é possível descrever padrões de agrupamentos entre as entidades que geram estas trajetórias. Estes padrões podem ser divididos em duas categorias: os genéricos, que analisam puramente a identificação de agrupamentos espaço-temporais; e comportamentais, mais voltados ao estudo de paradas e movimentos, que buscam o entendimento de qualquer tipo comportamental dos movimentos de objetos específicos. A detecção e a classificação destes padrões a partir de dados espaço-temporais são tarefas importantes no processo de compreensão do comportamento dos objetos de estudo [29].

Alguns tipos de padrões importantes são: floco (*flock*), liderança (*leadership*), reunião (*meeting*), padrão periódico (*periodic pattern*) e local frequente (*frequent location*) [1]. Alguns destes padrões podem ser observados na Figura 3. O padrão chamado floco é descrito por um número entidades se movimentando próximas entre si, por dado período de tempo, onde a proximidade é limitada através um disco de diâmetro ϵ [8, 18, 30]. O padrão liderança é parecido com o floco, porém nele um objeto (líder) é requerido a iniciar o movimento antes de um grupo que então o segue [31, 30, 21]. Já o padrão reunião, apesar de ser considerado espaço-temporal, marca a reunião espacial de um número de entidades. Por fim, os padrões periódico e local frequente referem-se a comportamentos repetitivos, onde o local frequente é um padrão definido por um local em que várias entidades passam no decorrer do tempo, e padrão periódico é um tipo de movimentação que ocorre de tempos em tempos [1]. Este capítulo apresenta o padrão floco, que é o objeto desta pesquisa (Seção 2.1), e os padrões de movimentação que apresentam a maior semelhança com o floco, a saber: o padrão grupo (Seção 2.2), o padrão enxame (Seção 2.3) e o padrão aglomeração (Seção 2.4).

2.1 O Padrão Floco

Atualmente, muitas pesquisas focadas na descoberta de padrões em dados de objetos móveis foram desenvolvidas, principalmente na área de descoberta de grupos de objetos movendo-se na mesma direção [30]. Por exemplo, na procura pelo entendimento do comportamento animal, muitos estudos da área de biologia têm usado coleiras equipadas com GPS-GSM [32, 8]. Estas coleiras geram posições marcadas com o tempo da amostra, gerando as trajetórias feitas pelos respectivos animais. Neste contexto, é importante reconhecer padrões que os animais possam estar formando, para então entender processos complexos de comportamento de bando, como processos migratórios ou de extinção [33].

Um dos padrões de movimentação utilizados nesses estudos é o padrão floco (*flock pattern*). Há algumas definições diferentes para o padrão floco na literatura. Por exemplo,

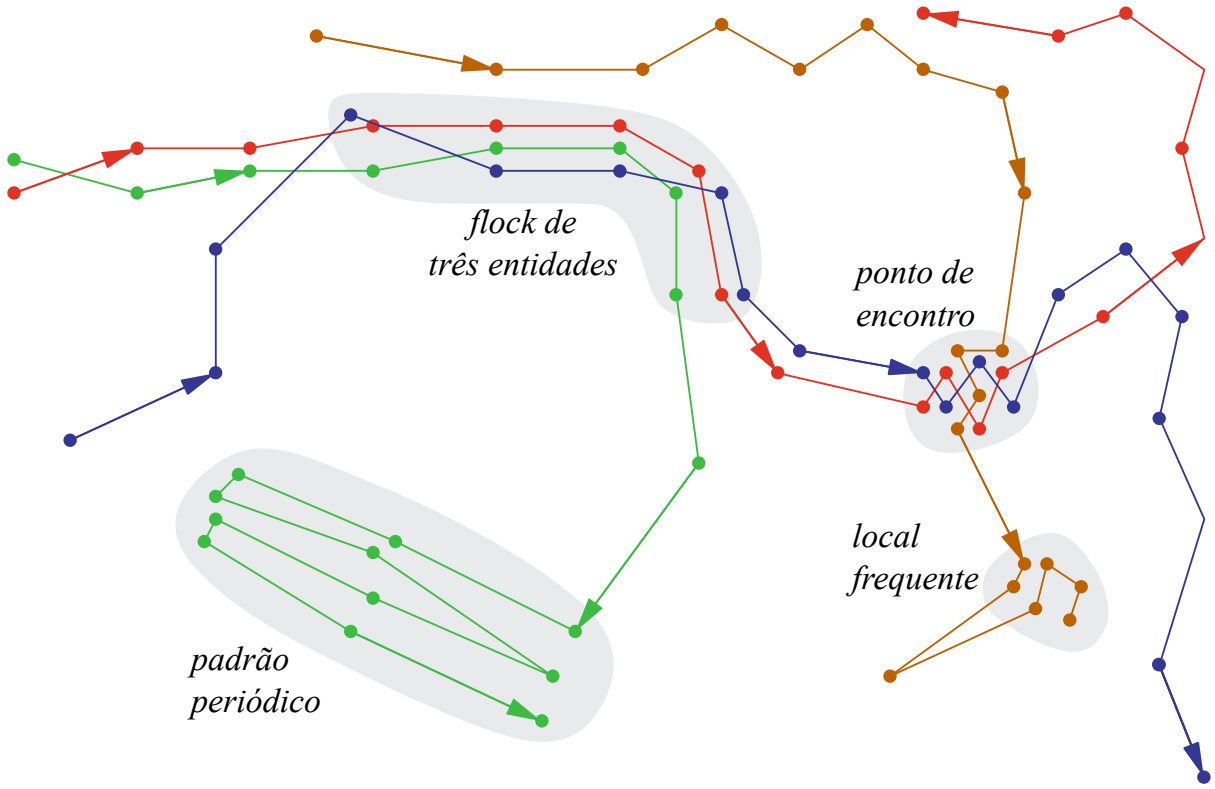


Figura 3 – Padrões de movimentação espaço-temporais (retirada de [1]).

no trabalho de Gudmundsson, Kreveld e Speckmann[34] é usada uma definição de floco em que se considera somente um instante de tempo. Entretanto, a definição mais utilizada define um floco como sendo um grupo de **m** entidades movendo-se em um disco de diâmetro ϵ por um intervalo **k** de tempo [2, 8, 16, 15]. Seguindo esse consenso, a Definição 1 estabelece formalmente o padrão floco.

Definição 1 (Padrão Floco)**.** *Dados um conjunto de trajetórias $\mathcal{T}$, um número mínimo de trajetórias $\mu > 1$ ($\mu \in \mathbb{N}$), uma distância máxima $\epsilon > 0$ definida por uma métrica de distância d e uma duração mínima de $\delta > 1$ instantes de tempo ($\delta \in \mathbb{N}$), um padrão* ***Floco***(μ, ϵ, δ) *reporta um conjunto $\mathcal{F}$ contendo todos os flocos f_k, que são conjuntos de trajetórias de tamanho máximo tais que: para cada $f_k \in \mathcal{F}$, o número de trajetórias é maior ou igual a μ e existem δ instâncias de tempo consecutivas $t_j, \ldots, t_{j+\delta-1}$ em que existe um disco com centro $c_k^{t_i}$ e diâmetro ϵ cobrindo todos os pontos das trajetórias de $f_k^{t_i}$, que é o floco f_k no instante t_i, $j \leqslant i \leqslant j + \delta$.*

A Figura 4 mostra um exemplo de padrão floco com duas ocorrências envolvendo três entidades, são elas: floco formado nas pelas trajetórias T_1, T_2 e T_3 nos discos c_1^1, c_1^2 e c_1^3 e floco composto por T_4, T_5 e T_6 nos discos c_2^2, c_2^3 e c_2^4. Os principais algoritmos propostos para a detecção de padrões floco, bem como as técnicas e estruturas de apoio utilizadas, serão apresentados no Capítulo 3.

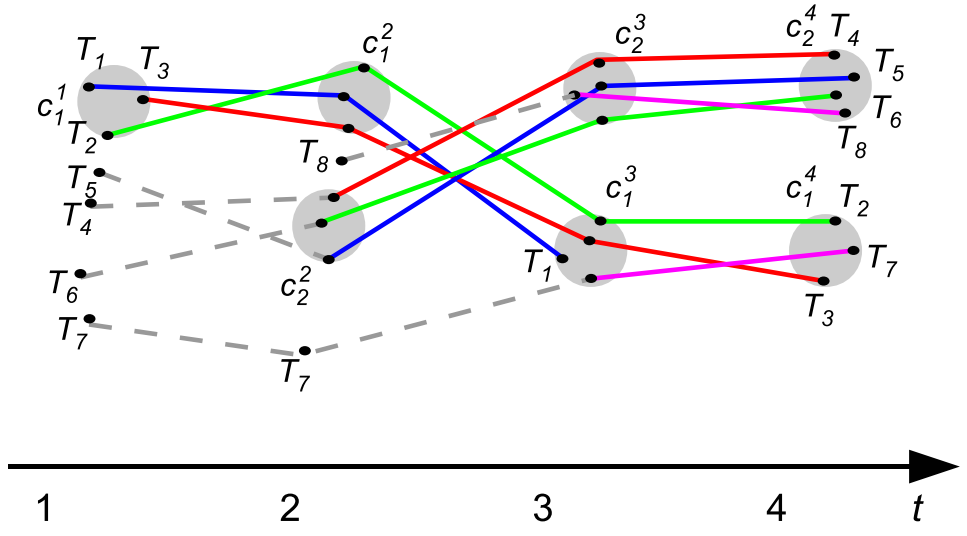


Figura 4 – Exemplo de padrão floco (retirada de [2]).

2.2 O Padrão Grupo

Um dos padrões que surgiram após o floco foi o padrão grupo (*group pattern*), que é baseado no princípio de densidade da conexidade. Assim como no padrão floco, não existe somente uma definição formal do padrão grupo, apesar das semânticas das diferentes definições serem parecidas.

A primeira definição desse padrão foi apresentada por Wang, Lim e Hwang[35]. Segundo este trabalho, um padrão grupo ocorre quando um conjunto de entidades permanece junto por um período de tempo, porém, ao contrário do definido no padrão floco, o formato do grupo não é limitado a um disco, mas definido por um agrupamento por densidade das entidades. Outra diferença considerável entre o padrão grupo e o floco é que o padrão grupo não requer um número de entidades participantes, mas somente a duração e a distância limite, uma vez que na detecção deste padrão todos os grupos de duas ou mais entidades são reportados. A quantidade de tempo que as entidades ficaram juntas é definida como um parâmetro de validação dos grupos. Este parâmetro tem o nome de peso (*weight*) do padrão P e é definido da seguinte maneira [35]:

$$weight(P) = \frac{\sum_{i=1}^{n} |S_i|}{N} \tag{2.1}$$

onde S_i são todos os segmentos em que as entidades andaram juntas e N é a quantidade total de segmentos nas trajetórias. Se $weight(P)$ é maior ou igual a um parâmetro min_weigth então o padrão de grupo P é válido.

Uma segunda definição de grupo foi apresentada por Li et al.[9], introduzindo

a restrição de cardinalidade das entidades e nominando essa variação do padrão como consulta de grupo. Logo, a consulta de grupo fica definida como um agrupamento definido por um número de entidades m e uma distância e que dura pelo menos t instâncias de tempo. A introdução do parâmetro m possibilitou soluções mais eficientes devido à poda constante de candidatos. Com base nessa modificação, os autores propuseram um *framework* para identificação *on-line* de grupos, utilizando uma abordagem independente de amostragem das trajetórias. No *framework* proposto é apresentado um algoritmo de clusterização contínua que, por sua vez, utiliza o algoritmo DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) [14] para encontrar agrupamentos de objetos se movendo juntos. Em uma segunda etapa, o *framework* detecta eventos que possam acontecer nas trajetórias entre objetos e agrupamentos ou interagrupamentos (separação, junção, saída de objeto).

2.3 O Padrão Enxame

Outro padrão sucessor do padrão floco é o padrão enxame (*swarm*). Este padrão se assemelha bastante ao padrão grupo, porém não é necessário que as instâncias de tempo onde o agrupamento ocorre sejam consecutivas. A definição do padrão enxame é como se segue. Seja $O_{DB} = \{o_1, o_2, \ldots, o_n\}$ o conjunto de objetos móveis no banco de dados e $T_{DB} = \{t_1, t_2, \ldots, t_m\}$ o conjunto de todas as instâncias de tempo possíveis no banco de dados e $C_{DB} = \{C_{t1}, C_{t2}, \ldots, C_{tm}\}$ o conjunto de todos os clusters que ocorrem entre as instâncias de tempo 1 e m. Um padrão enxame é um par (O, T), $O = \{oi_1, oi_2, \ldots, oi_p\} \subseteq O_{DB}, T \subseteq T_{DB}$, onde todos os elementos de O estão em um mesmo agrupamento em diferentes instâncias de tempo em T e estes dois conjuntos devem satisfazer as seguintes propriedades, dadas duas constantes min_o e min_t:

1. $|O| \geqslant min_o$, ou seja, deve haver pelo menos min_o entidades em O;
2. $|T| \geqslant min_t$, ou seja, os objetos em O estão em um mesmo agrupamento por pelo menos min_t instâncias de tempo;
3. $C_{t_i}(oi_1) \cap C_{t_i}(oi_2) \cap \ldots \cap C_{t_i}(oi_p) \neq \varnothing; \forall t_i \in T$, onde $C_{t_i}(oj)$ denota o conjunto de clusters no qual o objeto o_j participa na instância t_i. Isto é, existe pelo menos um cluster que contêm todos objetos em O em cada instância de T.

Uma das principais motivações que levaram à criação deste padrão foi o fato que perde-se formações de padrões de interesse quando se procura por padrões como floco ou grupo, onde é necessário que os instantes de tempo em que as entidades satisfazem ao critério de proximidade/conectividade sejam contínuos. Por exemplo, considerando o exemplo ilustrado na Figura 5, não há floco ou grupo contendo 2 ou mais entidades que permanecem próximas por pelo menos 3 instantes de tempo. Contudo, para

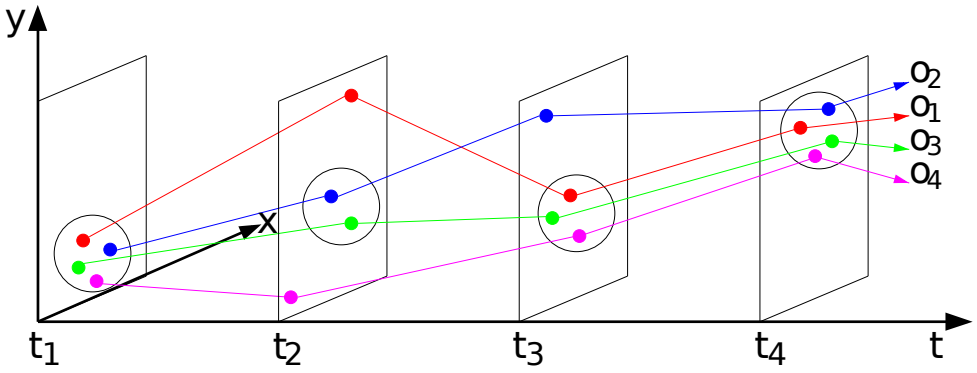


Figura 5 – Exemplo em que detecta-se um enxame mas não um floco com parâmetros equivalentes.

$min_o = 2$ e $min_t = 3$, detecta-se os enxames $(\{o_1, o_3\}, \{t_1, t_3, t_4\})$, $(\{o_1, o_4\}, \{t_1, t_3, t_4\})$, $(\{o_2, o_3\}, \{t_1, t_2, t_4\})$ e $(\{o_3, o_4\}, \{t_1, t_3, t_4\})$. É possível observar que não há a restrição de subsequência nas instâncias de tempo no padrão enxame, o que insere uma complexidade a mais neste problema. Devido ao fato da não continuidade dentro do intervalo de tempo, o problema de detecção de enxames possui complexidade polinomial na quantidade de objetos e instâncias presentes no banco $(2^{|O_D B|} * 2^{|T_D B|})$.

O primeiro trabalho a definir formalmente e propor soluções para este padrão propôs um algoritmo chamado *ObjectGrowth* que é o principal responsável por encontrar os padrões [12]. Este algoritmo utiliza a busca em profundidade (*Depth-First Search* — DFS) em uma árvore formada por todos os subconjuntos das trajetórias no banco de dados. Cada nó do grafo, é formado por um conjunto de objetos (*objectset*) e as instâncias de tempo onde eles apareceram no mesmo cluster (*timeset*). Para acelerar a busca são utilizados dois algoritmos de poda chamados de *backward* e *apriori*. A poda *apriori* utiliza o fato que seu um conjunto de objetos cresce o tamanho do *timeset* deles só pode diminuir ou continuar igual, logo se um conjunto de objetos não forem um enxame fechado os seus superconjuntos também não serão. Já a poda *backward* verifica se dois nós visitados tem o mesmo *timeset* e se um deles (O) é superconjunto do outro (O'), se isto acontece e O é enxame, qualquer nó que seja um superconjunto de O' e O ao mesmo tempo não é um enxame. Uma limitação desta abordagem é não apresentar possibilidade de processamento *on-line*, uma vez que a construção do grafo necessita que todos os dados estejam presentes no banco [10].

Li et al.[21] apresentam uma aplicação para mineração de padrões e trajetórias, chamada *MoveMine*, que detecta os padrões enxame e um padrão denominado periódico. Para identificar padrões periódicos os autores utilizam um algoritmo apresentado no mesmo trabalho chamado *Periodica*. Este algoritmo identifica pontos de referência, que são os pontos onde as entidades mais se movimentam, e então minera os padrões periódicos usando uma matriz de probabilidade do objeto o_i estar em um dado ponto de referência num momento t_j. Para minerar padrões enxame, a aplicação utiliza o algoritmo

ObjectGrowth, também proposto pelos mesmos autores. Além de minerar estes dois tipos de padrão, a aplicação minera *outliers* em trajetórias e faz o agrupamento utilizando o DBSCAN. Os resultados dos experimentos utilizando o algoritmo *ObjectGrowth* mostram que o mesmo é mais eficaz do que o algoritmo *VG-Growth* (apresentado em [35]) para resolver o problema de grupo, que tem definição parecida com a do padrão enxame.

Já em [36, 37, 10] é proposto um método chamado de *Incremental Get_Move* em que é montada um base de dados com todos os agrupamentos (C_{DB}) para cada instante de tempo no banco de dados (T_{DB}) utilizando algum algoritmo de clusterização (e.g. DBSCAN). A partir de C_{DB} é criada uma matriz de agrupamentos onde é possível determinar todos os objetos que participam de um determinado cluster (C_{ij}). Essa matriz é utilizada como entrada em um algoritmo de mineração de *Frequent Closed Items* que transforma as trajetórias de objetos em transações, onde cada cluster de objetos é um item. A partir da matriz de agrupamentos são montados *itemsets* representados Υ, os quais são subconjuntos de C_{DB}. Com base na operação de suporte de *itemset* $\sigma(\Upsilon)$, que é a quantidade de objetos em comum dentro de um mesmo *itemset*, é possível detectar padrões espaço-temporais baseados em cluster, incluindo grupo e enxame. O algoritmo apresentado é muito eficiente e apresenta melhora de desempenho considerável sobre os algoritmos desenvolvidos para detectar cada um dos padrões específicos como o *ObjectGrowth* [12] e o algoritmo de grupo proposto por Li et al.[9].

2.4 O Padrão Aglomeração

O primeiro trabalho a citar o padrão aglomeração (*gathering*) é o [38], onde os autores definem os conceitos de multidão (*crowd*) e, posteriormente, aglomeração. Este padrão se baseia em cinco atributos, sendo eles:

escala: quantas entidades participam da aglomeração;

densidade: os agrupamentos devem ser densos;

durabilidade: deve durar por um período de tempo contínuo;

estacionariedade: as propriedades geométricas devem ser relativamente estáveis (definido pela distância Hausdorff [39] entre os agrupamentos em diferentes *instante de tempos*);

compromisso: em qualquer período em que a aglomeração acontecer, deve haver um número de entidades que permanecem nela por um período de tempo (não necessariamente formado por instantes consecutivos).

Os quatro primeiros atributos são chave para a definição de multidão, pois o conceito de multidão é uma sequência de agrupamentos densos que dura pelo menos k *instante*

de tempos. A partir das multidões detectadas, é necessário verificar se existem pelo menos m participantes por um período k_p nestas multidões. Se esta condição for satisfeita uma aglomeração é reportada.

Em [38, 13] é apresentado um *framework* para a detecção do padrão aglomeração em bancos espaço-temporais. O processamento dos dados é dividido em três fases: agrupamento, detecção de multidões e detecção de aglomerações. Na fase de agrupamento, é aplicado o algoritmo de Douglas-Peucker [40] para simplificar as trajetórias e então os segmentos restantes são agrupados. Na fase de detecção de multidões, o algoritmo proposto adiciona ao final de cada multidão candidata um novo agrupamento. Se for possível adicionar este agrupamento, ou seja, se a distância entre os agrupamentos for menor ou igual a um limiar desejado, esta multidão continua a ser uma multidão candidata. Então, posteriormente é preciso verificar a multidão já tem a duração mínima de k instantes de tempo. Se tiver, ela é reportada pelo algoritmo como um padrão multidão e se não tiver ela é descartada. O padrão aglomeração não possui a propriedade de fecho descendente (*downward closure*) de padrões frequentes, isto é, dado uma sequência de multidões que não formam aglomeração não implica em superconjuntos destas multidões não formarem uma aglomeração [41]. Por não possuir a propriedade seriam necessários algoritmos com altos custos computacionais para detectar aglomerações fechadas. Para resolver este problema, os autores propuseram um algoritmo de Teste e Divisão (TED). Primeiro, o algoritmo TED verifica se a multidão inteira é uma aglomeração. Se for, a aglomeração é reportada. Caso contrário, o algoritmo identifica os agrupamentos inválidos e divide aquela multidão em conjuntos menores de agrupamentos que podem ou não ser multidões, a depender da sua duração, e continua o processamento até reportar todos os padrões identificados.

Em [42] é apresentada uma abordagem baseada em grafo para a detecção de aglomerações. Nesse trabalho é apresentada uma representação inédita para agrupamentos que se movem. A estrutura utiliza um grafo espaço-temporal que consiste de vértices (que representam os agrupamentos) e arestas que representam as relações espaciais e temporais entre os agrupamentos. O algoritmo apresentado no artigo é um mapeamento do algoritmo de Bron-Kerbosch (BK) [43] para descobrir cliques em grafos não-direcionados aplicado à descoberta de aglomerações. Para tanto, os autores definem que o peso em cada aresta do grafo é a distância Hausdorff entre os agrupamentos e que uma aresta ligando dois vértices (v_1, v_2) só existe se a diferença de tempo entre v_1 e v_2f or menor que um certo limite μ. A partir dessa definição do grafo, é proposto o algoritmo GR (*Gathering Retrieving*) que busca por cliques máximos utilizando o algoritmo BK. Posteriormente, são aplicadas regras de poda para reduzir o número de candidatos da verificação final com base na similaridade dos cliques (quantos vértices em comum cada par de cliques tem). Por fim, é feita uma caminhada nos cliques para detectar se os mesmos são ou não aglomerações. Os experimentos realizados no trabalho demonstram ganhos em desempenho consideráveis

(fator 10 e 10^2, dependendo do parâmetro) sobre o algoritmo apresentado em [38], apesar do algoritmo de [38] reportar mais padrões em alguns casos (5 a mais entre 800).

3 ALGORITMOS DE DETECÇÃO DO PADRÃO FLOCO

Os trabalhos que focaram esforços na descoberta de padrões floco em dados espaço-temporais se preocuparam em definir algoritmos eficientes para melhorar os limites de complexidade de seus concorrentes, como pode-se verificar em [2, 8, 18, 34]. Os primeiros trabalhos a propor soluções para este problema foram [44, 34], ainda usando uma definição simplificada de floco onde o padrão é formado por apenas o encontro de entidades em um instante de tempo. Mais tarde, em [45], foi proposta uma definição do padrão mais condizente com situações reais, em que para se caracterizar um floco é preciso que as entidades sigam juntas na mesma direção durante vários instantes de tempo. Esta definição é a mais utilizada desde então. As soluções propostas para a descoberta de flocos seguindo esta definição mais geral variam as técnicas usadas, sendo que entre elas está o mapeamento das trajetórias para um espaço altamente dimensional, alternativas puramente espaço-temporais, mapeamento das trajetórias para transações, entre outras [2, 8, 46].

Este capítulo apresenta os principais algoritmos para a detecção de flocos. A Seção 3.1 introduz algumas das técnicas básicas utilizadas pelos diferentes algoritmos apresentados. Tais técnicas incluem métodos de redução de dimensionalidade, métodos de geometria computacional e estruturas de dados que são úteis para a solução de problemas espaço-temporais. Em seguida, a Seção 3.2 apresenta algoritmos que utilizam mapeamento para espaços altamente dimensionais e técnicas de redução de dimensionalidade, a Seção 3.3 descreve algoritmos que utilizam mapeamento para transações para a detecção de flocos, e, por fim, a Seção 3.4 apresenta os algoritmos que baseiam-se em estratégias para a redução do espaço de busca.

3.1 Técnicas e Estruturas para Algoritmos Espaço-temporais

Várias técnicas e métodos computacionais são utilizados para a construção de algoritmos para a detecção de padrões de movimentação. Algumas técnicas, por exemplo, de computação geométrica podem ser utilizadas para fazer busca de entidades relacionadas de forma mais eficiente. Nas subseções a seguir serão introduzidos: métodos de redução de dimensionalidade utilizados por alguns dos algoritmos encontrados na literatura; uma técnica de computação geométrica e uma estrutura teste de pertinência, que são utilizadas neste trabalho para a proposta de um novo algoritmo de detecção de padrões floco.

3.1.1 Redução de dimensionalidade

A noção de "maldição da dimensionalidade" diz respeito aos problemas inerentes ao rápido crescimento do volume de dados com o aumento do número de dimensões do

espaço e consequente "espalhamento" dos dados nesse espaço. É um problema frequente em recuperação de dados por conteúdo, como, por exemplo, em recuperação de imagens por conteúdo descritas por meio de vetores numéricos de alta dimensionalidade, e também recorrente em algumas aplicações envolvendo dados espaço-temporais [17]. Existem várias técnicas que abordam o problema de redução de dimensionalidade, por exemplo, PCA (*Principal Component Analysis*) [47], TWD (transformada *wavelet* discreta), TCD (transformada cosseno discreta) e PR (projeção randômica) [48].

Na PR, por exemplo, o dado original d-dimensional é projetado para um subespaço k-dimensional ($k \ll d$) através da origem, utilizando uma matriz randômica R de dimensões $k \times d$, na qual as colunas têm tamanho unitário. Usando notação de matriz, onde $X_{d\times N}$ é o conjunto original de N observações N-dimensionais, $X^{RP}_{k\times N} = R_{k\times d}X_{d\times N}$ é a projeção dos dados em um subespaço k-dimensional. A ideia principal do mapeamento randômico origina do lema de Johnson-Lindenstrauss [49]: se um ponto em um espaço vetorial é projetado em um subespaço randômico selecionado de uma dimensão suficientemente alta, então as distâncias entre os pontos são preservadas aproximadamente [48]. A projeção randômica tem uma complexidade computacional relativamente baixa. Formar uma matriz randômica R e projetar os dados da matriz $X_{d\times N}$ nas k dimensões tem complexidade de ordem $O(dkN)$, e, se a matriz X é esparsa com aproximadamente c entradas que não são zero por coluna, então a complexidade é de $O(ckN)$ [50].

3.1.2 Varredura de plano

A técnica de varredura de plano foi primeiro utilizada em [51] para detectar interseção de segmentos de linhas no plano. Desde então, a técnica se mostrou extremamente valiosa para a redução de complexidade computacional em um grande número de problemas na área de computação geométrica. As aplicações desta técnica variam entre procura de pontos próximos, passando por formação de fecho convexo [52] até procura por árvore de extensão mínima [53]. É importante ressaltar que, até onde sabemos, esta técnica ainda não foi utiliza em algoritmos de descoberta de padrões espaço-temporais.

A ideia básica de todo algoritmo que utiliza a técnica de varredura de plano é passar uma reta (geralmente vertical) através de todo o plano, parando em alguns pontos. Então, faz-se operações geométricas somente com os objetos que intersectam a linha ou que estão próximos uns aos outros em relação à linha, formando uma espécie de "caixa". Desta forma, a quantidade de objetos a ser considerada em cada etapa do algoritmo é reduzida a essa região. A solução do problema é geralmente alcançada quando todos os pontos do conjunto de entrada foram processados [54] .

Para ilustrar o funcionamento da técnica, considere o problema onde deseja-se encontrar o par de pontos mais próximos dentro de um conjunto de pontos. Neste caso, uma abordagem ingênua teria complexidade $O(n^2)$, uma vez que seria necessário fazer uma

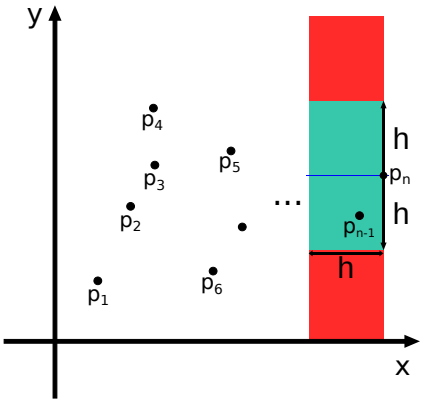


Figura 6 – Varredura de plano para descoberta de par de pontos mais próximos.

comparação de um ponto com todos os outros pontos do conjunto, para todos os pontos. Porém, com a utilização da técnica de varredura de plano esta complexidade é reduzida a $O(n \log n)$, utilizando-se, por exemplo, o algoritmo apresentado [55]. A Figura 6 ilustra a utilização deste algoritmo para encontrar o par de pontos mais próximos, sendo que, neste caso, é utilizada uma caixa em vez de simplesmente uma reta e a varredura ocorre no sentido dos pontos mais longe da origem até os mais próximos. No início do algoritmo processa-se $n-1$ dos n pontos do conjunto total $\{p_1, p_2, \ldots, p_n\}$, calculando a distância entre o ponto atual e o próximo, guardando sempre a menor distância e os dois pontos consecutivos, sendo esta distância mínima notada por h (mesmo h da Figura 6). Então, processa-se o último ponto p_n na tentativa de se encontrar algum ponto com distância menor que h em relação a p_n utilizando a faixa de tamanho h. Para conseguir achar estes pontos mais próximos deve-se manter um conjunto de pontos os quais tem a coordenada x a uma distância h de p_n, como é possível ver no retângulo vermelho da Figura 6. A medida que os pontos dentro da faixa vermelha são processados, eles são adicionados a um conjunto (estrutura), sendo que esta estrutura é ordenada pelo eixo-y. Uma árvore balanceada poderia ser utilizada para guardar estes pontos que estão dentro da faixa em x (a qual tem tempo de inserção $O(\log n)$). Para verificar se existe algum ponto que está a uma distância menor que h do ponto atual, basta verificar somente nos elementos da árvore, além disso considera-se somente os pontos que estão na faixa azul (de $y_n - h$ até $y_n + h$) da Figura 6. Estes pontos da faixa azul, por sua vez, podem ser recuperados em $O(\log n)$. Portanto é possível afirmar que, para cada ponto do conjunto, a busca tem complexidade $O(\log n)$, sendo assim o algoritmo como um todo tem complexidade de $O(n \log n)$.

É possível observar através deste exemplo que a varredura de planos pode alterar de forma crítica o tempo de execução de um algoritmo. Pôde-se verificar a utilização desta técnica também em outros trabalhos para diminuir a complexidade teórica dos algoritmos. No contexto dessa proposta, considera-se, portanto, que pode ser eficiente

utilizar a varredura de plano de forma semelhante à forma empregada para encontrar pares de pontos mais próximos a fim de encontrar discos que possam formar o padrão floco.

3.1.3 Skip Quadtree

A estrutura da Skip-Quadtree (SQT) é uma estrutura dinâmica eficiente para armazenar e indexar dados altamente dimensionais. Esta estrutura é composta de uma série de Quadtrees Comprimidas ligadas por uma *skiplist* e tem como principal objetivo reduzir os custos computacionais da consulta na estrutura quadtree comprimida [56].

A estrutura Quadtree Comprimida [57], por sua vez, é uma simplificação da representação da Quadtree original. Uma vez que ao invés de cada nó ter pelo menos quatro filhos, sendo eles quadrantes ou pontos, cada nó pode ter menos nós. Isto faz com que a quadtree comprimida tenha custos de atualização (inserção e remoção) muito menores do que a Quadtree comum. Na Figura 7 é possível ver uma Quadtree e sua representação como uma Quadtree Comprimida (retirado de [58]). É possível também ver a representação em forma de ponteiros, onde um quadrante é representado por um quadrado e um ponto é representado por um círculo sólido e um quadrante vazio é representado por um círculo vazio. Os quatro filhos de cada retângulo estão ordenados da esquerda para direita de acordo com os quadrantes I, II, III, IV.

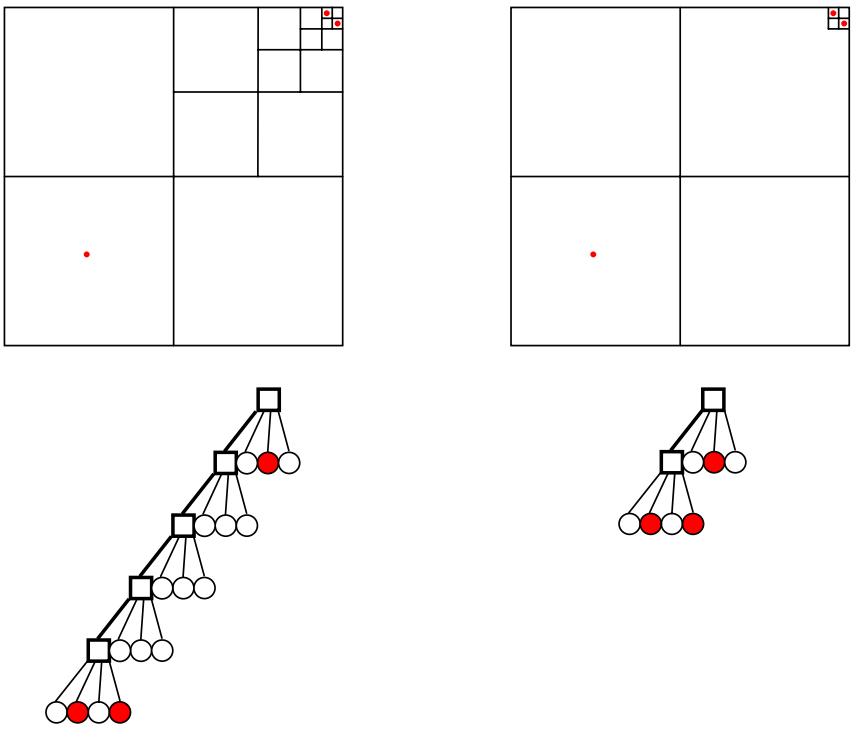


Figura 7 – Uma quadtree contendo três pontos (esq.) e a quadtree comprimida (dir.).

Já a Skip-Quadtree possui uma Quadtree Comprimida em cada nível folha da sua estrutura e liga essas estruturas a partir de uma *skiplist*, por isso o nome da estrutura.

Esta ideia de usar uma *skiplist* é para melhorar a complexidade de busca na estrutura, reduzindo de uma complexidade $O(n)$ na Quadtree Comprimida para $O(\log n)$ na Skip-Quadtree.

3.1.4 Filtro de Bloom

O filtro de Bloom é uma estrutura de dados probabilística eficiente em espaço, ou seja, requer espaço reduzido em memória. Foi desenvolvido na década de 1970 por Bloom[59] e é utilizado para testar se um determinado elemento pertence a um conjunto específico. Devido à "compressão" do conjunto de dados originais, a ocorrência de falsos positivos é possível enquanto a ocorrência de falso negativos não é [59].

O filtro de Bloom utiliza transformações de *hash* para computar um vetor (que é o filtro) o qual é uma representação do conjunto. O teste de membros é feito através da comparação dos resultados da operação de *hash* com os possíveis elementos no vetor. Na forma mais simples do algoritmo, o vetor contém N elementos, sendo que cada um deles é um bit. Uma determinada posição é acionada como '1' se alguma transformação *hash* mapeia para aquela posição do vetor. A Figura 8 mostra um filtro com $m = 4$ funções hash, produzindo respectivamente os valores $2, 5, 7$ e 4 para uma mesma chave K e mapeando para $N = 8$ bits [60].

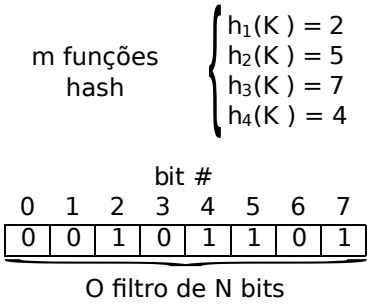
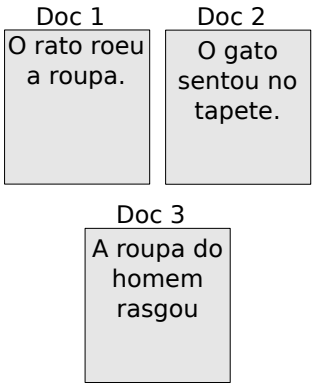


Figura 8 – Exemplo de filtro de Bloom com quatro funções hash e 8 bits.

3.1.5 Índice invertido

Apesar de não ter sido utilizado para busca do padrão floco, esta estrutura pode acelerar algumas fases da detecção de padrões espaço-temporais que envolvem a junção de conjuntos de entidades. A estrutura do índice invertido é bem conhecida na área de indexação de documentos e indexação de textos [61, 62, 63]. Originalmente a estrutura mantem uma lista de termos que aparecem no universo de documentos. Para cada termo é mantida uma lista dos identificadores dos documentos em que aquele termo apareceu. Quando se deseja buscar por documentos em que as palavras apareceram basta se encontrar a palavra na lista de termos e então recuperar a lista de documentos em que a palavra apareceu.



Termos	Documentos
O	{ 1, 2 }
rato	{ 1 }
roeu	{ 1 }
a	{ 1, 3 }
roupa	{ 1, 3 }
gato	{ 2 }
sentou	{ 2 }
no	{ 2 }
tapete	{ 2 }
do	{ 3 }
homem	{ 3 }
rasgou	{ 3 }

Figura 9 – Indexação de três documento usando índice invertido.

A Figura 9 mostra um exemplo de índice invertido e a indexação dos documentos neste índice. Ao procurar-se todos os documentos em que a palavra "gato" apareceu utilizando o índice da Figura 9 descobre-se que a esta palavra somente apareceu no documento 2. Isto reduz o espaço de busca, uma vez que ao invés de procurar em três documentos, somente um é pesquisado. Índices invertidos são de grande importância na área de busca textual, entretanto estas estruturas podem ser utilizadas em diversos problemas envolvendo busca dentro de conjuntos. Por exemplo, nesta dissertação foi utilizado o índice invertido para buscar entidades que participam de discos.

3.2 Algoritmos utilizando mapeamento para espaços altamente dimensionais

Alguns trabalhos que propuseram soluções para a descoberta de padrão do tipo floco utilizaram um mapeamento de trajetórias para um espaço altamente dimensional, arguindo e provando teoricamente que estas modificações fariam a complexidade deste problema ser reduzida [45, 8]. Entretanto, devido ao mapeamento para o espaço altamente dimensional, os dados sofrem uma degeneração que faz com que as respostas sejam aproximações no que se diz respeito ao diâmetro do floco. Isto acontece em todos os trabalhos que utilizam mapeamento.

O primeiro trabalho a explorar o mapeamento do problema de floco para espaços de alta dimensionalidade foi [8]. Na busca por uma solução que não precisasse manter os candidatos a floco entre as diferentes instâncias de tempo, os autores exploraram a transformação das trajetórias para pontos em um espaço altamente dimensional. O processo básico descrito no artigo envolve a criação de uma Skip Quadtree para cada tempo inicial possível do padrão de tamanho δ. Os trabalhos [1, 8, 45, 15, 17] utilizam a mesma técnica de redução na qual uma trajetória representada por uma linha poligonal, bi-dimensional contendo d vértices é mapeada a um único ponto em um espaço com $2d$ dimensões. Este tipo de abordagem gera uma dependência exponencial na duração do floco δ (para prova,

ver Lema 9 em [8]) [17].

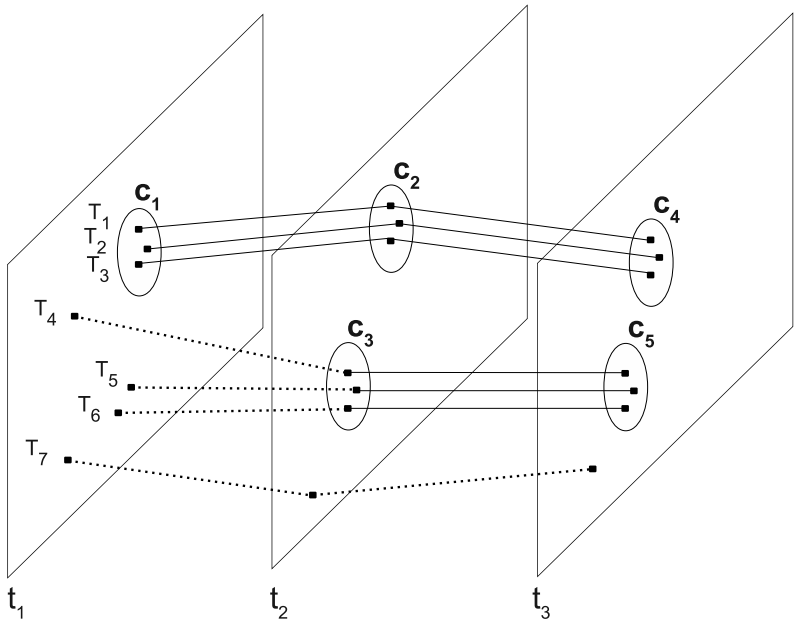
Em [17] são propostas consultas espaço-temporais para descoberta de padrões, entre eles o padrão floco. Este trabalho utiliza a mesma ideia empregada em [15], no qual as trajetórias são mapeadas de linhas poligonais para um espaço $2d$-dimensional, em que d é o número de vértices da polilinha. Como a dimensionalidade do espaço resultante é alta, é utilizada a projeção randômica para reduzir a sua dimensionalidade como uma etapa de pré-processamento. Nesse trabalho [17], não há melhora perceptível (no tempo de execução) entre um método força-bruta e o método proposto no artigo, mas a questão foi explorar a capacidade de representação da técnica de projeção randômica além de reduzir a dependência exponencial no tamanho do floco.

3.3 Algoritmos utilizando mapeamento para transações

O mapeamento de dados espaço-temporais para transações, apesar de não parecer tão óbvio, foi utilizado em vários trabalhos [10, 36, 18]. Nestes trabalhos costuma-se definir todos os agrupamentos descobertos no conjunto de dados, identificá-los de maneira unívoca e depois fazer o mapeamento para transações. A motivação de se fazer tal mapeamento é a utilização de algoritmos de mineração de padrões frequentes (MPF), como o *Linear time Closed itemset Miner* (LCM) [64, 65].

O algoritmo *LCM_Flock*, apresentado por Romero[18], foi o primeiro a utilizar MPF para identificação de padrões espaço-temporais e o único a identificar flocos. Nesse trabalho, o processo de mapeamento envolve a identificação unívoca de cada um dos possíveis discos do banco de dados, que é aliado a uma tabela adicional que guarda informações de quais trajetórias visitaram quais discos em um intervalo de tempo específico. Esta representação permite obter uma versão transacional das trajetórias, que são submetidas ao algoritmo LCM para a detecção de flocos com duração máxima. A Figura 10 ilustra a ideia de mapeamento de trajetórias para transações utilizada para a identificação de flocos. A Figura 10a mostra um conjunto de dados contendo sete trajetórias $(T_1, \ldots, T_7)$, sendo que são detectados cinco discos $(C_1, \ldots, C_5)$ durante toda a duração do conjunto de dados. A partir dos discos detectados é criada a tabela representada na Figura 10b que, quando tratada como um banco de dados transacional, serve de entrada para o algoritmo LCM.

A estratégia utilizada em [18] necessita que todos os dados estejam registrados no banco de dados, uma vez que é feito um passo, chamado de pré-processamento pelo autor, no qual todos os clusters possíveis são extraídos do conjunto de dados, sendo assim uma abordagem *off-line* de itentificação do padrão. É válido notar também que nesta abordagem são reportados os flocos de duração máxima, e por isso a quantidade padrões flocos reportados é menor (por vezes bem menor) em relação ao problema de flocos com



(a) Representação comum das trajetórias.

TID	ID dos discos
T_1	$\langle C_1, C_2, C_4 \rangle$
T_2	$\langle C_1, C_2, C_4 \rangle$
T_3	$\langle C_1, C_2, C_4 \rangle$
T_4	$\langle C_3, C_5 \rangle$
T_5	$\langle C_3, C_5 \rangle$
T_6	$\langle C_3, C_5 \rangle$
T_7	$\emptyset$

(b) Representação das trajetórias como transações.

Figura 10 – Transformação de trajetórias em transações.

tamanho fixo, discutido na próxima seção, uma vez que um floco com duração máxima pode conter vários flocos com tamanho fixo. Outras abordagens recentes também apostaram em mapeamentos para transações [66, 67, 16]. Porém, esse mapeamento adiciona a complexidade do pré-processamento de dados e, principalmente, tratam-se de abordagens *off-line*, não funcionando com *streams* de dados.

3.4 Algoritmos utilizando a redução do espaço de busca

Um dos principais obstáculos na busca pelo padrão floco é que o centro de um disco que define o padrão em uma dada instância de tempo pode não pertencer a uma das trajetórias. Qualquer ponto no espaço pode ser um centro de um floco, portanto existe um número infinito de possibilidades a serem testadas. Devido a este fato, não se pode simplesmente iterar sobre algumas posições das trajetórias e verificar se são um centro de um floco ou não. Para reduzir o espaço de busca e, por consequência, diminuir a complexidade computacional deste problema, Vieira, Bakalov e Tsotras[2] provaram que o espaço de busca para o problema de floco é discreto e limitado. Utilizando estas propriedades, os mesmos autores apresentaram um algoritmo base, de complexidade polinomial, para a detecção de flocos de duração fixa de forma *on-line* e mais quatro heurísticas utilizando o algoritmo base. Esta seção apresenta a estratégia de redução do espaço de busca e, em seguida, os algoritmos propostos que a utilizam.

Considerando-se a nomenclatura utilizada em [2] e sumarizada na Tabela 1, esse

Símbolo	Significado
O_j	Objeto identificado por j
T_j	Trajetória do objeto O_j
$p_j^{t_i}$	Ponto em $\mathbb{R}^2$ na instância de tempo t_i da trajetória T_j
$d(p_a^{t_i}, p_b^{t_i})$	Distância euclidiana entre $p_a^{t_i}$ e $p_b^{t_i}$
$f_k^{t_i}$	Floco k no tempo t_i
$c_k^{t_i}$	Centro do floco f_k no tempo t_i

Tabela 1 – Tabela com símbolos utilizados nesta seção.

resultado é derivado do Teorema 1.

Teorema 1. *Se para uma determinada instância de tempo t_i existe um ponto no espaço denotado por $c_k^{t_i}$ tal que:*

$$\forall T_j \in f_k, d(p_j^{t_i}, c_k^{t_i}) \leqslant \epsilon/2$$

então existe outro ponto no espaço $c_k'^{t_i}$ tal que:

$$\forall T_j \in f_k, d(p_j^{t_i}, c_k'^{t_i}) \leqslant \epsilon/2$$

e existem pelo menos as trajetórias $T_a \in f_k$ e $T_b \in f_k$ tal que:

$$\forall T_j \in \{T_a, T_b\}, d(p_j^{t_i}, c_k'^{t_i}) = \epsilon/2$$

O Teorema 1 afirma que se existir um disco com centro $c_k^{t_i}$ e raio $\epsilon/2$ que cobre todas as trajetórias de um floco f_k em alguma instância de tempo t_i, então existe outro disco com centro $c_k'^{t_i}$ e raio $\epsilon/2$ que cobre todas as entidades do primeiro e, além disso, há dois pontos em comum entre eles. Para a prova do teorema, refira-se a [2]. O Teorema 1 é de grande importância, visto que reduz o espaço de busca para a procura por flocos limitando o número de posições a serem processadas. Para um banco com $|\mathcal{T}|$ trajetórias, existem $|\mathcal{T}|^2$ pares possíveis de combinação em uma instância de tempo. Para cada par existem dois discos de raio $\epsilon/2$ que têm estes dois pontos nas suas circunferências (Figura 11) [2]. O resultado fundamental é que é possível alcançar uma resposta exata para o problema processando-se apenas esses discos.

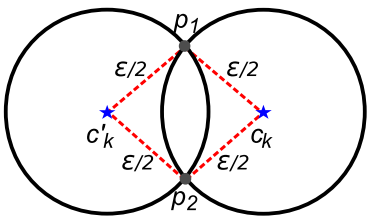


Figura 11 – Discos formados por p_1 e p_2.

3.4.1 O Algoritmo *Basic Flock Evaluation*

O algoritmo chamado *Basic Flock Evaluation* (BFE) é o mais simples dos algoritmos apresentados em [2]. Estes algoritmos têm algumas características em comum, entre elas o fato de se utilizarem de um índice baseado em uma grade para a procura de discos que possa formar novos flocos. Este índice é baseado em uma grade com células quadradas com ϵ de lado. Cada posição $p_{id}^{t_i}$ é inserida em uma célula deste índice dependendo das suas componentes de latitude e longitude, portanto o tamanho do índice é dependente da distribuição das trajetórias no espaço. A representação do índice pode ser vista na Figura 12.

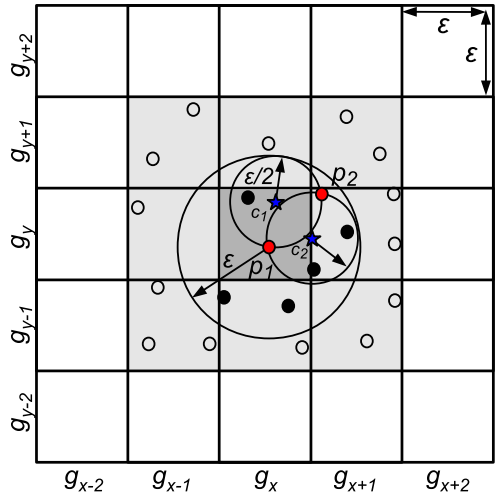


Figura 12 – Índice baseado em grade utilizado nos algoritmos propostos em [2].

Depois de construído o índice, o processo de encontrar discos pode ser representado através do Algoritmo 1. O algoritmo começa iterando sobre cada uma das células que não estão vazias e para cada ponto dentro destas células é executada uma consulta por abrangência nas nove células vizinhas, limitando assim o espaço de busca (ver Figura 12). Continuando o processo, o algoritmo processa cada um dos pontos da área de abrangência e caso o par formado pelo ponto sendo processado (p_r) e o ponto da área de abrangência (p_j) ainda não tenha sido processado, então são gerados discos a partir deste par de pontos (como visto na Figura 11). Por fim, caso os discos contenham mais de μ entidades eles são reportados como discos candidatos. Entretanto, este conjunto de discos ($\mathcal{C}$) retornado pelo Algoritmo 1 ainda precisar ser verificado a fim de se filtrar somente os discos que tem tamanho máximo, isto é, elimina-se os discos que estejam contidos em outros do conjunto $\mathcal{C}$.

Para manter somente estes discos que tem tamanho máximo o algoritmo utiliza as informações do centro do disco e do número total de trajetórias que os discos compartilham. Os discos são processados e testados somente com outros discos que estejam perto entre si, por exemplo, dado um disco de centro c_1 só se confere o disco de centro c_2 se $d(c_1, c_2) \leqslant \epsilon$. Caso a distância entre os centros dos discos seja maior que ϵ é possível

Algoritmo 1: Buscando discos no índice baseado em grade

Entrada: conjunto de pontos $\mathcal{T}[t_i]$ do instante de tempo t_i
Saída: Conjunto de discos de tamanho máximo $\mathcal{C}$

```
1  C ← ∅
2  Indice.Construir(T[t_i])
3  para cada célula não vazia g_{x,y} ∈ Indice faça
4      P_r ← g_{x,y}
5      P_s ← [g_{x-1,y-1} ... g_{x+1,y+1}] // Nove células vizinhas
6      se |P_s| ⩾ μ então
7          para cada p_r ∈ P_r faça
8              H ← BuscaAbrangencia(p_r, ε), |H| ⩾ μ, d(p_r, p_s) ⩽ ε ∈ P_s para
               cada p_j ∈ H faça
9                  se p_r, p_j não foram processados então
10                     computar discos c_1, c_2 definidos por p_r, p_j e diâmetro ε para
                       cada disco c_k ∈ c_1, c_2 faça
11                         c ← c_k ∩ H
12                         se |c| ⩾ μ então C ← C ∪ c;
13                     fim para cada
14                 fim se
15             fim para cada
16         fim para cada
17     fim se
18 fim para cada
```

afirmar com certeza que os discos não tem elementos em comum. Caso um par de discos não tenha os seus centros a mais de ϵ de distância então é preciso de fato verificar as trajetórias que participam daquele disco para afirmar se um é subconjunto do outro ou não.

A partir dos discos gerados na etapa anterior, este algoritmo faz a junção dos discos de tamanho máximo ao decorrer do tempo para encontrar os padrões floco. Os passos seguidos pelo BFE está apresentado no Algoritmo 2. Primeiro, ele adquire um conjunto de discos do índice baseado em grade, sendo estes discos já os discos de tamanho máximo. Então, para cada um disco destes discos do instante de tempo atual procura-se por discos de instantes de tempo anterior ($\mathcal{F}^{t_{i-1}}$) que tenham pelo menos μ entidades em comum com o disco sendo processado. Caso o mesmo disco (conjunto de trajetórias) aparecerem durante pelo menos δ instantes de tempo então um padrão floco é reportado (linha 11). Por fim, todos os discos do instante de tempo atual são inseridos em no conjunto de candidatos atual ($\mathcal{F}^{t_i}$) já que são vistos como potenciais flocos de tamanho 1.

3.4.2 Heurísticas aplicadas ao método BFE

O algoritmo de BFE mostrou-se eficiente devido à estratégia utilizada para redução do espaço de busca. Entretanto, o número de candidatos em uma instância de tempo pode

ser muito grande. Para tentar podar de forma mais eficiente este candidatos, algumas heurísticas foram adicionadas ao BFE [2].

O primeiro algoritmo a utilizar estas heurísticas é o ***Top-down Evaluation*** (**TDE**). Neste algoritmo, ao contrário da abordagem *bottom-up* utilizada no BFE, onde o processamento é feito de forma sequencial, é utilizada uma abordagem *top-down*, daí o nome do algoritmo. O processamento do algoritmo se dá através da comparação dos discos atuais com os da instância de tempo distante δ instantes de tempo. Isto é feito com base na premissa que a diferença entre os discos candidatos em duas instâncias consecutivas serão pequenas (gerando muito mais candidatos a serem mantidos), porém a diferença entre candidatos que estão a uma distância temporal de δ será mais significativa o que permite a poda de mais candidatos. Esta estratégia, de forma geral, mostrou-se a mais eficiente dentre as propostas por Vieira, Bakalov e Tsotras[2], pois o desempenho do algoritmo foi o melhor na grande maioria dos experimentos com dados reais apresentados pelos autores. Portanto, este algoritmo é considerado o estado da arte para a detecção de flocos de tamanho fixo. Uma observação a respeito desta heurística é que ela precisa reter em memória uma janela de dados de tamanho δ, ao passo que o BFE, apesar de simples, tem a vantagem de sempre processar um instante de tempo uma única vez. A mesma coisa ocorre com as demais heurísticas propostas.

A segunda heurística proposta por Vieira, Bakalov e Tsotras[2] é a ***Pipe Filter Evaluation*** (**PFE**), baseada no paradigma de filtragem e refinamento. A solução é jus-

Algoritmo 2: Algoritmo BFE

Entrada: parâmetros μ, ϵ e δ
Saída: Padrões floco

1 $\mathcal{F}^{t_0} \leftarrow \varnothing$
2 **para cada** *instante de tempo* t_i **faça**
3 $\quad \mathcal{F}^{t_i} \leftarrow \varnothing, \mathcal{C} \leftarrow IndiceGrade.Discos(\mathcal{T}[t_i])$
4 $\quad$ **para cada** $c \in \mathcal{C}$ **faça**
5 $\quad\quad$ **para cada** $f \in \mathcal{F}^{t_{i-1}}$ **faça**
6 $\quad\quad\quad$ **se** $|c \cap f| \geqslant \mu$ **então**
7 $\quad\quad\quad\quad u \leftarrow c \cap f$
8 $\quad\quad\quad\quad u.t_{inicio} \leftarrow f.t_{inicio}$
9 $\quad\quad\quad\quad u.t_{fim} \leftarrow t_i$
10 $\quad\quad\quad\quad$ **se** $(u.t_fim - u.t_inicio) = \delta$ **então**
11 $\quad\quad\quad\quad\quad$ reportar floco u de $u.t_{inicio}$ até $u.t_{fim}$
12 $\quad\quad\quad\quad$ **fim se**
13 $\quad\quad\quad\quad \mathcal{F}^{t_i} \leftarrow \mathcal{F}^{t_i} \cup u$
14 $\quad\quad\quad$ **fim se**
15 $\quad\quad$ **fim para cada**
16 $\quad\quad \mathcal{F}^{t_i} \leftarrow \mathcal{F}^{t_i} \cup c$
17 $\quad$ **fim para cada**
18 **fim para cada**

tamente efetuar um filtro nos dados no primeiro momento e aplicar o algoritmo BFE (refinamento) nessa nova entrada, beneficiando este algoritmo principalmente para o caso de ocorrer um bom número de podas no filtro, limitando assim o montante de dados processado no BFE. O filtro é feito aplicando um busca por abrangência para cada trajetória, mantendo apenas aquelas que aglomerem ao menos μ trajetórias dentro de uma área de raio ϵ. O nome tubo (*pipe*) da heurísca é por lembrar esse formato se visualizado no tempo, visto que são salvos como candidatos apenas as trajetórias que mantiverem esse comportamento pela quantidade mínima de δ instantes de tempo. Ou seja, se o total de trajetórias dentro desse tubo tiver ao menos μ trajetórias em cada instância então esse conjunto é salvo como possível candidato. Vale ressaltar que essa heurística também utiliza índice baseado em grade em sua estrutura.

Por fim, a heurística **CRE** (***Continuous Refinement Evaluation***), como o próprio nome sugere, é uma heurística que busca um refinamento contínuo do conjunto de trajetórias que podem participar de flocos. De forma similar ao processo de filtro do PFE, essa heurística localiza todos os possíveis discos da primeira instância de tempo. Entretanto, como parte do refinamento contínuo, apenas as trajetórias associadas a esses discos previamente selecinados é que serão processados na instância seguinte a fim de realizar a junção dos discos e verificar a continuidade do possível floco. Essa heurística demonstra-se eficiente em casos de alta seletividade de discos candidatos, ou seja, se são identificados relativamente poucos discos candidatos e contendo poucas trajetórias também.

4 MÉTODO PROPOSTO: O ALGORITMO PSI PARA DESCOBERTA *ON-LINE* DE FLOCOS

Como visto no Capítulo 3, existem diversos algoritmos para a descoberta do padrão floco, porém a sua grande maioria são algoritmos *off-line*, isto é, não suportam fluxos de dados e/ou resolvem variações do floco que não aquela estudada neste trabalho. Com o objetivo de preencher esta lacuna, este capítulo apresenta um novo algoritmo para detecção do padrão floco (duração fixa) de forma *on-line* chamado PSI (**P**lane Sweeping, Binary **S**ignatures and Inverted **I**ndex). Este método é a principal contribuição deste trabalho (resultados parciais foram publicados em [68]). A Seção 4.1 detalha as técnicas básicas utilizadas pelo PSI. Já a Seção 4.2 apresenta os algoritmos desenvolvidos no decorrer do projeto, baseados nas técnicas propostas e a Seção 4.3 descreve a aplicação de heurísticas propostas inicialmente para o BFE, gerando algoritmos adicionais baseados no PSI.

4.1 Técnicas básicas do algoritmo PSI

O PSI é um algoritmo baseado no BFE (método proposto em [2]) que utiliza técnicas e estruturas para acelerar a busca do padrão floco, sendo as principais: varredura de plano, assinaturas binárias, e índices invertidos. Por meio destas técnicas, o PSI é consideravelmente mais eficiente do que o BFE, como confirmado pelos experimentos descritos no próximo capítulo, tornando-se o estado da arte para a detecção *on-line* do padrão floco. A Subseção 4.1.1 mostra como o PSI remove a necessidade do uso de índices para procurar por discos candidatos (como no BFE), utilizando a técnica de varredura de plano. Na Subseção 4.1.2 é explicada a utilização das assinaturas para identificar os discos candidatos univocamente baseado nos participantes destes discos e posteriormente a utilização dessas assinaturas para podar duplicatas e subconjuntos de discos. Por fim, na Subseção 4.1.3 é mostrada a utilização do índice invertido para procurar por discos (parciais ou completos) do instante de tempo atual nas instâncias de tempo anteriores.

4.1.1 Descoberta de discos usando varredura de plano

Como mostrado na Seção 3.4, o algoritmo BFE e suas variações dependem de uma estrutura de índice baseado em grade para procurar por discos candidatos no espaço. Este índice, por sua vez, tem um custo computacional linear associado com sua criação. Esta varredura linear na entrada de dados por vezes é custosa uma vez que se pré-computa várias métricas, incluindo e não limitada a: (i) cálculos de distância entre trajetórias, (ii) contagem de trajetórias vizinhas, (iii) inserção das trajetórias em cada uma das células do índice. Tendo isto em vista, neste trabalho é proposta a utilização da varredura de plano

para eliminar a necessidade da criação de índices. No Algoritmo 3 é mostrado o processo da busca por discos.

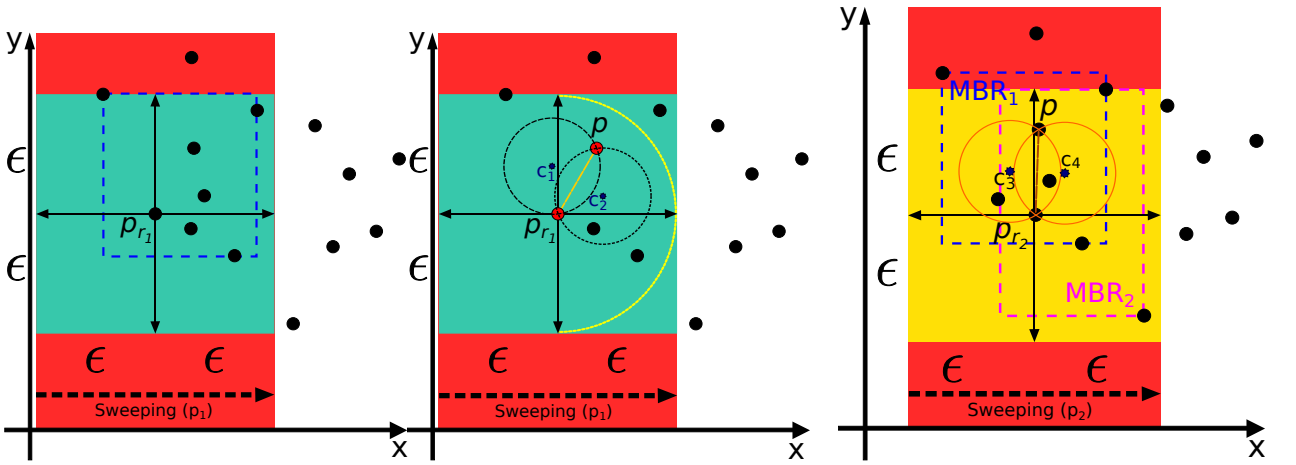
Algoritmo 3: Procurar discos candidatos usando a varredura de plano

Entrada: $\mathcal{T}[t_i]$: posições do instante de tempo t_i ordenados pelo valor do eixo x, ϵ: diâmetro do padrão, μ: tamanho mínimo do padrão
Saída: $\mathcal{C}$: discos candidatos do instante t_i, $\mathcal{B}$: caixas ativas no instante t_i

```
1  C ← ∅, B ← ∅
2  para cada p_r ∈ T[t_i] faça
3      P ← ∅ // elementos da caixa atual definidos por p_r
4      para cada p_s ∈ T[t_i] : | p_s.x − p_r.x | ⩽ ε faça // testar somente
       elementos dentro da faixa de 2ε no eixo-x
5          se |p_s.y − p_r.y| ⩽ ε então // verificar se p_s está na faixa 2ε
           em y
6              P ← P ∪ p_s // adiciona p_s para a caixa
7          fim se
8      fim para cada
9      para cada p ∈ P : p.x ⩾ p_r.x faça (// elementos na parte direita
10         se dist(p_r, p) ⩽ ε então // calcular distância do par
11             seja {c_1, c_2} discos definidos por {p_r, p} e raio ε/2
12             para cada c ∈ {c_1, c_2} faça
13                 se |c ∩ P| ⩾ μ então // verificar a quantidade de
                   entidades no disco
14                     C ← C ∪ c // adicionar c para discos candidatos
15                     B ← B ∪ box(p_r) // adicionar a caixa atual para as
                       caixas ativas
16                 fim se
17             fim para cada
18         fim se
19     fim para cada
20 fim para cada
21 retorna C, B
```

O Algoritmo 3 começa varrendo o plano (da esquerda para direita no eixo x) usando uma banda de tamanho 2ϵ centrada em um ponto do conjunto de dados (p_r) (faixa vermelha na Figura 13(a)). Depois o algoritmo seleciona todos os pontos dentro da banda que estão na faixa de distância $[p_r.y - \epsilon, p_r.y + \epsilon]$ (caixa verde na Figura 13(a)). Estes passos estão descritos nas linhas 2–8 do Algoritmo 3.

Depois de selecionar os pontos na caixa ($2\epsilon \times 2\epsilon$) definida por p_r, é necessário procurar por pares de pontos que possam gerar um novo disco candidato (ver Teorema 1). Estes discos vão ser definidos por pares de pontos entre p_r e qualquer outro ponto p dentro da parte direita da caixa onde a distância ($d(p_r, p) \leqslant \epsilon$) seja menor que ϵ (semi-círculo amarelo pontilhado na Figura 13(b)). Os pontos na metade esquerda da caixa já foram processados em passos anteriores e só são utilizados para calcular o número de entidades



(a) Varrendo o plano no eixo x. (b) Filtrar pontos no eixo y. (c) Procura por discos que são subconjuntos.

Figura 13 – Passos do PSI para encontrar discos em uma instância de tempo.

em discos (e.g., disco com centro c_1 na Figura 13(b)). Se um disco candidato contém pelo menos μ entidades, então o conjunto de entidades contido nele é reportado como um conjunto candidato e a caixa em que o disco pertence é reportada como ativa no instante de tempo atual. Toda caixa *ativa* é representada pelo seu MBR (*Minimum Bounding Rectangle*) que delimita todos os seus elementos (retângulos pontilhados na figura). Estes últimos passos são representados pelas linhas 9–19 do Algoritmo 3.

4.1.2 Utilização de assinaturas binárias para podar candidatos

Depois de ter detectado os discos (ou conjuntos) candidatos de um instante de tempo, é preciso verificar se os discos possuem subconjuntos entre eles. De modo similar ao algoritmo BFE, o interesse é encontrar somente os flocos. O algoritmo BFE utiliza somente as propriedades espaciais do discos para acelerar a detecção de conjuntos *maximais*, isto é, com o maior número de entidades. A proposta deste trabalho se difere ao passo que utiliza as propriedades espaciais das caixas (geradas pela varredura de plano) como passo de poda em nível macro e depois na fase de filtro fino utiliza-se das propriedades espaciais combinadas com assinaturas binárias geradas a partir dos identificadores dos participantes no disco. Estes passos garantem uma poda mais eficiente para evitar a execução de operações de interseção de conjuntos.

O processo de filtragem de conjuntos candidatos é mostrado no Algoritmo 4. O relacionamento espacial entre caixas é dado pelos seus MBRs. Cada caixa tem informação sobre os conjuntos candidatos que pertencem a ela. O passo de filtro verifica as caixas que possuem pelo menos um disco candidato. O Algoritmo 4 começa iterando as caixas ativas no instante de tempo atual, e verifica se há interseção entre o MBR da caixa atual e os MBRs das caixas próximas a ela. Caso haja interseção então é necessário certificar se realmente existem duplicatas ou subconjuntos entre estas caixas (Figura 13(c)).

Antes de executar a operação de interseção conjunto, o método proposto realiza

Algoritmo 4: Filtrar os discos candidatos finais

```
1  Algoritmo FiltrarCandidatos(B)
       Entrada: B: caixas ativas em t_i, ordenadas pela posição no eixo-x
       Saída: C: conjunto final de candidatos para instante t_i
2      C ← ∅
3      para j ← 0 to j ⩽ |B| faça
4          para k ← j + 1 to k ⩽ |B| faça
5              se IntersectaCom(B[j], B[k]) então
6                  para cada c ∈ B[j].disks faça
7                      C ← InserirDiscos (C, c)
8                  fim para cada
9              fim se
10             senão // Não há interseção
11                 break
12             fim se
13         fim para
14     fim para
15 Procedimento InserirDiscos(C, c)
       Entrada: C: conjunto de discos, c: novo disco
16     para cada d ∈ C faça
17         se c.sign ∧ d.sign = c.sign&&d(c.center, d.center) ⩽ ϵ então // c
           pode ser subconjunto d
18             se d ∩ c = c então // Retira a chance de falso-positivo
19                 retorna C // Não precisa inserir c
20             fim se
21         fim se
22         senão se c.sign ∧ d.sign = d.sign então // d pode ser
           subconjunto de c
23             se c ∩ d = d então // Retira a chance de falso-positivo
24                 C ← C\d // Retira d
25             fim se
26         fim se
27     fim para cada
28     retorna C ∪ c
```

uma segunda etapa filtro usando assinaturas binárias (Procedimento *InserirDiscos()* no Algoritmo 4). Ao passo que as entidades são inseridas nos discos, um conjunto de funções de espalhamento são utilizadas para gerar a assinatura dele. Uma assinatura inicialmente é representada por uma cadeia binária de zeros (por exemplo, 0000 0000 0000 0000). A assinatura resultante é computada executando as funções de espalhamento (*hash*) para cada identificador do disco candidato. Cada função mapeia um identificador de objeto para uma posição (ou *bucket*) na assinatura e então aquela posição é atualizada com o bit 1. As primitivas de execução de consultas de subconjuntos utilizando assinaturas binárias são descritas em [69]. Nesse trabalho o autor Goel e Gupta[69] usou um conjunto de fil-

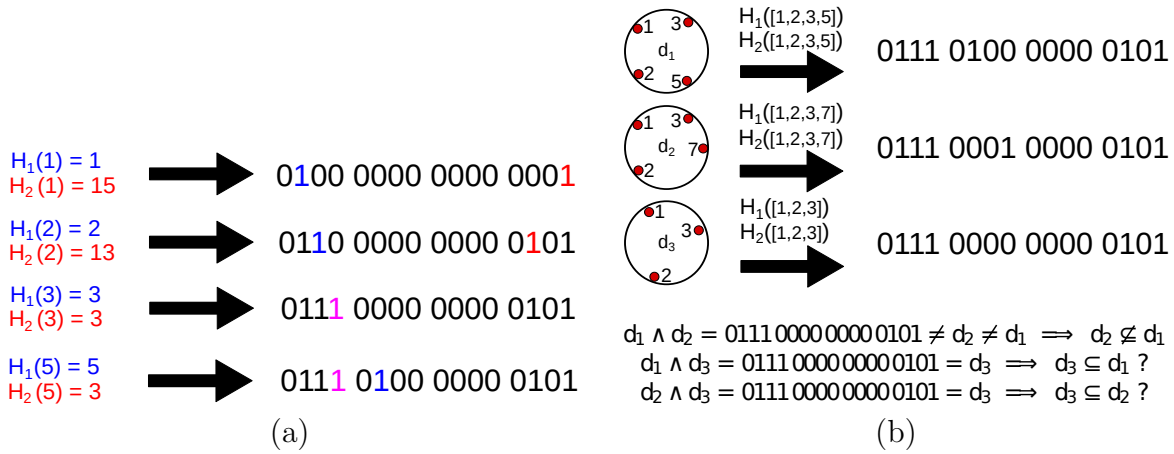


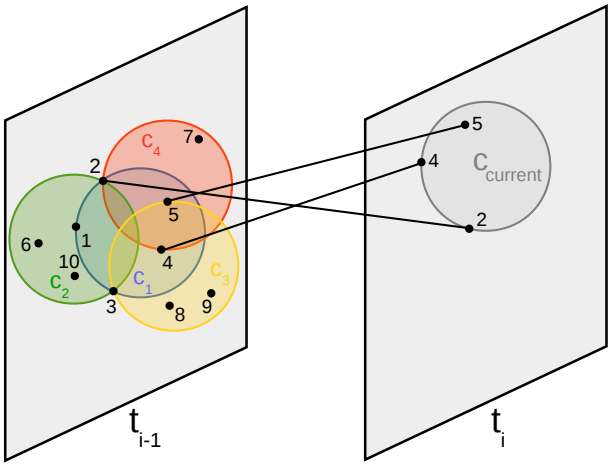
Figura 14 – (a) Processo de geração das assinaturas. (b) Verificação de subconjuntos via assinaturas.

tros de Bloom para representar os subconjuntos de um universo e então estes filtros, que basicamente são vetores binários de tamanho fixo, são utilizados para verificar subconjuntos entre os conjuntos. Sabe-se que os filtros de Bloom podem gerar falsos-positivos [60], porém não geram falso-negativos. Por esta razão, após testar se um disco é subconjunto do outro via assinaturas binárias é necessário fazer de fato a operação de interseção de conjuntos para eliminar eventuais falsos-positivos.

A Figura 14 ilustra o processo de filtro usando assinaturas binárias. Suponha um disco que contém os objetos identificados por $\{1, 2, 3\}$ e outro disco com os objetos $\{1, 2, 3, 5\}$, então suas assinaturas binárias serão idênticas, respectivamente, à terceira linha e quarta linha da Figura 14(a). Esta imagem foi inspirada pela execução das funções *hash* SpookyHash[1](H_1) e MurMurHash[2], que são funções de espalhamento rápidas com boa não-linearidade (medido pelo critério avalanche[3]). Um fato importante a se notar sobre as assinaturas binárias é que se o tamanho do universo for muito maior do que as assinaturas podem representar (tamanho em bits) podem haver colisões entre as funções de espalhamento (*bits* representados em violeta na Figura 14(a)).

Suponha, agora, que deseja-se evitar a execução de uma operação de interseção de conjuntos para determinar se um disco é ou não super/subconjunto de outro. Para tanto, aplica-se uma operação **AND** bit a bit (*bitwise*) entre as duas assinaturas dos discos. Se o resultado da operação é igual a um dos operandos, então aquele operando pode ser um subconjunto do outro. Caso contrário é possível afirmar, com certeza, que os dois discos não são subconjuntos um do outro. A Figura 14(b) exemplifica esse processo. O discos $d_1 = \{1, 2, 3, 5\}$, $d_2 = \{1, 2, 3, 7\}$ e $d_3 = \{1, 2, 3\}$ têm suas assinaturas indicadas à direita. O resultado da operação $d_1 \wedge d2$ é 0111 0000 0000 0101, que coincide com nenhuma das

[1] <burtleburtle.net/bob/hash/spooky.html>
[2] Murmurhash 2.0: <sites.google.com/site/murmurhash>
[3] <floodyberry.com/noncryptohashzoo>



Lista invertida dos discos em t_{i-1}	
OID	Discos
1	{c1, c2}
2	{c1, c2, c4}
3	{c1, c2, c3}
4	{c1, c3, c4}
5	{c1, c3, c4}
...	

Disco atual: {2,4,5}
Lista2 ⋂ Lista4 ⋂Lista5 = {c1,c4}

Figura 15 – Junção de discos em instantes de tempo consecutivos usando o índice invertido ($\mu = 3$).

assinaturas dos conjuntos. Portanto, pode-se afirmar que nenhum dos conjuntos é subconjunto do outro. Por outro lado, $d_1 \wedge d_3 = d_3$, portanto é possível que d_3 seja subconjunto de d_1. De fato, isso é verdade, mas só pode ser verificado posteriormente realizando-se uma operação de interseção de conjuntos. De maneira semelhante, como $d_2 \wedge d_3 = d_3$, a interseção entre esses conjuntos é necessária. Entretanto, o uso de assinaturas binárias elimina a necessidade de interseção para todos os verdadeiro-negativos detectados, número que tende a ser grande, dependendo das funções de espalhamento escolhidas.

4.1.3 Junção de discos eficiente usando índice invertido

Depois que o conjunto de discos finais de uma instância de tempo for detectado é necessário fazer a junção dos conjuntos candidatos com os conjuntos das instâncias de tempo anteriores, assim como no BFE. Uma maneira simples de realizar este processo é iterar todos os discos da instância atual e comparar com todos da anterior verificando a condição de junção dos conjuntos, isto é, se os conjuntos têm pelo menos μ entidade em comum. Porém, este processo é computacionalmente custoso, uma vez que é preciso executar operações de interseção de conjuntos para todos os candidatos (de modo combinatório) com todos já mantidos (das instâncias de tempo anteriores). Neste trabalho é proposta a utilização de índices invertidos para acelerar o passo de junção de conjuntos candidatos de instantes de tempo diferentes.

A estrutura de dados índice invertido (ou lista invertida) é bem conhecida para indexação de documentos e busca eficiente por termos neste índice [63]. Geralmente, um índice invertido tem uma lista de palavras-chave, que são termos comuns que aparecem em todos documentos do universo. Para usá-lo na junção de conjuntos de instantes de tempo subsequentes foi utilizado um índice invertido onde os "documentos" são os conjuntos do instante de tempo anterior (t_{i-1}) e os "itens" (ou palavras-chave) são as entidades

	Técnicas Aplicadas			
Algoritmo	**Varredura de plano**	**Ass. Binária**	**Índice inv.**	**Heurística**
BFI			✓	
PSW	✓			
PSB	✓	✓		
PSI	✓	✓	✓	
RPFE	✓			PFE
CRE_PS	✓			CRE
TDE_PS	✓			TDE

Tabela 2 – Algoritmos desenvolvidos neste trabalho.

participantes daquele conjunto. Para busca no índice é utilizada o seguinte parâmetro de consulta: retorne todos os documentos (conjuntos) que tenham pelo menos μ itens em comum com um conjunto dado como elemento de consulta (ver Figura 15).

Quando um disco do instante de tempo atual (t_i) está sendo processado, utiliza-se os identificadores dos objetos/entidades participantes (OIDs) como o elemento de consulta. No exemplo da Figura 15 os discos c_1 e c_4 compõem o resultado da consulta.

4.2 Algoritmos propostos

É possível observar que cada uma das técnicas apresentadas na seção anterior é aplicada em uma etapa diferente na descoberta do padrão Floco. Como já foi citado também, este trabalho buscou aprimorar os algoritmos do estado-da-arte (ou seja, algoritmo BFE e suas heurísticas) através da aplicação dessas técnicas. Para realizar uma melhor investigação do ganho real no desempenho de cada uma das técnicas, foram desenvolvidos vários algoritmos, aplicando-se de forma gradativa as técnicas no algoritmo BFE. Além destes métodos baseados no algoritmo básico, também foram desenvolvidos métodos baseados nas heurísticas propostas sobre o BFE (vide Seção 3.4.2), que são melhorias no processamento com o objetivo de podar discos candidatos na fase de junção de instantes de tempo. Esta poda auxilia a reduzir o número de operações de interseção de conjunto que devem ser realizadas durante esta fase de junção. Além desses algoritmos, foi desenvolvida uma versão *on-line* do algoritmo LCM_Flock (Seção 3.3), que utiliza uma estratégia de mapeamento de trajetórias para transações e, segundo os seus autores originais, apresenta desempenho superior ao BFE em várias situações, embora sua proposta inicial seja *off-line*.

A Tabela 2 resume os métodos propostos no trabalho. O algoritmo BFI é uma implementação do algoritmo BFE utilizando índice invertido. O algoritmo PSW utiliza varredura de plano, o PSB usa adicionalmente assinaturas binárias e o algoritmo PSI é o método completo que implementa as três técnicas propostas no trabalho. Já os algoritmos

Algoritmo 5: Algoritmo PSI

Entrada: parâmetros μ, ϵ e δ,
$\mathcal{T}$: Posições de todos instantes de tempo
Saída: Padrões floco

```
1  F^{t0} ← ∅, B ← ∅
2  para cada instante de tempo t_i faça
3      F^{ti} ← ∅
4      C ← ∅
5      B ← VarreduraDePlano(T[t_i]).caixas()
6      C ← FiltrarCandidatos(B) ; // Discos do instante atual
7      para cada c ∈ C faça
8          para cada f ∈ IndiceInvertido(F^{t_{i-1}}).discos(c, μ) faça
9              se |c ∩ f| ⩾ μ então
10                 u ← c ∩ f
11                 u.t_inicio ← f.inicio
12                 u.t_fim ← t_i se u.t_fim − u.t_inicio = δ então
13                     reporta floco u de u.t_inicio a u.t_fim
14                 fim se
15                 F_i^t ← F_i^t ∪ u
16             fim se
17         fim para cada
18         F_i^t ← F_i^t ∪ c
19     fim para cada
20 fim para cada
```

RPFE, CRE_PS e TDE_PS são variações das heurísticas PFE, CRE e TDE utilizando varredura de plano em vez de índice em grade. As subseções a seguir detalham cada um desses algoritmos.

4.2.1 Algoritmo PSI: método completo

O algoritmo **PSI** (***P**lane Sweeping with Binary **S**ignatures and **I**nverted indexes*) é a principal contribuição deste trabalho, pois combina todas as técnicas propostas e obteve o melhor resultado nos testes realizados. Este algoritmo aplica melhorias em todos os passos do algoritmo BFE, desde a procura por discos candidatos, passando pela detecção de subconjuntos e junção de discos em diferentes instantes de tempo. O Algoritmo 5 mostra todo o processo executado pelo algoritmo PSI para encontrar padrões floco de duração fixa dentro de um conjunto de trajetórias. Primeiro o algoritmo utiliza a varredura de plano para encontrar o conjunto de caixas, utilizando o Algoritmo 3, como descrito na Subseção 4.1.1 (linha 5). O próximo passo para se chegar ao conjunto final de discos candidatos é a eliminação dos discos que não têm tamanho máximo, em termos de número de entidades (linha 6). Ou seja, o conjunto de entidades no disco é um subconjunto do conjunto de entidades de outro disco do mesmo instante de tempo. Em vez de realizar diretamente operações de interseção de conjuntos para verificar a ocorrência de subcon-

juntos, que é uma operação custosa, o algoritmo efetua essa tarefa utilizando os passos a seguir (este processo esta detalhado no Algoritmo 4).

1. Verifica-se se há interseção entre a caixa atual e outras caixas. Se não houver, todos os discos da caixa atual são inseridos no conjunto de discos candidatos, pois não possuem subconjuntos neste instante de tempo, já que estão a pelo menos ϵ de distância dos demais discos.

2. Se houve interseção entre duas caixas, pode haver subconjuntos dentro delas. Então, verifica-se a distância entre os centros dos discos das caixas. Se esta distância for maior que ϵ, estes discos são disjuntos e, portanto, inseridos no conjunto de candidatos, pois suas entidades garantidamente estão distantes uma das outras pelo menos ϵ.

3. No último caso, se os centros dos discos não estiverem a mais de ϵ de distância, utiliza-se as assinaturas binárias dos discos (como explicado na Subseção 4.1.2) para verificar se eles possuem entidades distintas entre eles. Caso a comparação das assinaturas binárias não descarte a possibilidade de subconjunto, faz-se de fato a operação de interseção de conjuntos para decidir se um disco é ou não subconjunto do outro. Os discos resultantes são inseridos no conjunto de candidatos.

A partir do conjunto de discos candidatos, o algoritmo tenta fazer a junção dos flocos parciais que estão em instantes de tempo anteriores com cada disco do instante de tempo atual. Para reduzir a quantidade de flocos parciais a serem processados o algoritmo PSI utiliza a estrutura de índice invertido para procurar os discos que tenham pelo menos μ entidades em comum com o disco sendo processado (processo detalhado já discutido na Subseção 4.1.3). Para cada disco retornado da consulta ao índice invertido o algoritmo realiza a operação de interseção de conjuntos para verificar quais das entidades participantes do floco parcial (dos instantes de tempo anterior) continuaram a formar grupos no instante de tempo atual. Caso o tamanho da interseção seja de pelo menos μ, então este novo conjunto continua no conjunto de flocos parciais e tem suas informações de duração atualizadas (linhas de 9 até 15). Caso este floco parcial tenha alcançado uma duração de δ, então este floco é reportado como uma resposta (linha 13).

4.2.2 Variações do método completo

As técnicas desenvolvidas no método completo PSI podem ser aplicadas separadamente sobre o algoritmo BFE, gerando variações. A primeira variação proposta é o algoritmo **BFI** (***B**asic **F**lock Evaluation with **I**nverted index*), que é semelhante ao BFE, diferindo somente a forma como é feita a junção de discos em instantes de tempo diferentes. Neste algoritmo se utiliza a estrutura de índice invertido como foi apresentado

na Seção 4.1.3 para auxiliar a fazer a junção dos discos. Desta forma, pode-se avaliar o desempenho da estrutura do índice invertido de forma isolada.

A segunda variação é o algoritmo **PSW** (***P****lane* ***S****weeping Ra****W***), no qual utiliza-se a técnica de varredura de plano para procurar discos no espaço, ou seja, ao invés de utilizar o índice baseado em grade e consulta-lo por discos se utiliza a técnica de varredura que gera discos e caixas para facilitar a poda de duplicatas (ver Algoritmo 3). Este algoritmo só utiliza a varredura de plano, o que nos possibilita analisar o impacto dessa técnica de forma isolada.

Por fim, a terceira variação desenvolvida foi o algoritmo **PSB** (***P****lane* ***S****weeping with* ***B****inary Signatures*), que utiliza a combinação de duas técnicas: a varredura de plano e assinatura binária. Primeiro se utiliza a varredura de plano para se encontrar os discos. Em seguida, além de utilizar as propriedades espaciais das caixas, utiliza também assinaturas binárias para podar discos que não são subconjuntos uns dos outros no mesmo instante de tempo.

4.2.3 Algoritmo LCM_Flock *on-line*

O algoritmo LCM_Flock *on-line* desenvolvido neste trabalho é uma adaptação do método LCM_Flock (vide Seção 3.3). O algoritmo original primeiro busca por discos candidatos em *todos* os instantes de tempo e depois converte estes discos encontrados em uma versão transacional definida pelos identificadores das trajetórias. Em seguida, ele se utiliza do algoritmo de mineração de conjuntos de itens frequentes (*frequent itemset mining*) chamado LCM [65] para verificar os discos que participaram em pelo menos μ transações. Para que o algoritmo funcione é preciso carregar o conjunto de entrada inteiro para fazer o mapeamento de trajetórias em transações, o que impossibilita a utilização de fluxos de dados (*streams*) como entrada para o algoritmo.

Como parte deste projeto foi desenvolvida uma nova versão do algoritmo LCM_Flock capaz de processar fluxos de dados, ou seja, uma versão *on-line*. Para tanto, modificou-se o algoritmo original para que o mesmo tratasse uma janela de tempo de tamanho δ como um conjunto de dados. Desta forma, o algoritmo reporta respostas assim que o número de instantes disponíveis para serem processados forem pelo menos δ. Como consequência o algoritmo consegue reportar padrões mesmo em dados em forma de fluxo.

4.3 Heurísticas aplicadas ao algoritmo PSI

Na proposta original do algoritmo BFE foram incluídas algumas heurísticas para o aprimoramento do desempenho do algoritmo. Nesta seção são descritas três variações das heurísticas, ajustadas para uso com o algoritmo PSI: RPFE, CRE_PS e TDE_PS. Havia ainda uma possibilidade adicional de aplicar-se a heurística CFE. Porém, devido

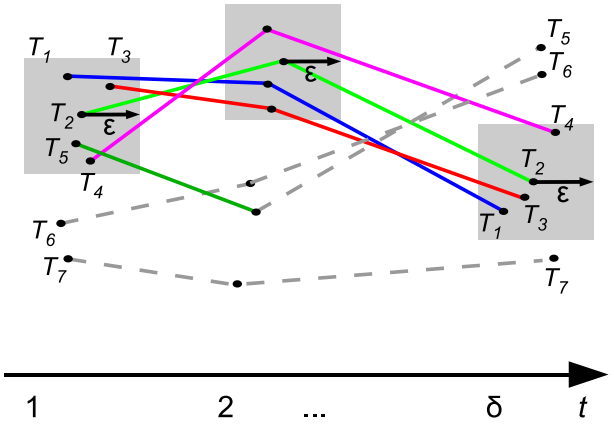


Figura 16 – Tubo retangular gerado pela heurística do método RPFE.

a motivos de performance e adaptação não foi feita uma implementação da CFE sobre o PSI neste trabalho, já que o algoritmo foi o que apresentou piores resultados quando aplicado sobre o BFE. Além disso, essa heurística usa o algoritmo DBSCAN, o qual teria que ser modificado para utilizar a varredura em disco.

O **RPFE** (***R**ectangular **P**ipe **F**iltering **E**valuation*) é o primeiro algoritmo baseado nas heurísticas citadas. Ele é baseado na heurística PFE, que utiliza clusters centrados nas trajetórias para limitar o número de posições a serem visitadas nas instâncias de tempo da janela de tamanho δ. Na adaptação feita no presente trabalho ao invés de utilizar clusters são utilizadas as caixas provenientes da varredura de plano. Somente as trajetórias presentes em caixas geradas no primeiro instante de tempo da janela que possuem pelo menos μ entidades são mantidas para a análise da janela inteira (ver Figura 16). Desta forma, ao invés de se utilizar um cluster de diâmetro 2ϵ utiliza-se as caixas retangulares com lado de *até* 2ϵ, uma vez que as caixas são limitadas pelos seus MBR's (ver Subseção 4.1.1).

A heurística **CRE_PS** (***C**ontinuous **R**efinement **E**valuation with **P**lane **S**weeping*) é baseado na heurística CRE. No algoritmo original é utilizado o índice baseado em grade para detectar os discos candidatos em instâncias de tempo consecutivas, desta forma só se analisa as trajetórias do tempo t_i que participaram de algum disco candidato em t_{i-1}. No algoritmo proposto neste trabalho, CRE_PS, é utilizada a varredura de plano para encontrar os discos candidatos, enquanto as demais etapas se mantêm.

Por fim, a heurística **TDE_PS** (***T**op-**D**own **E**valuation with **P**lane **S**weeping*), como o nome sugere, surge da aplicação da varredura de plano em conjunto com a heurística do método TDE. No TDE, primeiro se aplica a fase de busca de candidatos na primeira e última instância de tempo da janela-δ utilizando o índice baseado em grade. Já no algoritmo TDE_PS foi substituído o uso do índice de grade pela varredura de plano.

5 EXPERIMENTOS E RESULTADOS

Os algoritmos apresentados nesta dissertação foram submetidos a vários experimentos envolvendo diferentes conjuntos de dados (quatro reais e um sintético) e variando os parâmetros do padrão floco para se obter diferentes seletividades do padrão. Estes testes auxiliam na melhor avaliação das contribuições em diferentes cenários. Este capítulo apresentada os resultados obtidos por meio da combinação de cinco conjuntos de dados e a variação dos três parâmetros do padrão floco. O desempenho das contribuições foram comparados com o algoritmo BFE e suas correspondentes heurísticas (TDE, CRE e PFE) os quais são os algoritmos *baseline* e estado-da-arte no momento da elaboração desta dissertação. Além destes métodos uma modificação do algoritmo LCM_Flock [18] foi utilizada para comparar a eficácia das contribuições contra um método de outra categoria.

O restante do capítulo está organizado da seguinte forma: a Seção 5.1 descreve os conjuntos de dados utilizados nos experimentos; a Seção 5.2 mostra como as melhorias propostas na dissertação afetaram a performance do método básico BFE. a Seção 5.3 mostra o impacto na performance das heurísticas do BFE, através da aplicação das técnicas/estruturas propostas aqui no trabalho. por fim, na Seção 5.4 é feita uma análise de como a performance das melhorias se comportam em relação à assimetria dos dados de entrada.

5.1 Conjuntos de dados e ambiente de testes

Para melhor testar o desempenho dos algoritmos propostos neste trabalho foram utilizados cinco conjuntos de dados. Sendo eles: *Trucks*, *Buses*, *Cars*, *Caribous* e SG. O conjunto de dados *Trucks* tem 112.203 posições GPS geradas por 276 caminhões se movendo na área metropolitana de Atenas na Grécia. O segundo conjunto de dados *Buses* contém 66.096 posições de ônibus se movendo também em Atenas, Grécia. Já o *Cars* é composto de 134.264 posições coletadas a partir da movimentação de 183 carros privados se movimentando em Copenhague[1]. O conjunto *Caribous* é gerado a partir da movimentação de 43 cervídeos na região noroeste do Canadá, com um total de 15.796 localizações. Por fim, para testar a escalabilidade dos métodos utilizamos um conjunto de dados significantemente maior que os outros reais. Este conjunto é nomeado *SG* e contêm 2.548.084 posições geradas a partir da simulação de movimentação de 50 mil veículos na malha rodoviária de Cingapura, China.

Os experimentos foram realizados em máquina equipada com processador Intel®

[1] <www.daisy.aau.dk>

Conjunto	Posições	Objetos	μ	ϵ	δ
Cars	134,264	183	4, 6, ..., 20 **[5]**	0,4, 0,5, ..., 1,1 **[0,6]**	4, 6, ..., 20 **[10]**
Buses	66,096	145	4, 6, ..., 20 **[5]**	0,3, 0,4, ..., 1,0 **[0,7]**	4, 6, ..., 20 **[10]**
Trucks	112,203	276	4, 6, ..., 20 **[5]**	0,7, 0,8, ..., 1,4 **[0,9]**	4, 6, ..., 20 **[10]**
Caribous	15,796	43	2, 3, ..., 10 **[5]**	0,1, 0,2, ..., 0,8 **[0,7]**	4, 6, ..., 20 **[10]**
SG	2,548,084	50,000	4, 6, ..., 20 **[5]**	2,2, 2,6, ..., 5,0 **[3,0]**	4, 6, ..., 20 **[10]**

Tabela 3 – Número de objetos, quantidade de posições e valores testados para cada parâmetro do padrão.

Core™2 Quad-Q9505@2.83GHz, memória RAM de 4GB@1333MHz com barramento 64 bits, HD 500GB SATA 2.6@3Gb/s 7200 RPM e sistema operacional Elementary OS 0.3.2 Freya (Ubuntu 14.04.3) 64-bit. Todos os algoritmos foram desenvolvidos em C++11 e compilados com compilador GNU G++ 4.9.3.

5.2 Estudo de desempenho no algoritmo base

Esta seção apresenta os resultados obtidos nos testes para avaliação de desempenho, por meio da execução dos métodos sobre cada um dos conjuntos de dados e variando os parâmetros do padrão (μ, ϵ e δ) de acordo com o representado na Tabela 3. Nesta tabela são apresentados os valores testados para cada conjunto assim como os valores padrões utilizados (entre colchetes).

Nesta seção é discutido somente os resultados da aplicação das técnicas/melhorias propostas sobre o algoritmo BFE, sem as heurísticas. Estes resultados foram aceitos para publicação na GEOINFO [68], com exceção dos resultados do método LCM_Flock que é contribuição posterior.

5.2.1 Análise geral do resultado

A Figura 17 mostra os resultados experimentais de todos os conjuntos variando valores para os parâmetros μ – número de trajetórias, ϵ – distância entre trajetórias, e δ – duração do padrão (de acordo com a Tabela 3). Todos os gráficos mostram o tempo total (em segundos) necessário para processar o conjunto de dados *integralmente*. Com o objetivo de melhor apresentar a diferença de performance entre os métodos, os gráficos para os conjuntos *Caribous* e *SG* estão usando escala logarítimica. Além disso os gráficos para o conjunto *SG* não incluem o algoritmo LCM_Flock pois, além de ser o algoritmo com pior desempenho no geral, o número de discos que teriam que ser mantidos em memória seria muito além do que a máquina de teste suportaria sem a ocorrência de *disk thrashing*, o que poderia tornar os resultados enviesados.

De forma geral, os melhores resultados são apresentados pelos algoritmos que utili-

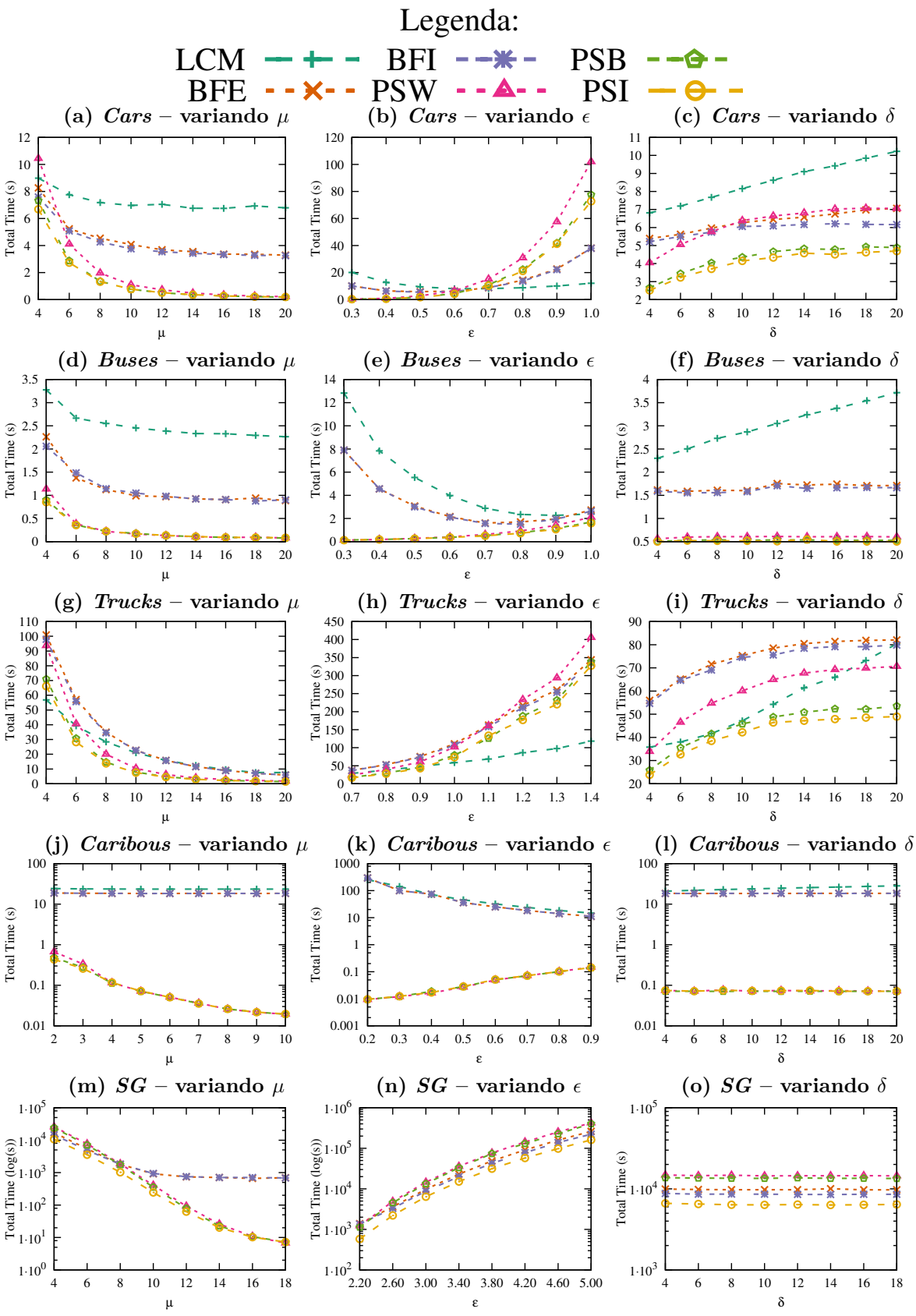


Figura 17 – Resultados dos experimentos com os conjuntos *Cars* (1ª linha), *Buses* (2ª linha), *Trucks* (3ª linha), *Caribous* (4ª linha) e *SG* (5ª linha) variando μ (cardinalidade - 1ª coluna), ϵ (diâmetro dos discos - 2ª coluna) e δ (duração - 3ª coluna).

zam a técnica de varredura de plano. Isto se deve ao fato que o BFE precisa construir um índice baseado em grade antes de começar a procurar por discos candidatos de fato. Outra conclusão que pode ser tirada ao observar os resultados é que à medida que aumenta-se μ o tempo necessário para detectar os flocos diminui. Este comportamento é esperado, uma vez que o os parâmetros do padrão se tornam mais seletivos, e, portanto os custos associados com a manutenção dos discos candidatos são reduzidos. Entretanto, quando os valores de ϵ diminuem, os tempos necessários para o método PSI e suas variantes processarem os conjuntos de dados também decrescem enquanto os tempos necessários de processamento para os métodos BFE e BFI decrescem até alcançar um valor ótimo e depois aumentam quando vai para abaixo desses valores (ver Figura 17(b) e 17(e)). Este comportamento é explicado pelo fato que o índice baseado em grade é altamente dependente da distribuição espacial dos dados (o índice tem baixa performance quando a ocupação das células é baixa). Já o algoritmo LCM_Flock obteve um dos piores resultados, pois só foi superior ao BFE e BFI o que era esperado quando se analisa o trabalho original [19, 18]. Este algoritmo só apresenta uma melhora na performance somente quando os valores de ϵ começam a ficar maiores (ver Figura 17(b)). Isto acontece devido ao modo em que o algoritmo faz a junção de discos em instantes de tempos diferentes. Neste algoritmo não é feito as operações de interseção de conjuntos que são computacionalmente caras, então quando o número de discos é razoavelmente alto em cada instante de tempo o algoritmo LCM [65] começa a compensar o custo de tradução das trajetórias para transações, já que o mesmo é altamente eficiente na deteção *itemsets* frequentes.

Em particular, nos conjuntos de dados *Caribous* e *SG* é possível ver que quando aumenta-se os valores de μ, o uso da varredura de plano e/ou do índice invertido tem um impacto maior no desempenho do que nos outros conjuntos, chegando a mais de duas ordens de magnitude em relação à utilização do índice baseado em grade. Isto se deve ao fato que em *Caribous* o número de objetos é relativamente pequeno (43) e eles estão dispersos no espaço, logo poucos flocos são encontrados. Por outro lado, o conjunto *SG* tem 50.000 objetos em cada instante de tempo, porém somente poucas posições por objeto (51 em média), resultando em poucos flocos também. Isso pode ser observado pelo número de respostas encontradas para estes conjuntos de dados (ver Tabela 4).

Analisando os resultados de forma mais profunda é possível concluir que o método PSI, que combina todas as técnicas apresentadas nesta dissertação, foi a que obteve o melhor desempenho. O impacto das assinaturas binárias, de forma isolada (PSB vs. PSW), no tempo de execução não foi expressivo, uma vez que ambos, BFE e PSW, já são otimizados de forma a utilizar as características espaciais dos discos/caixas para evitar operações de interseção de conjuntos desnecessárias. Este é o motivo principal da não inclusão dos resultados de um método que utilize apenas assinaturas binárias como melhoria. Além disso, quando se utiliza assinaturas binárias é preciso fazer um ajuste no tamanho das assinaturas para que não haja colisões na geração das assinaturas. Caso contrário à me-

dida que o número de colisões aumenta o número de falsos-positivos aumenta também, fazendo com que as operações de interseção continuem sendo feitas.

Por outro lado, a aplicação do índice invertido gerou um ganho substancial nos conjuntos de dados reais, especialmente quando se varia os valores de δ (ver terceira coluna da Figura 17); a utilização do índice também foi crucial no conjunto *SG* onde os métodos PSW e PSB tiveram desempenhos inferiores aos do BFE e BFI para valores baixos de μ, incluindo o padrão ($\mu = 5$). A razão do método PSI ter tido o melhor desempenho, nesse caso, é devido a grande quantidade de discos podados utilizando o índice invertido (chegando-se a eliminar 1.8×10^{11} discos de um total de 3.4×10^{11}). Isso acontece no conjunto *SG* pois há uma grande quantidade de discos candidatos por instante de tempo, porém eles não se mantêm ativo ao longo do tempo.

A Tabela 5 sumariza os resultados discutidos nesta seção. Pode-se concluir de maneira inteligível que o melhor método entre todos os testados, de forma geral, é o PSI. O método PSI só não tem performance melhor quando o raio do padrão se torna tão grande que o conjunto de dados se torna denso em relação aquele valor de ϵ, esse cenário é discutido mais a frente.

Conjunto de dados	**Valores padrão** (μ, ϵ, δ)	μ		ϵ		δ	
		Mín.	**Máx.**	**Mín.**	**Máx.**	**Mín.**	**Máx.**
Cars	1295	2598	0	203	5940	3906	319
Buses	319	787	20	36	1666	1247	34
Trucks	4926	5708	104	3741	15607	9934	756
Caribous	0	5063	0	0	150	20	0
SG	144	975	0	53	741	270	69

Tabela 4 – Número total de respostas para cada conjunto de dados em relação aos parâmetros padrão e os limites dos valores.

Conjunto	**Melhor – geral**	**Melhor** $-\ \mu$		**Melhor** $-\ \epsilon$		**Melhor** $-\ \delta$	
		Baixa	**Alta**	**Baixa**	**Alta**	**Curta**	**Longa**
Cars	PSI	PSB	PSI	PSI	LCM	PSI	PSI
Buses	PSI	PSI	PSI	PSI	PSI	PSI	PSI
Trucks	PSI	PSI	PSI	PSI	LCM	PSI	PSI
Caribou	PSI	PSI	PSI	PSI	PSI	PSI	PSI
SG	PSI	PSI	PSI	PSI	PSI	PSI	PSI

Tabela 5 – Resumo do desempenho dos algoritmos baseados no algoritmo básico.

5.3 Estudo de desempenho das heurísticas

As técnicas e estruturas apresentadas neste trabalho são generalistas, bem como a aplicação de tais técnicas. Devido a este fato foi possível não só estender o algoritmo base (BFE), bem como a suas heurísticas através da utilização da varredura de planos. Para avaliar o desempenho da aplicação da varredura de plano nas heurísticas comparou-se os algoritmos TDE, CRE, PFE contra o desempenho da versão de cada um desses utilizando a técnica. Nestes experimentos utilizamos o mesmo processo citado na Seção 5.1

A Figura 18 mostra o resultado dos experimentos realizados com todos os cinco conjuntos de dados. Novamente, todos os gráficos mostram o tempo total (em segundos) necessário para processar o conjunto de dados *integralmente*. Para auxiliar a diferença no ganho de desempenho os gráficos dos conjuntos *SG*, *Caribou* e *Cars* estão em escala logarítimica.

É possível observar que mais uma vez o método que teve o melhor desempenho de modo geral foi o PSI, exceto nos conjuntos *Cars* (somente distância) e *SG* onde os algoritmos TDE_PS e PFE_PS tiveram melhor rendimento, respectivamente. Entretanto, é importante observar que os métodos utilizando a varredura de plano tiveram sempre melhor desempenho que os métodos originais, isto é, a aplicação da técnica resultou em ganhos consideráveis no desempenho dos algoritmos. Isto se deve ao fato que nos algoritmos TDE e CRE a primeira etapa é a geração de candidatos utilizando o mesmo processo do BFE (que por sua vez utiliza o índice baseado em grade).

No conjunto de dados *Cars* é possível observar que ao passo que a distância aumenta o método TDE_PS tem um desempenho melhor que os outros. Para melhor explicar esse fato, é preciso primeiro explicar que ao aumentar o valor de ϵ o número de entidades em cada disco aumenta também. Como as entidades conjunto de dados *Cars* tendem a seguir trajetórias a diferentes destinos os grupos maiores de entidades tendem a se desfazer mais fácil em intervalos grandes de tempo. Devido a esse comportamento dos dados, a heurística TDE tende a podar mais discos candidatos quando observa a primeira e última instância de tempo da janela de tempo. Isto explica também o fato do algoritmo ter empatado com o PSI quando se varia δ (Figura 18(c)). Já que ao aumentar a janela de tempo a chance de os grupos se desmancharem durante toda a duração da janela também aumenta.

Já no conjunto *SG* o método PFE obteve o melhor desempenho. Neste caso, o conjunto de dados possui muitas entidades em cada instante de tempo e a construção do índice, por mais que seja computacionalmente cara, acaba sendo compensada pela eliminação da necessidade de operações de cálculo de distância, ordenações de todas as entidades e outras verificações que são precisas na varredura de plano. Além disso, como a distribuição dos dados é muito densa (ver Figura 19(d)) quase sempre que se gera caixas

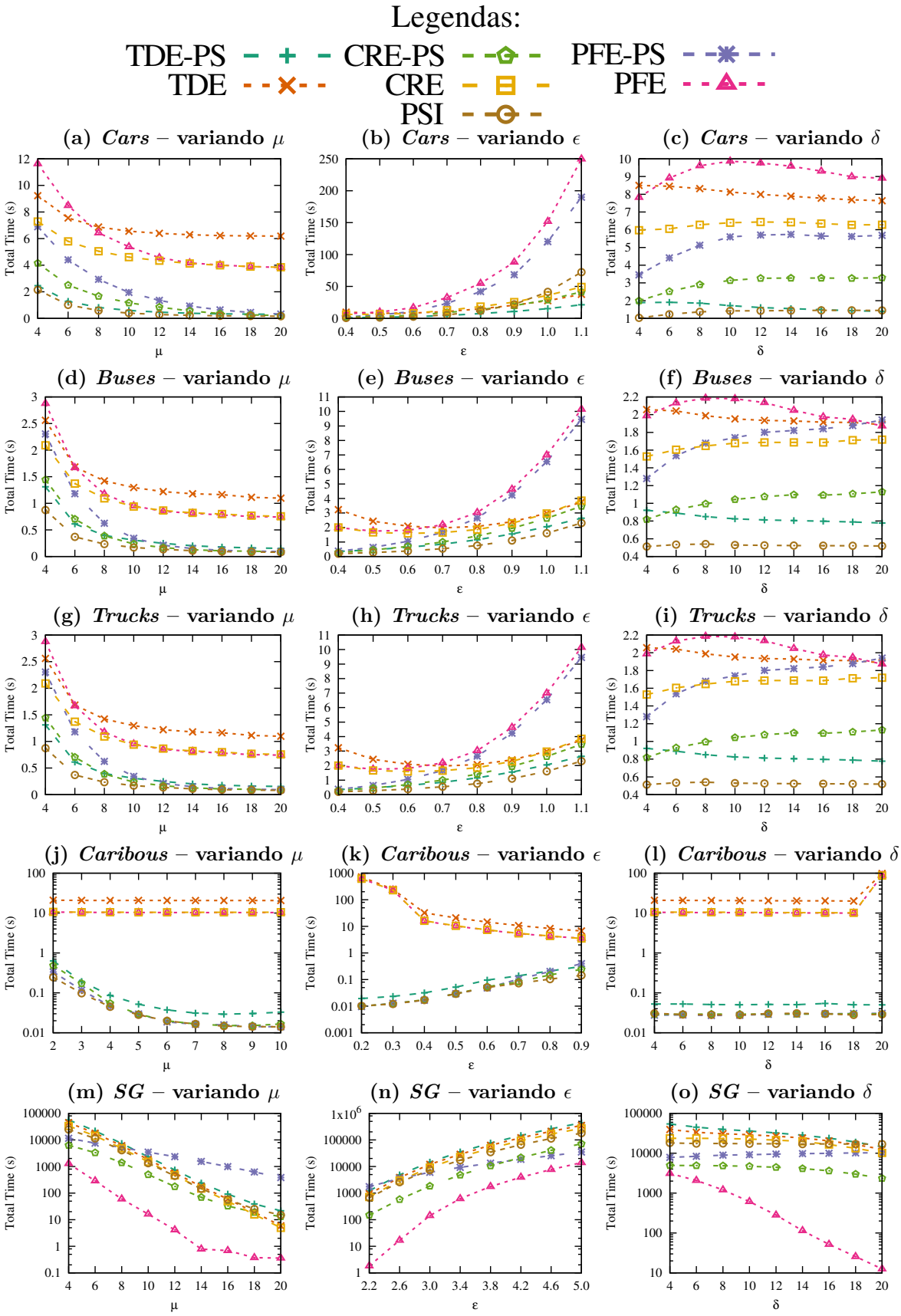


Figura 18 – Resultados dos experimentos com os conjuntos *Cars* (1ª linha), *Buses* (2ª linha), *Trucks* (3ª linha), *Caribous* (4ª linha) e *SG* (5ª linha) variando μ (cardinalidade - 1ª coluna), ϵ (diâmetro dos discos - 2ª coluna) e δ (duração - 3ª coluna).

na varredura de plano são gerados novos candidatos, logo não há poda de caixas sem candidatos. Quando isto acontece a geração de caixas se torna um *overhead*. Por fim, no conjunto *SG* a movimentação das entidades é randômico, o que faz com que entre instantes de tempo consecutivos a chance de várias entidades se manterem juntas.

Conjunto	**Melhor – geral**	**Melhor – μ**		**Melhor – ϵ**		**Melhor – δ**	
		Baixa	**Alta**	**Baixa**	**Alta**	**Curta**	**Longa**
Cars	PSI	PSI	PSI	PSI	LCM	PSI	PSI
Buses	PSI	PSI	PSI	PSI	PSI	PSI	PSI
Trucks	PSI	PSI	PSI	PSI	PSI	PSI	PSI
Caribou	PSI	PSI	PSI	PSI	PSI	PSI	PSI
SG	PFE	PFE	PFE	PFE	PFE	PFE	PFE

Tabela 6 – Resumo do desempenho dos algoritmos baseados nas heurísticas.

A Tabela 6 sumariza os resultados discutidos nesta seção. É possível ver que mais uma vez o melhor método no geral é o PSI. O único conjunto de dados onde o método não conseguiu manter o bom desempenho em relação aos outros métodos foi o *SG*. Entretanto, o conjunto *SG* tem um padrão de movimentação randômico, cenário que não se aplica ao mundo real, consequentemente para aplicações reais o PSI é a melhor opção, uma vez que obteve os melhores resultados nos conjuntos do mundo real.

5.4 Desempenho e assimetria dos dados

Como pode ser visto na Figura 17 o comportamento dos métodos apresentados neste trabalho pode variar muito entre conjunto de dados e para diferentes valores dos parâmetros do padrão. Somente observando os gráficos e tendo conhecimento das características globais dos conjuntos de dados não é suficiente para definir a causa de tal mudança no comportamento. Por isso, nesta seção é apresentada uma análise mais aprofundada dos resultados obtidos sob a luz de informações de assimetria dos conjuntos de dados estudados. Esta análise ajuda definir em quais situações a aplicação das técnicas propostas é mais eficiente ou não. Não obstante, esta análise auxilia a melhor descrever os conjuntos de dados e suas peculiaridades.

5.4.1 Varredura de plano *versus* densidade dos dados

Nesta subseção é apresentada a análise de como a técnica de varredura de plano se comporta em relação à densidade do conjunto de dados. Esta análise auxilia a descobrir os melhores cenários para a aplicação desta técnica e também aqueles onde ela deve ser evitada.

Quando as posições de um conjunto são esparsas em relação a um valor de ϵ dado, o índice baseado em grade tende a ficar maior o que leva a um aumento no tempo de

construção e de busca. Além disso, o número de cálculos que podem ser evitados durante o processo de se mover entre as células vizinhas (3×3) é reduzido, pois a sobreposição espacial entre as células é menor quando se comparado a dados densos. O método PSI e suas variações têm desempenho pior que o método BFE somente quando o valor de ϵ é grande. Isso é mais perceptível quando se observa o conjunto de dados *Cars* na Figura 17(b), entretanto esse tipo de comportamento pode se repetir em outros conjuntos de dados para valores de ϵ maiores que os testados neste trabalho. Neste trabalho o limite superior dos valores de ϵ são escolhidos de modo a manter alguma semântica espacial, já que na prática flocos formados por entidades que estão muito distantes uma das outras pode não fazer sentido. A razão para este resultado é que quando o diâmetro do padrão aumenta, o número de células no índice baseado em grade usado no BFE diminui fazendo com que o desempenho do índice aumente.

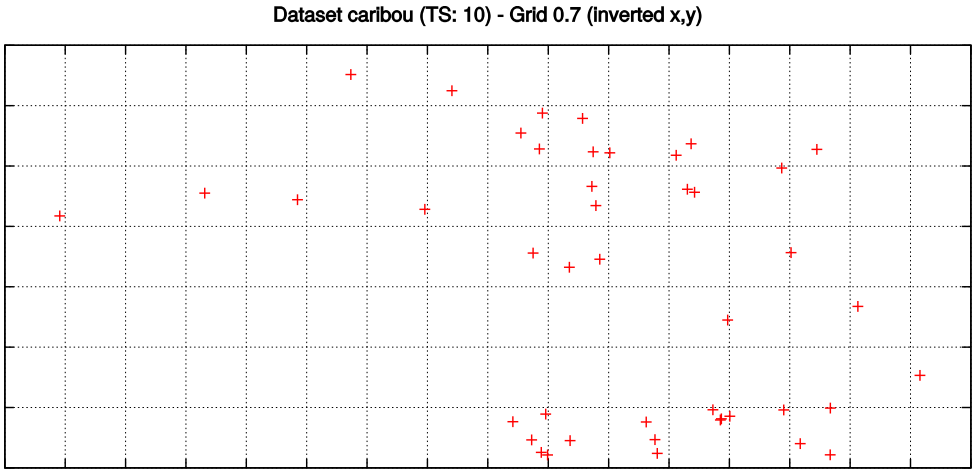
Um exemplo de conjunto de dados com dados que tende a ser esparso mesmo com valores pequenos de ϵ é o conjunto *Caribou*. Neste conjunto quando o diâmetro do padrão é pequeno o número de células criadas no índice de grade aumenta consideravelmente. Na Figura 19(a) é possível ver que mesmo com um valor razoavelmente grande os dados são esparsos em relação a ϵ (taxa de ocupação das células da grade é baixa). Quando há uma baixa ocupação das células ($\overline{ocup} \ll \mu$) e o tamanho do índice é grande o índice consome muito tempo tanto para construção quanto para a busca de discos candidatos uma vez que é preciso iterar todas as células não vazias, porém não é gerado candidatos já que não há entidades suficientes. Por outro lado, no PSI quando os dados se encontram esparsos os pontos que não possuem vizinhança são eliminados já na fase de geração das caixas. Isto faz com que o número de caixas a serem pesquisadas diminua muito e além disso o custo associado a ordenação das entidades também cai, uma vez que a ocupação das caixas também é baixa.

Desta forma é possível concluir que quando um conjunto de dados é esparso (baixa densidade) em relação a ϵ é quando os métodos que utilizam a varredura de plano têm o melhor desempenho em relação aos métodos que utilizam o índice de grade. Além disso é possível afirmar com mais certeza que ao aumentar o diâmetro do padrão o índice baseado em grade tende a melhorar o desempenho uma vez que ocorre mais sobreposição espacial dos pontos e os cálculos feitos ao criar o índice pode ser mais aproveitado.

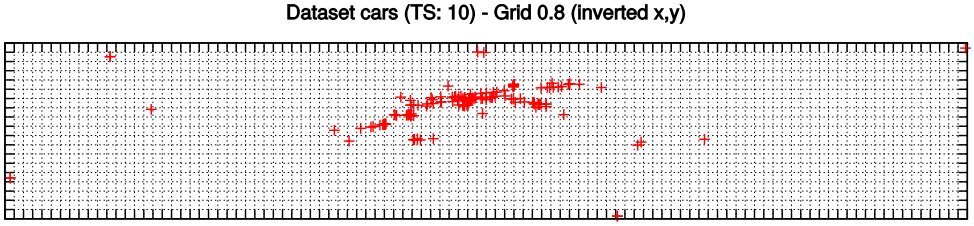
5.4.2 Índice invertido *versus* tendência de movimentação

Esta subseção apresenta a análise de melhores cenários para a aplicação da estrutura do índice invertido assim como os casos onde ele não deve ser aplicado. Além disso é mostrado como o desempenho dos métodos que utilizam o índice invertido é afetado em relação a tendência de movimentação das trajetórias em diferentes instantes de tempo.

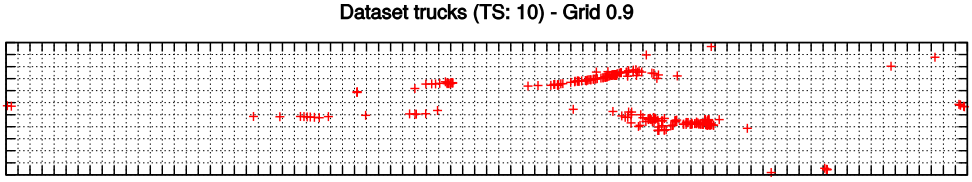
Uma das dificuldades na busca por padrões floco é que à medida que a seletividade



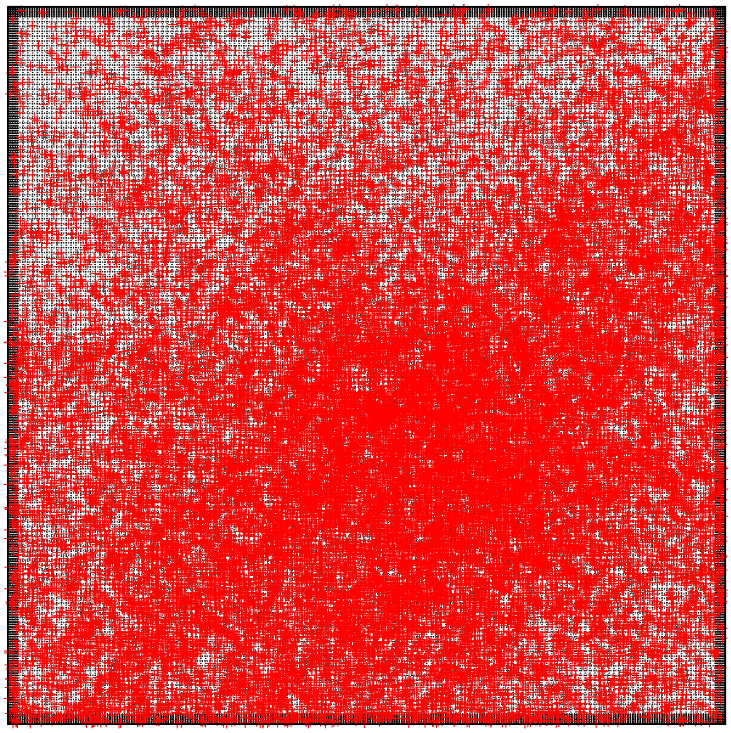
(a)



(b)



(c)



(d)

Figura 19 – Posições dos conjuntos *Caribous*, *Cars*, *Trucks* e *SG* em uma instância de tempo.

da consulta aumenta, o número de candidatos aumenta de forma rápida também. Esta situação gera um custo computacional muito alto não só em termos de espaço (associado a manter os discos em memória), mas de tempo também, uma vez que várias operações de interseção de conjuntos e outras comparações são feitas utilizando estes discos. Essa quantidade de discos mantidos está altamente relacionada com o comportamento e tendências de movimentação dos objetos estudados, pois se os objetos tendem a permanecer juntos por longos períodos de tempo os discos candidatos gerados em um instante de tempo tendem a permanecer "ativos" também. Quando esse tipo de situação ocorre o índice invertido não oferece grandes ganhos no desempenho dos métodos uma vez que não é possível podar grande quantidade de discos.

Em contrapartida no conjunto de dados SG a estrutura do índice invertido se provou essencial (ver quinta linha da Figura 17). Neste conjunto o número de discos gerados é muito grande chegando a $3,15 \times 10^5$ com os valores padrão do parâmetro e podem alcançar $1,26 \times 10^6$ quando a distância está no valor máximo. Apesar do grande número de discos gerados, depois que é feito a junção entre instantes de tempos diferentes o número de discos cai consideravelmente. A razão entre todos os discos encontrados e os discos após a junção em instantes diferentes pode passar de 11 vezes. Isso acontece devido a uma série de fatores, entre eles está o fato que o conjunto SG é muito denso (ver Figura 19(d)), além disso os objetos desse conjunto seguem um padrão de movimentação randômico e portanto entre instâncias de tempo consecutivas muitos dos grupos formados são desfeitos.

6 CONCLUSÃO

A grande disponibilidade de dados espaço-temporais tem despertado o interesse pela descoberta de padrões espaço-temporais. Entretanto, devido a quantidade de dados a ser processada, há demanda por algoritmos que possam descobrir tais padrões de forma eficiente. Além de serem eficientes, aplicações modernas requerem que tais algoritmos sejam capazes de processar fluxos de dados de em tempo real, pois atualmente muitos serviços como Swarm [1] e AccuTracking [2] oferecem dados nesta forma e é interessante se fazer o acompanhamento de comportamentos extraordinários em tempo real, principalmente na área de segurança.

6.1 Resumo das contribuições

Nesta dissertação foram desenvolvidas e avalio-se novas técnicas e estruturas com o objetivo de melhorar o desempenho de algoritmos para busca de flocos de forma *on-line*, estas são: varredura de plano e assinaturas binárias para auxiliar a encontrar discos candidatos e o índice invertido para podar discos em diferentes instantes de tempo que não são correlatos. Estas técnicas foram implementadas sobre o algoritmo base chamado BFE e suas heurísticas (TDE, PFE e CRE), gerando sete variações de algoritmos cada uma aplicando uma ou mais técnicas ao mesmo tempo, além disso foi implementado uma versão *on-line* do algoritmo LCM_Flock [19] para melhor comparar os resultados obtidos. Estes algoritmos desenvolvidos tiveram seus desempenhos comparado aos algoritmos originais, os quais são os métodos estado-da-arte para descoberta de flocos *on-line* e a versão *on-line* do LCM_Flock. Esta avaliação foi feita através da execução de uma bateria de testes utilizando cinco conjuntos de dados, sendo quatro reais e um sintético, variando os valores de cada um dos parâmetros do padrão (ϵ – distância, μ – cardinalidade e δ – duração).

Nos experimentos com os algoritmos básicos, isto é, quando a aplicação das técnicas foram comparadas com o BFE e LCM_Flock *on-line*, o tempo de execução dos algoritmos que utilizam as técnicas propostas (PSW, PSB e PSI) foram, de forma geral, menores que as dos algoritmos BFE, BFI e LCM. Embora o método PSI tenha tido o melhor desempenho de forma geral, por vezes a aplicação de todas as técnicas de uma só vez (em especial a combinação entre índice invertido e varredura de plano) não foi a que obteve o melhor resultado, nestas situações o método PSW foi o melhor. Por outro lado a estrutura de índice invertido, e o método PSI, se mostraram muito eficientes quando a cardinalidade do conjunto de discos candidatos em cada instante de tempo é muito alta e por isso obtiveram o melhor resultado no conjunto de dados sintético (SG).

[1] <https://pt.swarmapp.com/>
[2] <http://www.accutracking.com/>

Nos experimentos com os algoritmos baseados em heurísticas, isto é, quando as técnicas foram aplicadas às heurísticas do BFE (TDE, CRE e PFE), o algoritmo PSI continuou a ser o algoritmo com o maior ganho de desempenho exceto quando a distância máxima do padrão (ϵ) é alta no conjunto de dados *Cars*, entretanto para valores muito grandes de ϵ pode-se haver uma perda na semântica do padrão o que em aplicações reais pode não ser interessante.

De forma geral, os maiores ganhos da técnica de varredura de plano foram para valores de ϵ menores (em relação à densidade dos dados). Já para a estrutura índice invertido os maiores ganhos foram nos conjuntos onde o número de discos candidatos em cada instante de tempo é muito alto. Contudo, os ganhos foram expressivos na grande maioria dos cenários avaliados. Além disso, o fato dos maiores ganhos terem sido apresentados em conjuntos de dados capturados do mundo real (como *Caribous*) contribui para a aplicação destas técnicas em aplicações reais. Não obstante, outros algoritmos para busca de padrões espaço-temporais podem se beneficiar das técnicas apresentadas neste trabalho também, já que estas são generalistas e aplicáveis em vários contextos.

6.2 Trabalhos futuros

Uma das possibilidades de trabalho futuro diz respeito ao desenvolvimento de um método de classificação que, dado um conjunto de dados e a tríplice de parâmetros (μ, ϵ, δ), extrai uma série de estatísticas e métricas e prediz qual algoritmo será mais eficiente ao processar aquele conjunto. Isto é especialmente útil quando se utiliza conjuntos de dados sintéticos ou conjuntos reais onde o comportamento das entidades não é conhecido e não pode ser facilmente inferido. Neste caso, o usuário se encontra na situação onde deve escolher um algoritmo dentro de um universo de onze algoritmos (cinco originais [2] mais seis propostos na Seção 4.2), uma vez que não é tarefa simples deduzir qual algoritmo deve apresentar melhores resultados.

Outros trabalhos futuros poderiam analisar o impacto da aplicação das técnicas de varredura de plano em algoritmos como LCM_Flock, uma vez que esse utiliza o passo de detecção de candidatos do algoritmo BFE. Também poderiam ser aplicadas outras técnicas como assinaturas binárias e a estrutura do índice invertido nos métodos que utilizam as heurísticas e investigar o impacto da aplicação das técnicas nestes métodos. Por fim, um problema não abordado neste trabalho é aplicar técnicas que permitam considerar conjuntos de dados com variações de taxa de amostragem entre os diferentes objetos em estudo e/ou falhas de coleta de dados. Estes tipos de variações e inconsistências fazem com que os métodos propostos não reportem flocos que poderiam ser de interesse do usuário, demandando o uso de técnicas adicionais para tornar os métodos robustos a estas situações.

REFERÊNCIAS

[1] GUDMUNDSSON, J.; LAUBE, P.; WOLLE, T. Movement patterns in spatio-temporal data. *Encyclopedia of GIS*, p. 726–732, 2009. Disponível em: <http://www.gudmundsson.biz/HP_links/Papers/T-9.pdf>.

[2] VIEIRA, M. R.; BAKALOV, P.; TSOTRAS, V. J. On-line discovery of flock patterns in spatio-temporal data. In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '09*. New York, New York, USA: ACM Press, 2009. p. 286. ISBN 9781605586496. Disponível em: <http://dl.acm.org/citation.cfm?id=1653771.1653812>.

[3] INC., F. *About Foursquare*. 2016. <https://pt.foursquare.com/about>. Acessado em 17/01/2016. Disponível em: <https://pt.foursquare.com/about>.

[4] SERVICE, U. S. P. *Vehicles and fuel*. 2013. Acessado me 15/06/2015. Disponível em: <http://about.usps.com/what-we-are-doing/green/html/2014-sustainability-report/sustainabilityreport2013_011.htm>.

[5] CHEN, Y. L. et al. Market basket analysis in a multiple store environment. *Decision Support Systems*, v. 40, n. 2, p. 339–354, 2005. ISSN 01679236. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167923604000685>.

[6] DUAN, W. et al. Spatiotemporal evaluation of water quality incidents in Japan between 1996 and 2007. *Chemosphere*, 2013. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0045653513008060>.

[7] RAO, K.; GOVARDHAN, A.; RAO, K. Spatiotemporal Data Mining: Issues, Tasks and Applications. *Int'l J. Computer Science Eng. Survey*, 2012. Disponível em: <http://ijair.jctjournals.com/august2012/tn2.pdf>.

[8] BENKERT, M. et al. Reporting flock patterns. *Computational Geometry*, v. 41, n. 3, p. 111–125, 2008. Disponível em: <http://dx.doi.org/10.1016/j.comgeo.2007.10.003>.

[9] LI, X. et al. Effective Online Group Discovery in Trajectory Databases. *IEEE Transactions on Knowledge and Data Engineering*, v. 25, n. 12, p. 2752–2766, dez. 2013. ISSN 1041-4347. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6319297>.

[10] HAI, P. N.; PONCELET, P.; TEISSEIRE, M. ALL IN ONE: MINING MULTIPLE MOVEMENT PATTERNS. *International Journal of Information Technology & Decision Making*, v. 13, n. 8, 2014. Disponível em: <http://ix.cs.uoregon.edu/~haiphan/Publications/ijitdm2014.pdf>.

[11] LOGLISCI, C. Mining Trajectory Data for Discovering Communities of Moving Objects. *EDBT/ICDT Workshops*, 2014. Disponível em: <http://delab.csd.auth.gr/papers/MUD2014lmp.pdf>.

[12] LI, Z. et al. Swarm: Mining relaxed temporal moving object clusters. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 3, n. 1-2, p. 723–734, set. 2010. ISSN 2150-8097. Disponível em: <http://dl.acm.org/citation.cfm?id=1920841.1920934http://dl.acm.org/citation.cfm?id=1920934>.

[13] ZHENG, K. et al. Online Discovery of Gathering Patterns over Trajectories. *IEEE Transactions on Knowledge and Data Engineering*, v. 26, n. 8, p. 1974–1988, ago. 2014. ISSN 1041-4347. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6613478>.

[14] ESTER, M. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: SIMOUDIS, E.; HAN, J.; FAYYAD, U. M. (Ed.). *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*. AAAI Press, 1996. p. 226–231. Disponível em: <http://www.aaai.org/Library/KDD/1996/kdd96-037.php>.

[15] GUDMUNDSSON, J.; KREVELD, M. van. Computing longest duration flocks in trajectory data. In: *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems - GIS 2006*. New York, New York, USA: ACM Press, 2006. p. 35. ISBN 1595935290. Disponível em: <http://dl.acm.org/citation.cfm?id=1183471.1183479>.

[16] ARIMURA, H.; TAKAGI, T. Finding All Maximal Duration Flock Patterns in High-dimensional Trajectories. *Manuscript, DCS, IST, Hokkaido University, Apr*, 2014. Disponível em: <http://www-ikn.ist.hokudai.ac.jp/~arim/papers/maxlenflock201404r4.pdf>.

[17] AL-NAYMAT, G.; CHAWLA, S.; GUDMUNDSSON, J. Dimensionality reduction for long duration and complex spatio-temporal queries. In: *Proceedings of the 2007 ACM symposium on Applied computing - SAC '07*. New York, New York, USA: ACM Press, 2007. p. 393. ISBN 1595934804. Disponível em: <http://dl.acm.org/citation.cfm?id=1244002.1244095>.

[18] ROMERO, A. *Mining moving flock patterns in large spatio-temporal datasets using a frequent pattern mining approach*. Tese (Mestrado) — University of Twente, 2011. Disponível em: <http://admin.banrepcultural.org/sites/default/files/calderon_andres_tesis.pdf>.

[19] TURDUKULOV, U. et al. Visual mining of moving flock patterns in large spatio-temporal data sets using a frequent pattern approach. *International Journal of Geographical Information Science*, v. 28, n. 10, p. 2013–2029, 2014. ISSN 1365-8816. Disponível em: <http://www.tandfonline.com/doi/abs/10.1080/13658816.2014.889834>.

[20] ZHU, M. et al. Instant Discovery of Moment Companion Vehicles from Big Streaming Traffic Data. In: *2015 International Conference on Cloud Computing and Big Data (CCBD)*. IEEE, 2015. p. 73–80. ISBN 978-1-4673-8350-9. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7450533>.

[21] LI, Z. et al. MoveMine: Mining moving object data for discovery of animal movement patterns. *ACM Transactions on Intelligent Systems and Technology (TIST)*, ACM, v. 2, n. 4, p. 32, jul. 2011. ISSN 21576904. Disponível em: <http://dl.acm.org/citation.cfm?id=1989734.1989741http://dl.acm.org/citation.cfm?id=1989741>.

[22] MAKRI, P.; KALIVAS, D. Spatio-temporal analysis of groundwater pollution from BTEX in Thriassio Field, Attica, Greece. *10th IAEG Congress "Engineering Geology of Tomorrow cities"Nottingham, United Kingdom*, 2006. Disponível em: <http://iaeg2006.geolsoc.org.uk/cd/PAPERS/IAEG_409.PDF>.

[23] JEUNG, H. et al. Discovery of convoys in trajectory databases. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 1, n. 1, p. 1068–1080, ago. 2008. ISSN 21508097. Disponível em: <http://dl.acm.org/citation.cfm?id=1453856.1453971>.

[24] ERWIG, M. Toward Spatio-Temporal Patterns. *Spatio-Temporal Databases*, 2004. Disponível em: <http://link.springer.com/chapter/10.1007/978-3-662-09968-1_3>.

[25] TANG, L.; ZHENG, Y.; YUAN, J. On discovery of traveling companions from streaming trajectories. In: *IEEE 28th International Conference on Data Engineering (ICDE)*. [s.n.], 2012. p. 186—-197. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6228083>.

[26] WANG, D.; CHENG, T. A spatio-temporal data model for activity-based transport demand modelling. *International Journal of Geographical Information Science*, Taylor & Francis Group, v. 15, n. 6, p. 561–585, set. 2001. ISSN 1365-8816. Disponível em: <http://www.tandfonline.com/doi/abs/10.1080/13658810110046934>.

[27] BOGORNY, V.; SHEKHAR, S. Spatial and spatio-temporal data mining. *IEEE International Conference on Data Mining (ICDM)*, 2010. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5694111>.

[28] CHENG, T.; HAWORTH, J. Spatiotemporal Data Mining. *Handbook of Regional of Science*, 2014. Disponível em: <http://link.springer.com/10.1007/978-3-642-23430-9_68>.

[29] DODGE, S.; WEIBEL, R.; LAUTENSCHüTZ, A.-K. Towards a taxonomy of movement patterns. *Information Visualization*, Palgrave Macmillan, v. 7, n. 3-4, p. 240–252, ago. 2008. ISSN 1473-8716. Disponível em: <http://dl.acm.org/citation.cfm?id=1594710.1594716>.

[30] LAUBE, P.; KREVELD, M. van; IMFELD, S. Finding REMO—detecting relative motion patterns in geospatial lifelines. *Developments in spatial data handling*, 2005. Disponível em: <http://link.springer.com/chapter/10.1007/3-540-26772-7_16>.

[31] ANDERSSON, M. et al. Reporting leadership patterns among trajectories. In: *Proceedings of the 2007 ACM symposium on Applied computing - SAC '07*. New York, New York, USA: ACM Press, 2007. p. 3. ISBN 1595934804. Disponível em: <http://dl.acm.org/citation.cfm?id=1244002.1244004>.

[32] DETTKI, H.; ERICSSON, G.; EDENIUS, L. Real-time moose tracking: an internet based mapping application using GPS/GSM-collars in Sweden. *Alces*, 2004. Disponível em: <http://www.researchgate.net/profile/Goeran_Ericsson/publication/236993930_Real-time_moose_tracking_an_internet_based_mapping_application_using_gpsgsm-collars_in_Sweden/links/02e7e5373a300d1e58000000.pdf>.

[33] BALME, G.; HUNTER, L. Mortality in a protected leopard population, Phinda Private Game Reserve, South Africa: a population in decline. *Ecological Journal*, 2004. Disponível em: <https://www.panthera.org/sites/default/files/Balme_Hunter_2004_Mortality_in_a_protected_leopard_population_in_South_Africa_0.pdf>.

[34] GUDMUNDSSON, J.; KREVELD, M. van; SPECKMANN, B. Efficient Detection of Motion Patterns in Spatio-temporal Data Sets. In: *Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems*. [s.n.], 2004. p. 250–257. ISBN 1-58113-979-9. Disponível em: <http://doi.acm.org/10.1145/1032222.1032259>.

[35] WANG, Y.; LIM, E.; HWANG, S. Efficient mining of group patterns from user movement data. *Data Knowl. Eng.*, v. 57, n. 3, p. 240–282, 2006. Disponível em: <http://dx.doi.org/10.1016/j.datak.2005.04.006>.

[36] Nhat Hai, P.; PONCELET, P.; TEISSEIRE, M. GeT_Move: An Efficient and Unifying Spatio-temporal Pattern Mining Algorithm for Moving Objects. In: HOLLMéN, J.; KLAWONN, F.; TUCKER, A. (Ed.). *Advances in Intelligent Data Analysis XI*. Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, v. 7619). p. 276–288. ISBN 978-3-642-34155-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-34156-4_26>.

[37] HAI, P. N. *Mining Object Movement Patterns from Trajectory Data*. Tese (Doutorado) — Université Montpellier, 2013. Disponível em: <http://www.lirmm.fr/~poncelet/publications/papers/these_Phan.pdf>.

[38] ZHENG, K.; ZHENG, Y. On discovery of gathering patterns from trajectories. In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. [s.n.], 2013. p. 242—-253. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6544829>.

[39] ROTE, G. *Computing the minimum Hausdorff distance between two point sets on a line under translation*. 1991, 123–127 p.

[40] DOUGLAS, D.; PEUCKER, T. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, v. 10, n. 2, p. 112—-122, 1973. Disponível em: <http://utpjournals.metapress.com/index/FM576770U75U7727.pdf>.

[41] TZVETKOV, P. Mining top-k frequent closed patterns without minimum support. In: *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* IEEE Comput. Soc, 2002. p. 211–218. ISBN 0-7695-1754-4. Disponível em: <http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1183905>.

[42] ZHANG, J. et al. On Retrieving Moving Objects Gathering Patterns from Trajectory Data via Spatio-temporal Graph. In: *2014 IEEE International Congress on Big Data*. IEEE, 2014. p. 390–397. ISBN 978-1-4799-5057-7. Disponível em: <http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6906807>.

[43] BRON, C.; KERBOSCH, J. *Algorithm 457: finding all cliques of an undirected graph*. 1973, 575–577 p.

[44] LAUBE, P.; IMFELD, S. Analyzing relative motion within groups oftrackable moving point objects. *Geographic information science*, 2002. Disponível em: <http://link.springer.com/chapter/10.1007/3-540-45799-2_10>.

[45] BENKERT, M. et al. Reporting flock patterns. In: *Algorithms – ESA 2006*. Springer Berlin Heidelberg, 2006. v. 4168, p. 660–671. ISBN 978-3-540-38875-3. Disponível em: <http://link.springer.com/chapter/10.1007/11841036_59>.

[46] ROSERO, O. E. C.; ROMERO, A. O. C. Performance analysis of flock pattern algorithms in spatio-temporal databases. In: *2014 XL Latin American Computing Conference (CLEI)*. IEEE, 2014. p. 1–6. ISBN 978-1-4799-6130-6. Disponível em: <http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6965180>.

[47] HOTELLING, H. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 1933. Disponível em: <http://psycnet.apa.org/journals/edu/24/6/417/>.

[48] BINGHAM, E.; MANNILA, H. Random projection in dimensionality reduction: applications to image and text data. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. [s.n.], 2001. p. 245—-250. Disponível em: <http://dl.acm.org/citation.cfm?id=502546>.

[49] JOHNSON, W.; LINDENSTRAUSS, J. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics*, v. 26, n. 1, p. 189–206, 1984. Disponível em: <http://www.ams.org/books/conm/026/>.

[50] PAPADIMITRIOU, C.; TAMAKI, H. Latent semantic indexing: A probabilistic analysis. In: *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. [s.n.], 1998. p. 159—-168. Disponível em: <http://dl.acm.org/citation.cfm?id=275505>.

[51] SHAMOS, M. I.; HOEY, D. Geometric intersection problems. In: *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*. IEEE, 1976. p. 208–215. ISSN 0272-5428. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4567905>.

[52] GRAHAM, R. L. An efficient algorith for determining the convex hull of a finite planar set. *Information processing letters*, v. 1, n. 4, p. 132—-133, 1972. Disponível em: <http://www.math.ucsd.edu/~ronspubs/72_10_convex_hull.pdf>.

[53] GUIBAS, L.; STOLFI, J. On computing all north-east nearest neighbors in the L 1 metric. *Information Processing Letters*, 1983. Disponível em: <http://www.sciencedirect.com/science/article/pii/0020019083900455>.

[54] NIEVERGELT, J.; HINRICHS, K. H. *Algorithms and data structures - with applications to graphics and geometry*. [S.l.]: vdf, 1999. (vdf Lehrbuch). ISBN 978-3-7281-2523-1.

[55] HINRICHS, K.; NIEVERGELT, J.; SCHORN, P. Plane-sweep solves the closest pair problem elegantly. *Information Processing Letters*, v. 26, n. 5, p. 255—-261, 1988. Disponível em: <http://www.sciencedirect.com/science/article/pii/0020019088901500>.

[56] EPPSTEIN, D.; GOODRICH, M.; SUN, J. Skip quadtrees: Dynamic data structures for multidimensional point sets. *nternational Journal of Computational Geometry & Applications*, v. 18, n. 01n02, p. 131—-160, 2008. Disponível em: <http://www.worldscientific.com/doi/abs/10.1142/S0218195908002568>.

[57] BERN, M. Approximate closest-point queries in high dimensions. *Information Processing Letters*, 1993. Disponível em: <http://www.sciencedirect.com/science/article/pii/002001909390222U>.

[58] EPPSTEIN, D.; GOODRICH, M. T.; SUN, J. Z. The skip quadtree. In: *Proceedings of the twenty-first annual symposium on Computational geometry - SCG '05*. New York, New York, USA: ACM Press, 2005. p. 296. ISBN 1581139918. Disponível em: <http://dl.acm.org/citation.cfm?id=1064092.1064138>.

[59] BLOOM, B. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970. Disponível em: <http://dl.acm.org/citation.cfm?id=362692>.

[60] BLUSTEIN, J.; EL-MAAZAWI, A. *Bloom filters: a tutorial, analysis, and survey*. [S.l.], 2002. Disponível em: <https://www.cs.dal.ca/sites/default/files/technical_reports/CS-2002-10.pdf>.

[61] KNUTH, D. E. Retrieval on secondary keys. In: . [S.l.: s.n.], 1997. v. 3, cap. 6, p. 550–567.

[62] ZOBEL, J.; MOFFAT, A.; RAMAMOHANARAO, K. Inverted files versus signature files for text indexing. *ACM Trans. Database Syst.*, v. 23, n. 4, p. 453–490, 1998. Disponível em: <http://doi.acm.org/10.1145/296854.277632>.

[63] ZOBEL, J.; MOFFAT, A. Inverted files for text search engines. *ACM Comput. Surv.*, v. 38, n. 2, p. 6, 2006.

[64] UNO, T. et al. An efficient algorithm for enumerating closed patterns in transaction databases. In: SUZUKI, E.; ARIKAWA, S. (Ed.). *Discovery Science, 7th International Conference, DS 2004, Padova, Italy, October 2-5, 2004, Proceedings*. Springer, 2004. (Lecture Notes in Computer Science, v. 3245), p. 16–31. Disponível em: <http://dx.doi.org/10.1007/978-3-540-30214-8_2>.

[65] UNO, T.; KIYOMI, M.; ARIMURA, H. Lcm ver.3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In: *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*. New York, NY, USA: ACM, 2005. (OSDM '05), p. 77–86. ISBN 1-59593-210-0. Disponível em: <http://doi.acm.org/10.1145/1133905.1133916>.

[66] GENG, X.; UNO, T.; ARIMURA, H. Trajectory Pattern Mining in Practice: Algorithms for Mining Flock Patterns from Trajectories. In: *5th International Conference On Knowledge Discovery and Information Retrieval (KDIR)*. [s.n.], 2013. Disponível em: <http://www-ikn.ist.hokudai.ac.jp/~arim/papers/fpmInPractice201303.pdf>.

[67] GENG, X. et al. Enumeration of complete set of flock patterns in trajectories. In: *Proceedings of the 5th ACM SIGSPATIAL International Workshop on GeoStreaming - IWGS '14*. New York, New York, USA: ACM Press, 2014. p. 53–61. ISBN

9781450331395. Disponível em: <http://dl.acm.org/citation.cfm?id=2676552.2676560>.

[68] TANAKA, P. S.; VIEIRA, M. R.; KASTER, D. S. Efficient algorithms to discover flock patterns in trajectories. In: *XVI Brazilian Symposium on Geoinformatics (GEOINFO)*. São Paulo, Brazil: [s.n.], 2015. v. 1, n. XVI, p. 56–67.

[69] GOEL, A.; GUPTA, P. Small subset queries and bloom filters using ternary associative memories, with applications. In: ACM. *ACM SIGMETRICS Performance Evaluation Review*. [S.l.], 2010. v. 38, n. 1, p. 143–154.

TRABALHOS PUBLICADOS PELO AUTOR

Trabalhos publicados pelo autor durante o programa.

1. Tanaka, P. S., Vieira, M. R., & Kaster, D. S. (2015). Efficient Algorithms to Discover Flock Patterns in Trajectories. Em XVI Brazilian Symposium on Geoinformatics (GEOINFO) (Vol. 1, pp. 56–67).