
Agrupamento de dados em fluxos contínuos com estimativa automática do número de grupos

Jonathan de Andrade Silva

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Agrupamento de dados em fluxos contínuos com estimativa automática do número de grupos¹

Jonathan de Andrade Silva

Orientador: Prof. Dr. Eduardo Raul Hruschka

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA.*

USP – São Carlos
Maio de 2015

¹ Trabalho Realizado com Suporte Financeiro da FAPESP, processo n. 2010/15049-7, e CAPES.

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

S586a Silva, Jonathan de Andrade
 Agrupamento de dados em fluxos contínuos com
 estimativa automática do número de grupos /
 Jonathan de Andrade Silva; orientador Eduardo Raul
 Hruschka. -- São Carlos, 2015.
 175 p.

Tese (Doutorado - Programa de Pós-Graduação em
Ciências de Computação e Matemática Computacional) --
Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2015.

1. Agrupamento de dados. 2. Fluxos contínuos de
dados. 3. Algoritmos evolutivos. 4. Agrupamento de
dados online. I. Hruschka, Eduardo Raul, orient.
II. Título.

À minha família

Agradecimentos

Gostaria aqui de expressar meus sinceros agradecimentos, minha gratidão e admiração pelo meu amigo e orientador Prof. Eduardo Raul Hruschka com quem pude contar ao longo desses anos de pós-graduação e que também tenho como exemplo profissional.

Agradeço ao Prof. João Gama, meu supervisor durante o doutorado sanduíche que realizei na Universidade do Porto, por ter me aceitado em seu grupo de pesquisa e seus ensinamentos durante o desenvolvimento desse trabalho.

À minha família, meus pais Maria e Zilto, minhas irmãs Jennifer e Jéssica, meus tios Perpétua e Levi, minha avó Maria Selma, pois foram pessoas fundamentais para que eu chegassem até aqui e sempre me apoiaram em todas as minhas decisões.

Gostaria também de agradecer aos amigos de longa data que me acompanharam desde a graduação e hoje trabalhamos juntos, Bruno Brandoli e Wesley Nunes e outros amigos que fiz no ICMC, em especial ao André Rossi, Danilo Horta, Elaine Ribeiro, Lucas Vendramin, Luís Paulo, Luiz Coletta, Murilo Naldi, Pablo Andretta, Renato Ramos, Ricardo Cerri, Ricardo Rios, Rodrigo Barros, Rosane Maffei, Tiago Silva, pelos momentos de discussão sobre assuntos relacionados a este trabalho, pelos momentos de companheirismo e confraternização e por estarem sempre dispostos a ajudar.

A todos os funcionários do ICMC da USP, pela competência e dedicação.

À FAPESP pelo apoio financeiro para realização desse trabalho (processo n. 2010/15049-7).

Resumo

Técnicas de agrupamento de dados usualmente assumem que o conjunto de dados é de tamanho fixo e pode ser alocado na memória. Neste contexto, um desafio consiste em aplicar técnicas de agrupamento em bases de dados de tamanho ilimitado, com dados gerados continuamente e em ambientes dinâmicos. Dados gerados nessas condições originam o que se convencionou chamar de Fluxo Contínuo de Dados (FCD). Em aplicações de FCD, operações de acesso aos dados são restritas a apenas uma leitura ou a um pequeno número de acessos aos dados, com limitações de memória e de tempo de processamento. Além disso, a distribuição dos dados gerados por essas fontes pode ser não estacionária, ou seja, podem ocorrer mudanças ao longo do tempo, denominadas mudanças de conceito. Nesse sentido, algumas técnicas de agrupamento em FCD foram propostas na literatura. Muitas dessas técnicas são baseadas no algoritmo das k -Médias. Uma das limitações do algoritmo das k -Médias consiste na definição prévia do número de grupos. Ao se assumir que o número de grupos é desconhecido *a priori* e que deveria ser estimado a partir dos dados, percorrer o grande espaço de soluções possíveis (tanto em relação ao número de grupos, k , quanto em relação às partições possíveis para um determinado k) torna desafiadora a tarefa de agrupamento de dados - ainda mais sob a limitação de tempo e armazenamento imposta em aplicações de FCD. Neste contexto, essa tese tem como principais contribuições: (i) adaptar algoritmos que têm sido usados com sucesso em aplicações de Fluxo Contínuo de Dados (FCD) nas quais k é conhecido para cenários em que se deseja estimar o número de grupos; (ii) propor novos algoritmos para agrupamento que estimem k automaticamente a partir do FCD; (iii) avaliar sistematicamente, e de maneira quantitativa, os algoritmos propostos de acordo com as características específicas dos cenários de FCD. Foram desenvolvidos 14 algoritmos de agrupamento para FCD capazes de estimar o número de grupos a partir dos dados. Tais algoritmos foram avaliados em seis bases de dados artificiais e duas bases de dados reais amplamente utilizada na literatura. Os algoritmos desenvolvidos podem auxiliar em diversas áreas da Mineração em FCD. Os algoritmos evolutivos desenvolvidos mostraram a melhor relação de custo-benefício entre eficiência computacional e qualidade das partições obtidas.

Palavras-chave: agrupamento de dados, fluxo contínuo de dados, algoritmos evolutivos.

x

Abstract

Several algorithms for clustering data streams based on k-Means have been proposed in the literature. However, most of them assume that the number of clusters, k , is known a priori by the user and can be kept fixed throughout the data analysis process. Besides the difficulty in choosing k , data stream clustering imposes several challenges to be dealt with, such as addressing non-stationary, unbounded data that arrives in an online fashion. In data stream applications, the dataset must be accessed in order and that can be read only once or a small number of times. In this context, the main contributions of this thesis are: (i) adapt algorithms that have been used successfully in data stream applications where k is known to be able to estimate the number of clusters from data; (ii) propose new algorithms for clustering to estimate k automatically from the data stream; (iii) evaluate the proposed algorithms according to different scenarios. Fourteen clustering data stream algorithms were developed which are able to estimate the number of clusters from data. They were evaluated in six artificial datasets and two real-world datasets widely used in the literature. The developed algorithms are useful for several data mining tasks. The developed evolutionary algorithms have shown the best trade-off between computational efficiency and data partition quality.

Keywords: data stream, clustering, evolutionary algorithms.

Lista de Algoritmos

2.1	Inserção de um novo objeto no conjunto de blocos B (Ackermann et al., 2012). A função <i>coresetReduction</i> (·) (linha 7) recebe $2m$ objetos e retorna m objetos sumarizados.	20
2.2	Variante do k -Médias para lidar com sumários estatísticos (<i>Cluster Features</i> (CFs)) (Aggarwal et al., 2003).	27
2.3	Algoritmo k -means++ (Arthur e Vassilvitskii, 2007).	27
4.1	Algoritmo Evolutivo para Agrupamento de Dados.	55
4.2	Operador de mutação OM1	56
4.3	Operador de mutação OM2	57

Lista de Figuras

2.1	Estrutura de trabalho (framework) para os algoritmos de agrupamento em FCD.	9
2.2	Estrutura de dados CF-Tree.	12
2.3	Estrutura de microgrupo usada no algoritmo <i>CluStream</i> (Aggarwal et al., 2003).	14
2.4	Estruturas de <i>Temporal Cluster Features</i> (TCF) e <i>Exponential Histograms Cluster Features</i> (EHCF) usadas no algoritmo <i>SWClustering</i> (Zhou et al., 2008).	15
2.5	Vetores de protótipos utilizada no algoritmo <i>Stream</i> (Guha et al., 2000).	18
2.6	Modelo de janela deslizante.	22
2.7	Modelo de janela amortizada.	23
2.8	Modelo de janela de ponto de referência para um intervalo de tempo de tamanho 13.	24
3.1	Exemplo ilustrativo do algoritmo <i>CluStream</i> .	34
3.2	Ilustração do passo de redução utilizado no StreamKM++ (Ackermann et al., 2012).	35
3.3	Exemplo ilustrativo do passo de redução usado no algoritmo StreamKM++ (Ackermann et al., 2012).	36
3.4	Arquitetura do método proposto.	39
3.5	Exemplo de execução do método que é utilizado para estimar o número de grupos a partir dos dados.	40
3.6	Arquitetura do método incremental proposto.	44
4.1	Visão geral da execução do <i>Fast Evolutionary Algorithm for Clustering Data Stream based on Page Hinkley Test</i> (FEACS-PHT). O Passo 3 está relacionado à estimação do número de grupos. Os passos 1 e 2 estão relacionados à manutenção dos grupos de dados (encontrar o grupo mais próximo e remover o grupo mais antigo) e detectar mudanças (monitorar as distâncias entre os objetos e seus respectivos centroides), respectivamente.	52
4.2	Visão geral da execução do <i>Fast Evolutionary Algorithm for Clustering Data Stream based on Incremental Simplified Silhouette</i> (FEACS-ISS). O Passo 2 está relacionado à estimação do número de grupos. O passo 1 está relacionado à manutenção dos grupos de dados (encontrar o grupo mais próximo) e detectar mudanças (monitorar as distâncias entre os objetos e seus respectivos centroides), respectivamente.	53
4.3	Esquema de codificação para uma partição de dados.	54

4.4 Exemplo ilustrativo da execução do <i>Page-Hinkley Test</i> (PHT). Este teste monitora as distâncias médias entre os objetos e os seus centroides mais próximos.	60
5.1 Exemplo ilustrativo da evolução dos grupos ao longo do tempo — simulação realizada via gerador de dados sintético. As imagens da esquerda ilustram os grupos antes das modificações, enquanto que as imagens da direita ilustram o resultado das modificações.	65
5.2 Tempo de processamento dos algoritmos CLS-B _k M, CLS-IB _k M, CLS-MEO _k e CLS-MEO _{kI}	71
5.3 Tempo de processamento dos algoritmos SKM-B _k M, SKM-IB _k M, SKM-MEO _k e SKM-MEO _{kI}	72
5.4 Tempo de processamento (escala logarítmica) dos algoritmos SLS-B _k M, SLS-IB _k M, SLS-MEO _k e SLS-MEO _{kI} - Base de dados artificial com 2 dimensões.	73
5.5 Tempo de processamento (escala logarítmica) dos algoritmos SLS-B _k M, SLS-IB _k M, SLS-MEO _k e SLS-MEO _{kI} - Base de dados artificial com 4 dimensões.	74
5.6 Tempo de processamento (escala logarítmica) dos algoritmos SLS-B _k M, SLS-IB _k M, SLS-MEO _k e SLS-MEO _{kI} - Base de dados artificial com 8 dimensões.	74
5.7 Tempo de processamento (escala logarítmica) dos algoritmos SLS-B _k M, SLS-IB _k M, SLS-MEO _k e SLS-MEO _{kI}	75
5.8 Tempo de processamento dos algoritmos FEACS-SSI e FEACS-PHT - Base de dados artificial com 2 dimensões.	76
5.9 Tempo de processamento dos algoritmos FEACS-SSI e FEACS-PHT - Base de dados artificial com 4 dimensões.	77
5.10 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (Silhueta Simplificada Incremental (ssi)), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos CLS-B _k M (Figura 5.10a) e CLS-IB _k M (Figura 5.10b).	78
5.11 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (ssi), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos CLS-MEO _k (Figura 5.11a) e CLS-MEO _{kI} (Figura 5.11b).	79
5.12 Frequência de cada operação de inserção, remoção e união de microgrupos e sua soma (total) para os algoritmos CLS-B _k M (Figura 5.12a) e CLS-IB _k M (Figura 5.12b).	80
5.13 Frequência de cada operação de inserção, remoção e união de microgrupos e sua soma (total) para os algoritmos CLS-MEO _k (Figura 5.13a) e CLS-MEO _{kI} (Figura 5.13b).	81
5.14 Tempo de processamento em milissegundos de cada componente (abstração de dados, silhueta simplificada incremental (ssi), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) para os algoritmos SKM-B _k M (Figura 5.14a) e SKM-IB _k M (Figura 5.14b).	82

5.15 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos SKM-MEO k (Figura 5.15a) e SKM-MEO k I (Figura 5.15b).	83
5.18 Tempo de processamento de cada componente do algoritmo FEACS-SSI - Base de dados artificial com 2 dimensões.	84
5.16 Tempo de processamento em milissegundos de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) para os algoritmos SLS-B k M (Figura 5.16a) e SLS-IB k M (Figura 5.16b) - para a base de dados artificial com 2 dimensões.	85
5.17 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos SLS-MEO k (Figura 5.17a) e SLS-MEO k I (Figura 5.17b) - Base de dados artificial com 2 dimensões.	86
5.19 Tempo de processamento de cada componente do algoritmo FEACS-PHT - Base de dados artificial com 2 dimensões.	87
5.20 Tempo de processamento de cada componente do algoritmo FEACS-SSI - Base de dados artificial com 16 dimensões.	87
5.21 Tempo de processamento de cada componente do algoritmo FEACS-PHT - Base de dados artificial com 16 dimensões.	87
5.22 Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos CLS-B k M, CLS-IB k M, CLS-MEO k e CLS-MEO k I - Base de dados artificial com 2 dimensões. Valores positivos indicam que o valor de \hat{k} foi superestimado e valores negativos indicam que o número de grupos foi subestimado.	88
5.23 Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos CLS-B k M, CLS-IB k M, CLS-MEO k e CLS-MEO k I para as bases de dados artificiais com 4 e 8 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k	89
5.24 Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos CLS-B k M, CLS-IB k M, CLS-MEO k e CLS-MEO k I para as bases de dados artificiais com 16, 32 e 64 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k	90
5.25 Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SKM-B k M, SKM-IB k M, SKM-MEO k e SKM-MEO k I para a base de dados artificial bidimensional. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k	91

- 5.26 Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SKM-B_kM, SKM-IB_kM, SKM-MEO_k e SKM-MEO_kI para as bases de dados artificiais com 4 e 8 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k 92
- 5.27 Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SKM-B_kM, SKM-IB_kM, SKM-MEO_k e SKM-MEO_kI para as bases de dados artificiais com 16, 32 e 64 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k 93
- 5.28 Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SLS-B_kM, SLS-IB_kM, SLS-MEO_k e SLS-MEO_kI para a base de dados artificial bidimensional. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k 94
- 5.29 Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SLS-B_kM, SLS-IB_kM, SLS-MEO_k e SLS-MEO_kI para as bases de dados artificiais com 4 e 8 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k 95
- 5.30 Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SLS-B_kM, SLS-IB_kM, SLS-MEO_k e SLS-MEO_kI para as bases de dados artificiais com 16, 32 e 64 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k 96
- 5.31 Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos FEACS-SSI e FEACS-PHT para a base de dados artificial bidimensional. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k 97
- 5.32 Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos FEACS-SSI e FEACS-PHT para as bases de dados artificiais com 4 e 8 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k 98
- 5.33 Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos FEACS-SSI e FEACS-PHT para as bases de dados artificiais com 16, 32 e 64 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k 99

5.34 Exemplo ilustrativo da evolução do número de grupos e a sua diferença com relação ao número de grupos encontrado pelos algoritmos para a base de dados artificial.	102
5.35 Tempo de processamento em milissegundos dos algoritmos baseados no <i>CluStream</i> (Figura 5.35a) e no <i>StreamKM++</i> (Figura 5.35b) - base de dados artificial.	103
5.36 Tempo de processamento em milissegundos dos algoritmos baseados no <i>Stream LSearch</i> (Figura 5.36a) e dos algoritmos FEACS-SSI e FEACS-PHT (Figura 5.36b) - base de dados artificial.	104
5.37 Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos baseados no <i>CluStream</i> (CLS-BkM, CLS-IBkM, CLS-MEO k e CLS-MEO k I) para a base de dados KDDCup99.	108
5.38 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (ssi), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos CLS-BkM (Figura 5.38a) e CLS-IBkM (Figura 5.38b).	109
5.39 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (ssi), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos CLS-MEO k I (Figura 5.39a) e CLS-MEO k (Figura 5.39b).	110
5.40 Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos baseados no <i>StreamKM++</i> (SKM-BkM, SKM-IBkM, SKM-MEO k e SKM-MEO k I) para a base de dados KDDCup99.	111
5.41 Tempo de processamento em milissegundos de cada componente (abstração de dados, silhueta simplificada incremental (ssi), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) para os algoritmos SKM-BkM (Figura 5.41a) e SKM-IBkM (Figura 5.41b).	112
5.42 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (ssi), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos SKM-MEO k (Figura 5.42a) e SKM-MEO k I (Figura 5.42b).	113
5.43 Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos baseados no <i>Stream LSearch</i> (SLS-BkM, SLS-IBkM, SLS-MEO k e SLS-MEO k I) - Base de dados KDDCup99.	114
5.44 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (ssi), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) para os algoritmos SLS-BkM (Figura 5.44a) e SLS-IBkM (Figura 5.44b)	115
5.45 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (ssi), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos SLS-MEO k (Figura 5.45a) e SLS-MEO k I (Figura 5.45b).	116
5.46 Número de grupos obtidos pelo algoritmo SLS-MEO k - Base de dados KDD-Cup99.	117
5.47 Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos FEACS-PHT e FEACS-SSI para a base de dados KDDCup99.	117

5.48 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (silhueta simplificada incremental (SSI), atualização incremental combinada com o PHT e estimativa de \hat{k}) e sua soma (total) dos algoritmos FEACS-SSI 5.48a e FEACS-PHT 5.48b.	118
5.49 Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos baseados no <i>CluStream</i> (CLS-BkM, CLS-IBkM, CLS-MEO k e CLS-MEO k I) para a base de dados <i>Forest CoverType</i>	122
5.50 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos CLS-BkM (Figura 5.50a) e CLS-IBkM (Figura 5.50b) - Base de dados <i>Forest CoverType</i>	123
5.51 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos CLS-MEO k I (Figura 5.51a) e CLS-MEO k (Figura 5.51b) - Base de dados <i>Forest CoverType</i>	124
5.52 Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos baseados no StreamKM++ (SKM-BkM, SKM-IBkM, SKM-MEO k e SKM-MEO k I) para a base de dados <i>Forest CoverType</i>	125
5.53 Tempo de processamento em milissegundos de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) para os algoritmos SKM-BkM (Figura 5.53a) e SKM-IBkM (Figura 5.53b) - Base de dados <i>Forest CoverType</i>	125
5.54 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos SKM-MEO k (Figura 5.54a) e SKM-MEO k I (Figura 5.54b) - Base de dados <i>Forest CoverType</i>	126
5.55 Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos baseados no Stream LSearch (SLS-BkM, SLS-IBkM, SLS-MEO k e SLS-MEO k I) - Base de dados <i>Forest CoverType</i>	127
5.56 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) para os algoritmos SLS-BkM (Figura 5.56a) e SLS-IBkM (Figura 5.56b).	128
5.57 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos SLS-MEO k (Figura 5.57a) e SLS-MEO k I (Figura 5.57b).	129
5.59 Tempo de processamento em milissegundos (escala logarítmica) de cada componente (silhueta simplificada incremental (SSI), atualização incremental combinada com o PHT e estimativa de \hat{k}) e sua soma (total) dos algoritmos FEACS-SSI 5.59a e FEACS-PHT 5.59b - Base de dados <i>Forest CoverType</i>	130
5.58 Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos FEACS-PHT e FEACS-SSI para a base de dados <i>Forest CoverType</i>	131

Lista de Tabelas

2.1	Algoritmos que utilizam vetores de características para representar o FCD.	17
2.2	Algoritmos que utilizam vetores de protótipos para representar o FCD.	18
2.3	Algoritmos que utilizam conceitos de malha para representar o FCD.	22
5.1	Valores médios de IRA - Base de Dados Sintética.	69
5.2	Valores médios de tempo de processamento em milissegundos- Bases de Dados Artificiais. Em negrito estão destacados os menores valores para cada base de dados.	70
5.3	Taxas de acerto do número correto de grupos para os algoritmos baseados no CluStream - Bases de Dados Artificiais.	91
5.4	Taxas de acerto do número correto de grupos para os algoritmos baseados no StreamKM++ - Bases de Dados Artificias.	94
5.5	Taxas de acerto do número correto de grupos para os algoritmos baseados no Stream LSearch - Base de Dados Articiais.	97
5.6	Taxas de acerto do número correto de grupos para os algoritmos baseados no FEACS - Bases de Dados Articiais.	100
5.7	Valores médios de IRA e tempo de processamento - Base de Dados Sintética.	101
5.8	Diferença estatisticamente significativas para os tempos de processamentos nas bases de dados articiais. O símbolo ● indica que não foram encontradas diferenças significativas, + indica que os resultados para o algoritmo na linha <i>i</i> são superiores ao das colunas <i>j</i> e oposto é representado pelo símbolo —.	106
5.9	Valores de Silhueta Simplificada (SS) e tempo de processamento - Base de dados KDDCup99.	107
5.10	Análise de diferenças estatisticamente significativas para os valores de SS para a base de dados KDDCup99. O símbolo ● indica que não foram encontradas diferenças significativas, + indica que os resultados para o algoritmo na linha <i>i</i> são superiores ao das colunas <i>j</i> e oposto é representado pelo símbolo —.	119
5.11	Análise de diferenças estatisticamente significativas para os valores de tempo de processamento para a base de dados KDDCup99. O símbolo ● indica que não foram encontradas diferenças significativas, + indica que os resultados para o algoritmo na linha <i>i</i> são superiores ao das colunas <i>j</i> e oposto é representado pelo símbolo —.	120

5.12 Valores de Silhueta Simplificada (SS) e tempo de processamento - Base de dados <i>Forest Cover Type</i>	121
5.13 Análise de diferenças estatisticamente significativas para os valores de SS para a base de dados <i>Forest CoverType</i> . O símbolo ● indica que não foram encontradas diferenças significativas, + indica que os resultados para o algoritmo na linha <i>i</i> são superiores ao das colunas <i>j</i> e oposto é representado pelo símbolo —	131
5.14 Resultados dos testes de significância estatística para os valores de tempo de processamento para a base de dados <i>Forest CoverType</i> . O símbolo ● indica que não foram encontradas diferenças significativas, + indica que os resultados para o algoritmo na linha <i>i</i> são superiores ao das colunas <i>j</i> e oposto é representado pelo símbolo —	132
6.1 Componentes dos algoritmos estudados.	137

Lista de Siglas

MD	Mineração de Dados
AE	Algoritmo Evolutivo
AM	Aprendizagem de Máquina
IRA	Índice Rand Ajustado
CF	<i>Cluster Features</i>
TCF	<i>Temporal Cluster Features</i>
EHCF	<i>Exponential Histograms Cluster Features</i>
FEAC	<i>Fast Evolutionary Algorithm for Clustering</i>
FEACS	<i>Fast Evolutionary Algorithm for Clustering Data Streams</i>
FEACS-PHT	<i>Fast Evolutionary Algorithm for Clustering Data Stream based on Page Hinkley Test</i>
FEACS-ISS	<i>Fast Evolutionary Algorithm for Clustering Data Stream based on Incremental Simplified Silhouette</i>
FCD	Fluxo Contínuo de Dados
SS	Silhueta Simplificada
SSI	Silhueta Simplificada Incremental
SGDB	Sistemas de Gerenciamento de Bancos de Dados
BIRCH	<i>Balanced Iterative Reducing and Clustering Using Hierarchies</i>
PID	<i>Partition Incremental Discretization</i>
PHT	<i>Page-Hinkley Test</i>
FIFO	<i>first in, first out</i>
DBSCAN	<i>Density Based Spatial Clustering of Applications with Noise</i>

DGClust	<i>Distributed Grid Clustering</i>
MEO_k	Múltiplas Execuções Ordenadas do k -Médias
BkM	<i>Bisecting k-Means</i>
MEO_{kl}	Múltiplas Execuções Ordenadas do k -Médias Incremental
IBkM	<i>Incremental Bisecting k-Means</i>
SLS	Stream LSearch
CLS	CluStream
SKM	StreamKM++

Sumário

1	Introdução	1
1.1	Contextualização e Motivação	1
1.2	Hipótese de Pesquisa e Objetivos	3
1.3	Resumo das Contribuições	4
1.4	Organização	4
2	Agrupamento em Fluxos Contínuos de Dados	7
2.1	Abstração dos Dados	10
2.1.1	Estrutura de Dados	10
2.1.2	Modelos de Janelas	22
2.1.3	Agrupamento de Dados	26
2.2	Considerações Finais	29
3	Proposta de um Método para Estimação do Número de Grupos	31
3.1	Algoritmos Baseados nas k -Médias	31
3.1.1	<i>Stream LSearch</i>	32
3.1.2	<i>CluStream</i>	32
3.1.3	<i>StreamKM++</i>	33
3.2	Estimando o Número de Grupos	36
3.2.1	SS	36
3.3	Algoritmos para Gerar Partições de Dados	37
3.3.1	Múltiplas Execuções Ordenadas do k -Médias (MEO k)	38
3.3.2	<i>Bisecting k-Means</i> (BkM)	38
3.4	Proposta de um Método para Estimação de k	38
3.4.1	Adaptação para o <i>Stream LSearch</i>	40
3.4.2	Adaptação para o <i>CluStream</i>	40
3.4.3	Adaptação para o <i>StreamKM++</i>	40
3.5	Método Incremental para Estimação de k	41
3.5.1	SSI	41
3.5.2	Método Incremental	44
3.6	Considerações Finais	46
4	Algoritmos Evolutivos para o Agrupamento em FCD com Número de Grupos Variável	49
4.1	<i>Fast Evolutionary Algorithm for Clustering Data Streams</i> (FEACS)	50

4.1.1	FEACS-PHT	51
4.1.2	FEACS-ISS	51
4.1.3	Esquema de Codificação	53
4.1.4	Evoluindo Partições de Dados	54
4.1.5	Detecção de Mudanças	58
4.2	Considerações Finais	61
5	Avaliação Experimental	63
5.1	Bases de Dados	63
5.1.1	Bases de Dados Artificiais	63
5.1.2	Bases de Dados Reais	64
5.2	Configuração Experimental	66
5.2.1	Algoritmos	66
5.2.2	Avaliação do Agrupamento	68
5.3	Resultados	68
5.3.1	Bases de Dados Artificiais	68
5.3.2	Bases de Dados Reais	107
5.4	Considerações Finais	133
6	Conclusões	135
6.1	Contribuições	135
6.1.1	Publicações Geradas e Artigos Submetidos	138
6.2	Limitações e Trabalhos Futuros	138
R	Referências	140
A	Apêndice	149

Introdução

1.1 Contextualização e Motivação

Os avanços recentes em *hardware* e *software* têm proporcionado a aquisição de dados em grande escala. Estes podem ser gerados continuamente, em ambientes dinâmicos e em grandes velocidades, contrariamente ao que acontece com as bases de dados tradicionais. Fontes que geram dados nessa escala são chamadas de Fluxo Contínuo de Dados (FCD) ([Guha et al., 2000](#)). Dados que são gerados continuamente podem ser encontrados em diversas aplicações. Alguns dos exemplos típicos incluem ([Gama e Gaber, 2007](#)):

- Sistemas de segurança. Realizam o rastreamento de indivíduos em sequências de imagens para a identificação de intrusos ou suspeitos.
- Redes de sensores. Conjunto de sensores que são geograficamente distribuídos para extrair informações específicas. Por exemplo, redes de sensores podem ser encontradas em diversas aplicações tais como em agricultura de precisão (para obter informações sobre a temperatura, fertilidade do solo, entre outras) e redes elétricas (para obter informações sobre consumo ou previsão de apagões).
- Mercado Financeiro. Dados relacionados à bolsa de valores que necessitam ser analisados à medida que surgem para reportar os resultados que são relevantes aos investidores.
- Redes de Computadores. Análise do tráfego na internet em grandes velocidades, monitorar pacotes, filtrar os pacotes e se possível, detectar padrões não usuais da

rede e.g., detecção de IP de invasores.

Um fluxo contínuo de dados é uma sequência de objetos¹ (possivelmente ilimitada) que necessitam ser acessados na ordem que surgem e lidos apenas uma vez (ou um pequeno número de vezes) com limitações de armazenamento e de tempo de processamento (Gama, 2010). Estas restrições impossibilitam o uso de Sistemas de Gerenciamento de Bancos de Dados (SGDB) tradicionais para o armazenamento de cada objeto que surge em fluxo contínuo. Isso porque os tradicionais SGDBs não foram desenvolvidos para operar em dados gerados continuamente e de maneira rápida (Babcock et al., 2002). Dessa forma, torna-se imprescindível o desenvolvimento de ferramentas que possam lidar com dados gerados em fluxo contínuo para atender a demanda de aplicações que necessitam extrair informações de tais dados.

O desenvolvimento de métodos e algoritmos para a exploração do dados em busca de padrões válidos e potencialmente úteis é objeto de estudo do campo de pesquisa conhecido como Mineração de Dados (MD), do inglês *Data Mining* (Fayyad et al., 1996). Na mineração de dados, algoritmos de Aprendizagem de Máquina (AM) são tipicamente usados para a aquisição automática de conhecimento. AM estuda métodos computacionais que aprendem e aperfeiçoam o seu desempenho com a experiência (Mitchell, 1997). Métodos de AM têm sido utilizados em diversas aplicações, tais como, detecção de fraudes em transações bancárias (Lee et al., 2013), desenvolvimento de jogos (Hassan e Crandall, 2013) e análise de registros médicos (Bennett e Hauser, 2013).

Atualmente, a grande maioria das aplicações necessita de algoritmos de AM que transformem os dados que surgem continuamente em informações úteis para que seja possível a tomada de decisão em ambientes dinâmicos (Gama, 2010). Para essa finalidade, tais algoritmos necessitam incorporar novos dados de maneira incremental. Algoritmos de AM incrementais que são capazes de aprender continuamente com o decorrer do tempo desempenham um papel importante em tarefas de MD. Particularmente, FCD necessita de algoritmos de AM que podem evoluir modelos, eventualmente capazes de esquecer a fração dos dados que se tornam obsoletos (Gama, 2010). Além disso, para dados não estacionários, o conceito a ser aprendido, que envolve o fenômeno sob estudo, pode mudar com o tempo. Dessa maneira, uma propriedade desejável dos algoritmos incrementais é lidar com mudanças de conceitos. Neste contexto, algoritmos incrementais são de grande relevância para evitar tarefas computacionalmente custosas de treinar novamente todo o modelo, enquanto explicam padrões dinâmicos de dados que mudam com o tempo. Além disso, os algoritmos de AM incremental devem fazer uma simples-varredura (*single-pass*) sobre o FCD, uma vez que não é possível acessar todo o FCD mais de uma vez devido às limitações de armazenamento.

¹O termo objeto pode ser também compreendido como uma instância, uma tupla ou um registro de um conjunto de dados.

A partir da última década, algoritmos de agrupamento de dados têm sido utilizados para analisar FCD (Guha et al., 2000). A literatura sobre agrupamento de dados é vasta e continua em forte expansão. Em (Jain, 2009; Jain et al., 1999) são apresentadas algumas aplicações que têm utilizado técnicas de agrupamento, por exemplo, aplicações em segmentação de imagens e reconhecimento de objetos e caracteres. Dentre os algoritmos disponíveis na literatura, o algoritmo k -Médias é um dos mais populares para mineração de dados (Wu et al., 2007) por ser simples de implementar, escalável e por apresentar resultados promissores em diversas aplicações (Jain, 2009). Devido ao seu sucesso em diversas aplicações, um número de variantes do algoritmo k -Médias tem sido desenvolvido para lidar com FCD — e.g., (Ackermann et al., 2012; Agarwal et al., 2004; Aggarwal et al., 2003; Guha et al., 2000; O'Callaghan et al., 2002). Entretanto, tais variantes do k -Médias apresentam uma grande limitação no que diz respeito ao número de *clusters*², k , que necessita ser definido pelo usuário *a priori*. Dessa forma, técnicas que automaticamente identificam o número de grupos e acompanham a evolução dos grupos são necessárias. Embora existam métodos baseados em densidades que não necessitam definir *a priori* o número de grupos, por exemplo (Cao et al., 2006), estes métodos requerem a especificação de outros parâmetros (e.g., tamanho da vizinhança (raio) e o número mínimo de vizinhos para a inclusão em um grupo) que são difíceis de determinar na prática (Xu e Wunsch, 2009). Tais parâmetros capturam o número de grupos indiretamente e, neste sentido, fazem com que o usuário tenha de definir o número de grupos *a priori* e de maneira indireta, possivelmente levando à uma dificuldade ainda maior do que “simplesmente” escolher o número de grupos.

Uma difícil tarefa em agrupamento de dados envolve estimar o número ideal de grupos. (Liu, 1968) define que o número de possibilidades de atribuir N objetos em k grupos é dado por: $C(N,k) = \frac{1}{k!} \sum_{i=0}^k (-1)^i \binom{k}{i} (k-i)^N$. Dessa forma, existem, por exemplo, $C(25,5) = 2.436.684.974.110.751$ formas de agrupar 25 objetos em 5 grupos. Considerando casos em que o valor de k é desconhecido, como em várias aplicações de MD, o número total de maneiras de agrupar N objetos em k grupos é: $\sum_{k=1}^{k=N} C(N,k)$. De maneira formal, o problema de encontrar uma solução ótima para a separação de N objetos em k grupos é considerada NP-difícil (Falkenauer, 1998). Nestes cenários, uma estratégia de busca consiste em múltiplas execuções do algoritmo k -médias, variando o valor de k e gerando múltiplas partições iniciais, para então escolher a partição com k grupos que minimiza o erro quadrático (Jain, 2009). Porém, esta estratégia, desprovida de adaptações importantes que são objeto dessa tese, é usualmente inviável do ponto de vista computacional, principalmente em dados gerados em fluxo contínuo. O problema de encontrar o número de grupos pode ser visto como um problema de otimização. Neste caso, Algoritmos Evolutivos (AEs) têm sido amplamente utilizados para abordar tais problemas em

²Denomina-se aqui que um *cluster* é um grupo de dados, enquanto que um agrupamento (ou alternativamente, uma partição) é um conjunto de *clusters*.

bases de dados tradicionais ([Hruschka et al., 2009](#)).

1.2 Hipótese de Pesquisa e Objetivos

Esta tese aborda técnicas de agrupamento de dados em fluxos contínuos. De maneira específica, foram estudados algoritmos para agrupamento em FCD capazes de estimar o número de grupos. Tais algoritmos são muito pouco explorados na literatura ([Silva e Hruschka, 2011](#); [Silva et al., 2013](#)). Nesse contexto, pode-se formular a hipótese desta tese como:

É possível desenvolver algoritmos eficazes para agrupamento de dados que estimem o número de grupos (k) automaticamente sem comprometer significativamente a eficiência computacional do processo de agrupamento em ambientes dinâmicos com restrições de tempo e armazenamento encontrados em FCD.

No contexto desta hipótese de pesquisa, os principais objetivos deste trabalho são: (i) adaptar algoritmos que têm sido usados com sucesso em aplicações de FCD nas quais k é conhecido para cenários em que se deseja estimar o número de grupos; (ii) propor novos algoritmos para agrupamento que estimem k automaticamente a partir do FCD; (iii) avaliar sistematicamente, e de maneira quantitativa, os algoritmos propostos de acordo com as características específicas dos cenários de FCD.

1.3 Resumo das Contribuições

As contribuições dessa tese podem ser sumarizadas nos seguinte itens:

- Proposição de um método para estender algoritmos existentes na literatura no sentido de estimar automaticamente o número de grupos. No contexto desse método, foram propostos quatorze algoritmos, contribuindo significativamente para a área de agrupamento em FCD ([Silva e Hruschka, 2011, 2014](#)).
- Desenvolvimento de dois algoritmos evolutivos para o agrupamento em FCD capazes de estimar o número de grupos automaticamente a partir dos dados ([Silva et al., 2014](#)).
- Além dos algoritmos desenvolvidos, o extenso levantamento bibliográfico realizado durante a tese teve como resultado um *survey* da área de pesquisa em questão ([Silva et al., 2013](#)).

1.4 Organização

Essa tese está estruturada como segue:

Capítulo 2 - Agrupamento em Fluxo Contínuo de Dados: Durante a pesquisa bibliográfica sobre algoritmos de agrupamento em FCD foram identificados componentes comuns aos diversos algoritmos. Este capítulo aborda os diversos algoritmos da literatura sob o prisma de seus componentes comuns, explicitando as conexões e diferenças existentes entre diferentes abordagens. Os resultados desse estudo foram publicados em (Silva et al., 2013).

Capítulo 3 - Proposta de um Método para Estimação do Número de Grupos: Este capítulo descreve um método que estende alguns algoritmos da literatura para estimar o número de grupos. Em particular, foram escolhidos alguns algoritmos que representam o estado-da-arte dessa área do conhecimento. Os resultados dessa parte da pesquisa foram documentados em (Silva e Hruschka, 2011, 2014).

Capítulo 4 - Algoritmo Evolutivo (AE)s para o Agrupamento em FCD com Número de Grupos Variável:

Neste capítulo são apresentados dois novos algoritmos evolutivos para o agrupamento em FCD que estimam o número de grupos. Os resultados desse estudo foram documentados em (Silva et al., 2014).

Capítulo 5 - Experimentos: Uma completa análise experimental em cenários distintos nos quais os algoritmos desenvolvidos nessa tese podem ser aplicados são apresentados nesse capítulo.

Capítulo 6 - Conclusões e Trabalhos Futuros: Conclusões, oportunidades de trabalhos futuros e as publicações são descritas nesse capítulo.

Apêndice 1 - Protótipo de Sistema: Um protótipo de um sistema que incorpora os algoritmos desenvolvidos na tese foi proposto. Por meio desse protótipo de sistema, o usuário pode monitorar diversas informações importantes no processo de agrupamento (número de grupos, qualidade das partições, tempo de execução e memória consumida) em tempo real, bem como visualizar os dados sendo processados, tornando conveniente a análise de dados em aplicações reais de FCD.

Agrupamento em Fluxos Contínuos de Dados

Um Fluxo Contínuo de Dados (FCD), representado por \mathcal{X} , consiste de uma sequência de objetos $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$, ou seja, $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^N$, (potencialmente ilimitada, $N \rightarrow \infty$). Cada objeto é descrito por um vetor de valores de atributos (ou características) l -dimensional $\mathbf{x}_i = [x_i^j]_{j=1}^l$ pertencente a um espaço Ω que pode ser contínuo, categórico ou misto. Cada objeto também é associado a um marcador de tempo t_i que pode ser explícito (quando a fonte que gera os dados informa o tempo de criação do objeto) ou implícito (obtida pelo tempo de chegada do objeto no sistema) (Babcock et al., 2002). Além disso, a distribuição de probabilidades que gera os objetos pode mudar com o tempo (Gama, 2010). Formalmente, um FCD pode ser composto de vários subconjuntos $X^j \subset \mathcal{X}$, sendo que cada objeto do subconjunto X^j é gerado por alguma mistura de distribuições $D^j \in \mathcal{D}$, sendo \mathcal{D} uma família de mistura de distribuições de probabilidade.

Devido à grande quantidade de dados que surgem em fluxo contínuo e à sua natureza dinâmica, novas estruturas de dados e técnicas para mineração em FCD são necessárias (Han e Kamber, 2000). Mineração em FCD envolve duas questões principais (Aggarwal, 2007): i) processamento rápido e incremental e ii) lidar com mudanças na distribuição dos dados. A primeira questão refere-se ao processamento dos dados que deve ser realizado com uma simples-varredura, uma vez que não é possível armazenar o FCD para realizar uma análise completa nos dados. A segunda questão está relacionada com mudanças de conceitos que podem ocorrer com o tempo devido à evolução do fenômeno em estudo. Dessa maneira, FCD defronta com questões de larga escala e, assim, existe uma crescente

demandam por novos algoritmos de Aprendizagem de Máquina (AM) (e.g., para classificação e agrupamento de dados) que podem evoluir modelos, eventualmente capazes de esquecer (uma porção dos) dados que podem se tornar obsoletos. Nesta tese, o foco reside em problemas de agrupamento de dados.

O processo de agrupamento de dados envolve a identificação de um conjunto de categorias – também chamadas de grupos ou de *clusters* – que descrevam um conjunto de dados (Fayyad et al., 1996), objetivando-se maximizar a homogeneidade entre os objetos de um mesmo grupo e, concomitantemente, maximizar a heterogeneidade entre objetos de grupos distintos. Algoritmos de agrupamento podem ser classificados em duas principais categorias: hierárquicos e particionais (Jain e Dubes, 1988). Algoritmos hierárquicos representam os dados em vários níveis por meio de uma estrutura baseada em árvore, enquanto métodos particionais agrupam os dados em um único nível. Embora métodos hierárquicos (e.g., *single-linkage* e *average-linkage*) sejam conhecidos por frequentemente obter resultados de melhor qualidade, estes apresentam complexidade computacional pelo menos quadrática com o número de objetos. Por outro lado, algoritmos de agrupamento particionais apresentaram-se mais adequados para agrupamento em grandes bases de dados devido sua complexidade computacional relativamente baixa (Xu e Wunsch, 2009).

Nesta tese, são abordados métodos particionais para agrupamento de dados que envolvem a obtenção de partições rígidas (*crisp*) de dados em k *clusters*, i.e., cada objeto do conjunto de dados pertence a um único grupo da partição de dados. Uma partição rígida para um conjunto finito de objetos $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^N$, onde N é o tamanho do conjunto, é uma coleção $C = \{C_j\}_{j=1}^k$ de k subconjuntos de objetos sem sobreposição $C_i \neq \emptyset$ (grupos não vazios), tal que $\bigcup_{i=1}^k C_i = \mathcal{X}$ e $C_i \cap C_j = \emptyset$ para $i \neq j$.

Em bases de dados estáticas, algoritmos particionais típicos incluem k -Médias, k -Medóides e suas variantes (Han e Kamber, 2000). O algoritmo de agrupamento baseado nas k -Médias é o mais utilizado para minerar dados (Wu et al., 2007) por ser simples de programar, escalável e por apresentar resultados promissores (Jain, 2009). Porém, em grandes volumes de dados, a aplicação de algoritmos de agrupamento tradicionais torna-se inviável devido às limitações de armazenamento e tempo, tornando ainda mais desafiadora a tarefa de agrupamento.

Agrupamento em FCD necessita de algoritmos capazes de continuamente agrupar os objetos, respeitando as restrições de tempo e memória. Tendo em mente essas restrições, algoritmos para o agrupamento em FCD devem apresentar, idealmente, as seguintes propriedades (Silva et al., 2013): i) prover os resultados em tempo viável, realizando processamento de maneira rápida e incremental; ii) adaptar-se rapidamente à dinâmica dos dados, i.e., os algoritmos devem detectar quando novos grupos aparecem ou desaparecem; iii) ser escalável ao número de objetos que surgem continuamente; iv) prover um modelo que não seja apenas compacto, mas que também o seu tamanho não cresça com o número de objetos processados (note que mesmo um crescimento linear normalmente

não é tolerado); v) rapidamente detectar *outliers* e reagir de forma adequada; e vi) lidar com diferentes tipos de dados, por exemplo, árvores XML, sequências de DNA e informações temporais e espaciais de GPS. Embora estes requerimentos sejam parcialmente considerados na prática, é instrutivo ter em mente essas propriedades ao desenvolver algoritmos para agrupamento em FCD.

Em geral, o problema de agrupamento em FCD é definido de tal forma a manter continuamente um agrupamento consistente dos objetos processados usando uma pequena quantidade de memória. Dessa maneira, vários algoritmos de agrupamento em FCD têm sido propostos na literatura (Silva et al., 2013). Estes algoritmos podem ser summarizados via duas etapas principais: i) etapa de abstração dos dados (também conhecida como componente *online*) e ii) etapa de agrupamento (também conhecida como componente *offline*), conforme ilustrado na Figura 2.1.

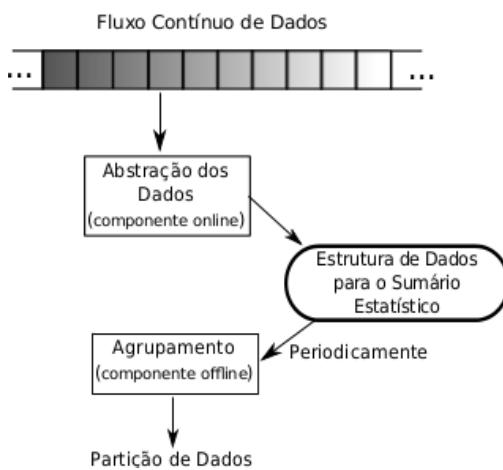


Figura 2.1: Estrutura de trabalho (framework) para os algoritmos de agrupamento em FCD.

A etapa de abstração de dados sumariza o FCD com o auxílio de estruturas de dados específicas para lidar com as restrições de memória em aplicações de FCD. Estas estruturas sumarizam os dados do fluxo de maneira a preservar o significado dos objetos processados sem a necessidade de armazená-los. Existem diversas estruturas de dados desenvolvidas com este propósito (Silva et al., 2013). Por exemplo, pode-se citar as seguintes: vetor de características, vetor de protótipos, árvore de conjunto de objetos representativos e malha. Na Seção 2.1 serão fornecidos os detalhes da etapa de abstração e as estruturas de dados utilizadas. Para sumarizar os objetos que surgem continuamente no fluxo de dados e, ao mesmo tempo, dar mais importância aos objetos mais recentes, um método popular consiste na definição de janelas temporais. Entre os diversos modelos de janelas propostos na literatura, os modelos mais comuns são janelas deslizantes, janelas amortizadas e janelas de ponto de referência (Silva et al., 2013).

Após realizar a etapa de abstração dos dados, algoritmos de agrupamento em FCD obtêm uma partição dos dados via etapa de agrupamento (componente *offline*). A componente *offline* é utilizada em conjunção com uma variedade de parâmetros de entrada (e.g.,

horizonte temporal e o número de grupos) para fornecer aos usuários um rápido entendimento dos grupos quando for solicitada. Embora esta etapa necessite dos sumários estatísticos como entrada, a sua execução é muito eficiente na prática. Baseando-se nessa suposição, algoritmos de agrupamento tradicionais (por exemplo, *Density Based Spatial Clustering of Applications with Noise* (DBSCAN) (Ester et al., 1996) e *k*-Médias (MacQueen, 1967)) podem ser utilizados para encontrar uma partição de dados nos sumários, cujo tamanho é relativamente menor ao ser comparado com o tamanho do FCD. A seguir são detalhadas cada uma dessas etapas dos algoritmos de agrupamento em FCD.

2.1 Abstração dos Dados

A abstração de dados pode ser considerada como um pré-agrupamento do FCD. Nesse sentido, Nesta Seção serão apresentados importantes aspectos envolvidos na etapa de abstração dos dados, tais como estrutura de dados, modelos de janelas e mecanismos de detecção de *outliers*.

2.1.1 Estrutura de Dados

O desenvolvimento de estruturas de dados adequadas para o armazenamento dos sumários estatísticos do FCD é um passo crucial para qualquer algoritmo de agrupamento em FCD, principalmente devido às restrições impostas em aplicações de FCD. Considerando que o FCD não pode ser armazenado na memória principal, estruturas de dados bem projetadas devem ser empregadas para a summarização incremental do FCD. A seguir são apresentados quatro tipos de estruturas de dados tipicamente utilizadas na etapa de abstração dos dados: i) vetor de características; ii) vetor de protótipos; iii) árvore de conjunto de objetos representativos e iv) malha.

Vetor de Características

O uso de vetores de características para sumarizar grandes quantidades de dados foi introduzido no algoritmo *Balanced Iterative Reducing and Clustering Using Hierarchies* (BIRCH) (Zhang et al., 1996). Este vetor, denominado *Cluster Features* (CF), tem três componentes: número de objetos n , soma linear dos vetores de objetos $\mathbf{ls} = \sum_{i=1}^n \mathbf{x}_i$ e sua soma quadrática $\mathbf{ss} = \sum_{i=1}^n \mathbf{x}_i^\top \mathbf{x}_i$ (assumindo \mathbf{x}_i um vetor coluna). Os componentes \mathbf{ls} e \mathbf{ss} são l dimensionais. Esses três componentes permitem calcular medidas de cada grupo de maneira incremental, tais como o vetor médio (Equação 2.1), o raio (Equação 2.2) e o diâmetro (Equação 2.4).

$$\boldsymbol{\mu} = \frac{\mathbf{ls}}{n} \quad (2.1)$$

$$\text{raio} = \sqrt{\frac{\mathbf{ss}}{n} - \left(\frac{\mathbf{ls}}{n}\right)^T \left(\frac{\mathbf{ls}}{n}\right)} \quad (2.2)$$

A Equação 2.2 referente ao raio de um grupo é uma expansão da equação do desvio padrão para o caso multivariado, conforme se pode observar a partir do desenvolvimento da Equação 2.3:

$$\begin{aligned} \sigma &= \left(\frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \boldsymbol{\mu})^T (\mathbf{x}_i - \boldsymbol{\mu}) \right)^{\frac{1}{2}} \\ &= \left(\frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i^T \mathbf{x}_i - 2\mathbf{x}_i^T \boldsymbol{\mu} + \boldsymbol{\mu}^T \boldsymbol{\mu}) \right)^{\frac{1}{2}} \\ &= \left(\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i^T \mathbf{x}_i - 2 \left(\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \right)^T \boldsymbol{\mu} + \boldsymbol{\mu}^T \boldsymbol{\mu} \right)^{\frac{1}{2}} \\ &= \left(\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i^T \mathbf{x}_i - 2(\boldsymbol{\mu}^T \boldsymbol{\mu}) + \boldsymbol{\mu}^T \boldsymbol{\mu} \right)^{\frac{1}{2}} \\ &= \left(\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i^T \mathbf{x}_i - \boldsymbol{\mu}^T \boldsymbol{\mu} \right)^{\frac{1}{2}} \\ &= \sqrt{\frac{\mathbf{ss}}{n} - \left(\frac{\mathbf{ls}}{n}\right)^T \left(\frac{\mathbf{ls}}{n}\right)} \end{aligned} \quad (2.3)$$

Analogamente, a Equação 2.4, que captura o diâmetro do grupo, é uma expansão da equação da distância média intragrupo, conforme se pode observar do desenvolvimento da Equação 2.5.

$$\text{diâmetro} = \sqrt{\left(\frac{2n\mathbf{ss} - 2\mathbf{ls}^T \mathbf{ls}}{n(n-1)} \right)} \quad (2.4)$$

$$\begin{aligned}
\Delta &= \left(\frac{1}{n(n-1)} \sum_{i=1}^n \sum_{j=1}^n (\mathbf{x}_i - \mathbf{x}_j)^2 \right)^{\frac{1}{2}} \\
&= \left(\frac{1}{n(n-1)} \sum_{i=1}^n \sum_{j=1}^n (\mathbf{x}_i^\top \mathbf{x}_i - 2\mathbf{x}_i^\top \mathbf{x}_j + \mathbf{x}_j^\top \mathbf{x}_j) \right)^{\frac{1}{2}} \\
&= \left(\frac{1}{n(n-1)} \left(\sum_{i=1}^n \sum_{j=1}^n \mathbf{x}_i^\top \mathbf{x}_i - 2 \sum_{i=1}^n \sum_{j=1}^n \mathbf{x}_i^\top \mathbf{x}_j + \sum_{i=1}^n \sum_{j=1}^n \mathbf{x}_j^\top \mathbf{x}_j \right) \right)^{\frac{1}{2}} \\
&= \left(\frac{1}{n(n-1)} \left(n \sum_{i=1}^n \mathbf{x}_i^\top \mathbf{x}_i - 2 \left(\sum_{i=1}^n \mathbf{x}_i \right)^\top \left(\sum_{j=1}^n \mathbf{x}_j \right) + n \sum_{j=1}^n \mathbf{x}_j^\top \mathbf{x}_j \right) \right)^{\frac{1}{2}} \\
&= \left(\frac{(nss - 2ls^\top ls + nss)}{n(n-1)} \right)^{\frac{1}{2}} \\
&= \sqrt{\frac{(2nss - 2ls^\top ls)}{n(n-1)}}
\end{aligned} \tag{2.5}$$

Os componentes do CF são estatísticas suficientes para modelos de mistura de gaussianas, embora [Zhang et al. \(1996\)](#) não façam nenhuma suposição sobre os dados. O vetor de CF apresenta também importantes propriedades que são úteis em um cenário de FCD, tais como:

1. Incrementalidade: Um objeto \mathbf{x}_i pode ser facilmente inserido (ou removido) no CF apenas com a atualização do seu sumário estatístico da seguinte maneira:

$$\begin{aligned}
ls &= ls + \mathbf{x}_i \\
ss &= ss + \mathbf{x}_i^\top \mathbf{x}_i \\
n &= n + 1
\end{aligned}$$

2. Aditividade: Dois vetores CF disjuntos podem ser facilmente combinados formando um terceiro CF por meio da soma de seus componentes.

$$\begin{aligned}
n_3 &= n_1 + n_2 \\
ls_3 &= ls_1 + ls_2 \\
ss_3 &= ss_1 + ss_2
\end{aligned}$$

Além dos vetores CF, uma estrutura de dados baseada em árvore denominada CF-Tree é empregada no BIRCH ([Zhang et al., 1996](#)) para a organização dos CFs. Especificamente, é utilizada a árvore B+, na qual cada nó não-folha da árvore contém no máximo B entradas, uma para cada CF com seu respectivo ponteiro para um nó folha. Um nó folha contém

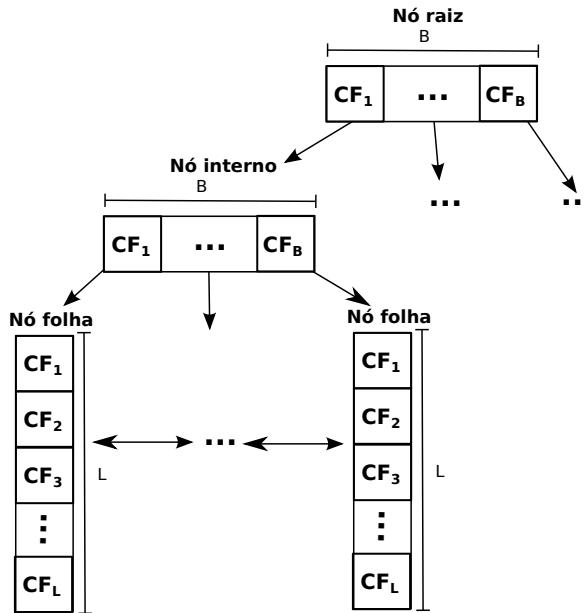


Figura 2.2: Estrutura de dados CF-Tree.

no máximo L entradas. Na Figura 2.2 é ilustrada uma CF-Tree, na qual cada nó não-folha representa um grupo consistindo de subgrupos (seus nós filhos) e os nós folhas representam grupos formados pela soma de suas entradas.

O vetor CF do BIRCH foi empregado por diferentes algoritmos da literatura. O algoritmo *Scalable k-Means* (Bradley et al., 1998), por exemplo, utiliza o vetor CF para aumentar a escalabilidade do algoritmo k -Médias para grandes bases de dados. A ideia básica é que os objetos do conjunto de dados não são igualmente importantes para o agrupamento de dados. A partir dessa observação, *Scalable k-Means* emprega diferentes mecanismos para identificar objetos que não necessitam ser armazenados na memória. Este algoritmo armazena os objetos em um bloco (também denominado *buffer*) na memória. Por meio dos vetores CF, os objetos são sumarizados e mantidos no bloco e os objetos representados por esses vetores CF são descartados para liberar espaço no bloco. O tamanho do bloco é determinado pelo usuário. Quando o bloco está cheio, uma variante do k -Médias é executada nos objetos. Essa variante, denominada *Extended k-Means*, pode lidar tanto com objetos quanto com vetores CF. Baseada na primeira partição de dados gerada, duas fases de compressão são aplicadas nos objetos que surgem continuamente e são armazenados no bloco.

A primeira fase de compressão consiste em encontrar e descartar os objetos que provavelmente não modificarão a sua pertinência aos grupos em futuras iterações do *Scalable k-Means*. Para detectar esses objetos, dois métodos são empregados: PDC_1 e PDC_2 . O método PDC_1 encontra uma porcentagem de objetos (parâmetro determinado pelo usuário) que estão dentro do raio de Mahalanobis de um grupo e então comprime-os via CF. Em seguida, os objetos comprimidos são descartados. PDC_2 é aplicado nos objetos que não foram comprimidos pelo PDC_1 (objetos que estão fora do raio do seu grupo). A ideia

do segundo método é considerar o pior caso que consiste no deslocamento dos grupos (aplicar uma perturbação no centroide¹) e verificar se a pertinência dos objetos aos seus respectivos grupos é ainda mantida. Portanto, para cada objeto restante, \mathbf{x}_i , PDC_2 encontra o centroide do grupo mais próximo (de acordo com a distância de Mahalanobis). Considere que este seja o centroide do grupo C_i . Em seguida, o centroide de C_i é perturbado, especificamente deslocando-se o centroide de C_i o mais distante possível de \mathbf{x}_i (dentro de um intervalo de confiança calculado). Além disso, o centroide do segundo grupo mais próximo (C_j) é deslocado para estar mais próximo possível de \mathbf{x}_i (respeitando o intervalo de confiança). Se \mathbf{x}_i continua mais próximo de C_i , o CF correspondente do grupo C_i é atualizado com a inserção de \mathbf{x}_i e o objeto é descartado. Caso contrário, \mathbf{x}_i é mantido na memória para a segunda fase de compressão.

A segunda fase de compressão é aplicada nos objetos que não foram descartados na compressão primária. O objetivo dessa fase é liberar o espaço na memória para o armazenamento de novos objetos. Esses objetos que não foram descartados são agrupados pelo k -Médias em k_2 grupos, cujo valor de k_2 é determinado pelo usuário. Cada um dos k_2 grupos (representados pelo vetor CF) é avaliado de acordo com um critério de compactação que verifica quando a variância de um grupo está abaixo de um limiar β (parâmetro de entrada). Os grupos CF que atendem esse critério são combinados com os CFs obtidos na compressão primária. Os vetores CF que não atendem esse critério de compactação são descartados da memória (Bradley et al., 1998).

Uma simplificação do *Scalable k-Means*, apresentada em (Farnstrom et al., 2000), resulta no algoritmo denominado *Single-Pass k-Means*. Com o objetivo de reduzir o custo computacional, este algoritmo não emprega as fases de compressão do *Scalable k-Means*. Ao invés disso, em cada iteração do algoritmo todos os objetos na memória são descartados após criar os sumários estatísticos (CF) mantidos na memória.

O conceito de CF foi estendido e denominado microgrupo no algoritmo *CluStream* (Aggarwal et al., 2003). Cada microgrupo tem cinco componentes. Três deles são os existentes no CF (n , \mathbf{l}_s e \mathbf{s}_s). Os componentes adicionais são a soma dos marcadores de tempo (\mathbf{l}_{st}) e a soma dos quadrados dos marcadores de tempo (s_{st}). Este algoritmo realiza o processo de agrupamento em duas fases: i) fase *online* que periodicamente armazena um conjunto de microgrupos, e ii) a fase *offline* que utiliza os microgrupos para gerar macrogrupos a partir de alguns parâmetros definidos pelo usuário. A fase *online* armazena q microgrupos na memória, onde q é um parâmetro de entrada. A Figura 2.3 apresenta a estrutura do *CluStream*.

Cada microgrupo tem um limite máximo de raio — Equação 2.2 — multiplicado por um fator f . Para cada novo objeto, o microgrupo mais próximo (de acordo com a distância Euclidiana) é selecionado para absorvê-lo. Para decidir quando um microgrupo deve absorver um novo objeto, é verificado se a distância entre o novo objeto e o centroide mais próximo

¹Centroide é o vetor médio de um grupo.

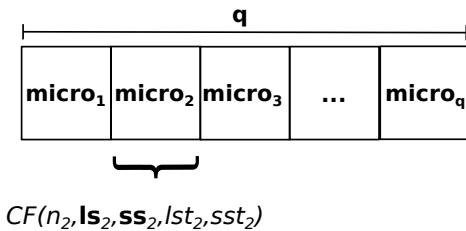


Figura 2.3: Estrutura de microgrupo usada no algoritmo *CluStream* (Aggarwal et al., 2003).

está dentro do limite máximo. Caso esteja, o objeto é incorporado no microgrupo e o seu sumário estatístico é atualizado. Se nenhum dos microgrupos pode absorver o objeto, um novo microgrupo é criado.

Para criar um microgrupo, um microgrupo mais antigo deve ser removido, ou dois microgrupos devem ser unidos. O microgrupo mais antigo é removido se seu marcador de tempo está abaixo do limiar δ (parâmetro de entrada), o qual é considerado um *outlier* e, portanto, deve ser removido. Dessa maneira, o algoritmo *CluStream* calcula uma estimativa de relevância média desse microgrupo (também conhecida como *relevance time*). Esta estimativa é calculada para o $o/(2n_i)^{th}$ percentil dos n_i objetos no microgrupo i , assumindo-se que os valores do marcador de tempo seguem uma distribuição normal (sendo o um parâmetro de entrada). Caso contrário, dois microgrupos são unidos, utilizando a propriedade aditiva dos vetores CF, o qual tem o custo computacional de $O(q^2)$.

Os q microgrupos (denominados de *snapshots*) são armazenados em um dispositivo secundário em intervalos de tempo que decaem exponencialmente, α^d , onde α e d são parâmetros cujos valores são definidos pelo usuário. Esses *snapshots* permitem que o usuário realize o agrupamento de dados em diferentes horizontes temporais, h , por meio do conceito de janelas piramidais (*pyramidal time window*) (Aggarwal et al., 2003).

Similarmente ao *CluStream*, Zhou et al. (2008) propuseram o algoritmo *SWClustering* que utiliza uma variante temporal do CF (conhecido como *Temporal Cluster Features* (TCF)). O TCF contém os três componentes originais do vetor CF e o marcador de tempo t que registra o tempo do último objeto inserido em um TCF. O algoritmo *SWClustering* também tem uma nova estrutura de dados denominada *Exponential Histograms Cluster Features* (EHCF), a qual é definida como uma coleção de TCFs. A Figura 2.4 apresenta as estruturas do *SWClustering*. Cada EHCF é distribuído em níveis que contém no máximo $\frac{1}{\phi} + 1$ TCFs, sendo $0 < \phi < 1$ um parâmetro definido pelo usuário. O número de objetos representados em um TCF_i é igual ou duas vezes maior que o número de objetos em um TCF_j , para $i > j$. Inicialmente, o primeiro TCF contém apenas um objeto. O centroide de um EHCF é calculado pela média dos ls de todos os seus TCFs. Quando um novo objeto x_i surge, o EHCF mais próximo é selecionado de acordo com a distância Euclidiana entre x_i e o centroide do EHCF. Se o EHCF mais próximo pode absorver o objeto x_i , i.e., sua distância para o objeto x_i está abaixo de $R \times \beta$, onde R é o raio desse EHCF e β é um limiar para tolerância do raio ($\beta > 0$), então x_i é inserido no EHCF. Caso contrário, um novo EHCF é

criado. Nesse caso, é necessário verificar se o número máximo de EHCF (capturado por meio de NC , que é um parâmetro definido pelo usuário) permitido foi alcançado. Se essa condição for atendida, dois EHCF vizinhos são unidos. Em seguida, os objetos expirados de um EHCF são removidos, restando apenas os z (tamanho da janela de dados) mais recentes.

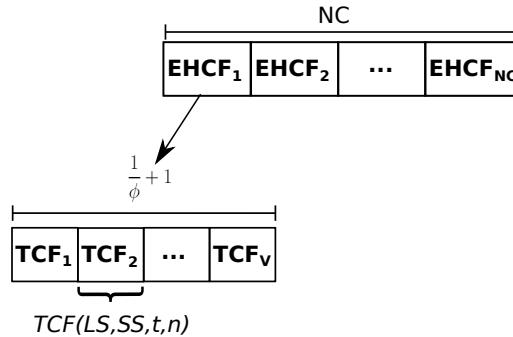


Figura 2.4: Estruturas de TCF e EHCF usadas no algoritmo *SWClustering* (Zhou et al., 2008).

DenStream (Cao et al., 2006) é um algoritmo de agrupamento para FCD baseado em densidade que também utiliza o conceito de CF do *Birch*. Em sua fase *online*, duas estruturas, denominadas de *p-microclusters* (microgrupos potenciais) e *o-microclusters* (microgrupos candidatos a *outliers*) são utilizadas para manter toda a informação necessária para o agrupamento dos dados. Cada uma dessas estruturas de microgrupos tem um peso associado que indica sua importância baseada no tempo (microgrupos com nenhum objeto recente tendem a perder sua importância *i.e.* seus respectivos pesos decrescem com o tempo tornando-os expirados). O peso de uma estrutura de *p-microclusters*, w , no tempo t é calculado de acordo com a Equação 4.4 e sua importância decai de acordo com a função de esquecimento descrita na Equação 2.7, parametrizada com λ , cujo valor é definido pelo usuário.

$$w(t) = \sum_{i \in p\text{-micro-cluster}} f(t - t_i) \quad (2.6)$$

$$f(u) = 2^{-\lambda u} \quad (2.7)$$

Além do peso, outras duas estatísticas são armazenadas para cada microgrupo: a soma dos pesos (wls) e a soma quadrática dos pesos (wss), calculadas de acordo com as Equações 2.8 e 2.9, respectivamente. Por meio dessas equações é possível calcular o raio r do microgrupo (Equação 2.10) e o seu centroide ($\frac{wls}{w}$). Além dessas estatísticas, apenas os *o-microclusters* mantêm armazenado o tempo de criação do microgrupo t_{init} que é utilizado para determinar a sua duração (tempo de vida).

$$\text{wls}(t) = \sum_{i \in p\text{-micro-cluster}} f(t - t_i) \mathbf{x}_i \quad (2.8)$$

$$\text{wss}(t) = \sum_{i \in p\text{-micro-cluster}} f(t - t_i) \mathbf{x}_i^\top \mathbf{x}_i \quad (2.9)$$

$$r(t) = \sqrt{\left(\frac{\text{wss}(t)}{w(t)} - \left(\frac{\text{wls}(t)}{w(t)} \right)^\top \left(\frac{\text{wls}(t)}{w(t)} \right) \right)} \quad (2.10)$$

Quando um objeto \mathbf{x}_i surge, o algoritmo *DenStream* tenta inseri-lo no *p-microcluster* mais próximo. A inserção será bem sucedida se o valor do seu raio estiver dentro de um limite ϵ (definido pelo usuário). Caso contrário, o algoritmo tentará inserir \mathbf{x}_i no *o-microcluster* mais próximo. Neste caso, a inserção será bem sucedida se o valor do seu raio também estiver dentro do limite ϵ . Além disso, se o peso do *o-microcluster* exceder o limiar $\beta \times \eta$, este microgrupo cresceu o suficiente para se tornar um *p-microcluster*, sendo β e η parâmetros de entrada. O parâmetro β controla o nível de tolerância dos microgrupos candidatos a *outlier* e η é o peso mínimo de um *p-microcluster*. Se \mathbf{x}_i não for absorvido pelo *o-microcluster* mais próximo, um novo *o-microcluster* é criado para absorver \mathbf{x}_i .

Em períodos de tempo delimitado t_p (Equação 2.11), o conjunto de *p-microclusters* é verificado para determinar se um *p-microcluster* deve ser tornar um *o-microcluster*. De maneira similar, um *o-microcluster* pode se tornar um *p-microcluster* após a análise dos seus pesos correspondentes. Se os valores dos parâmetros λ , β e η sugeridos pelos autores ([Cao et al., 2006](#)) forem utilizados, esta tarefa de verificação se tornará muito frequente, $t_p \leq 4$, levando a um custo computacional elevado.

$$t_p = \frac{1}{\lambda} \log \left(\frac{\beta\eta}{\beta\eta - 1} \right) \quad (2.11)$$

De maneira similar ao *DenStream*, o algoritmo *ClusTree* ([Kranen et al., 2011](#)) também usa CF ponderados, o qual é mantido em uma estrutura hierárquica (árvore da família R-tree). Dois parâmetros são usados para construir essa árvore: o número de entradas em um nó folha e o número de entradas em um nó não-folha. O *ClusTree* provê estratégias para lidar com restrições de tempo para realizar o agrupamento quando solicitado, *i.e.*, a possibilidade de interromper o processo de inserção de um novo objeto na árvore em qualquer momento. Este algoritmo não faz nenhuma suposição *a priori* do tamanho do agrupamento (número de microgrupos), uma vez que aplica operações de divisão e união para ajustar o tamanho do agrupamento automaticamente. Durante a caminhada na árvore, os objetos que não foram inseridos nos nós folhas devido a uma interrupção são temporariamente armazenados em um *buffer* no nó imediato à interrupção. Quando esse nó é acessado novamente, os objetos temporariamente armazenados no *buffer* são levados na caminhada da árvore como caroneiros (*hitchhiker*) e então a operação de inserção

desses objetos nos nós folhas continua. O algoritmo *ClusTree* pode também se adaptar à velocidade do FCD. Para FCD com alta velocidade de fluxo, o algoritmo *ClusTree* agrupa objetos similares para fazer uma rápida inserção na árvore. Em FCD com baixa velocidade, o tempo ocioso é utilizado para melhorar a qualidade do agrupamento.

Na Tabela 2.1 são apresentados os algoritmos descritos que utilizam vetores de características. Ao todo sete algoritmos adotam o conceito de vetores de características para representar os sumários do FCD.

Tabela 2.1: Algoritmos que utilizam vetores de características para representar o FCD.

Algoritmo	Referência
BIRCH	(Zhang et al., 1996)
Scalable K-Means	(Bradley et al., 1998)
Single Pass K-Means	(Farnstrom et al., 2000)
CluStream	(Aggarwal et al., 2003)
DenStream	(Cao et al., 2006)
SWClustering	(Zhou et al., 2008)
ClusTree	(Kranen et al., 2011)

Vetor de Protótipo

Os vetores de protótipos (Domingos e Hulten, 2001; Shah et al., 2005) summarizam os dados do fluxo apenas com os protótipos (por exemplo, centroides ou medóides) dos grupos.

Considerando o uso de vetores de protótipos, o algoritmo *Stream* (Guha et al., 2000) summariza o FCD por meio da estratégia de divisão e conquista. Especificamente, o algoritmo divide o FCD em blocos de tamanhos iguais $m = N^\rho$, $0 < \rho < 1$. Cada bloco contendo m objetos é summarizado em $2k$ objetos representativos utilizando uma variante do algoritmo k -Medóides (Kaufman e Rousseeuw, 1990) conhecido como localização de facilidades (*Facility Location*) (Charikar e Guha, 1999; Meyerson, 2001). O processo de summarizar a descrição dos objetos é repetido até que um vetor com m protótipos seja obtido. Em seguida, estes m protótipos são futuramente agrupados em $2k$ protótipos e o processo continua ao longo do fluxo, conforme ilustrado na Figura 2.5.

O algoritmo *Stream LSearch* (O'Callaghan et al., 2002) também usa uma estrutura para summarização baseada no vetor de protótipos. Este algoritmo assume que os objetos surgem em blocos X_1, X_2, \dots, X_z , sendo que cada bloco $X_i (i \in [1, z])$ pode ser agrupado na memória, assim produzindo k grupos. No i -ésimo bloco do FCD, o algoritmo mantém $O(i \times k)$ medóides. Entretanto, como $z \rightarrow \infty$ não é possível manter os $O(i \times k)$ medóides na memória. Portanto, quando a memória está cheia, o algoritmo *Stream LSearch* agrupa os $O(i \times k)$ medóides e mantém na memória apenas os k medóides obtidos desse processo.

Na Tabela 2.2 são apresentados os algoritmos descritos que utilizam conceitos de vetores de protótipos. Ao todo dois algoritmos adotam esse conceito para representar os sumários do FCD.

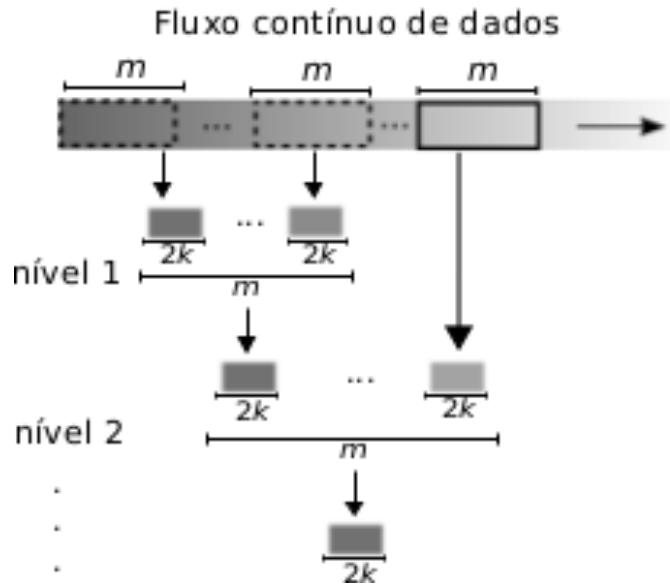


Figura 2.5: Vetores de protótipos utilizada no algoritmo *Stream* (Guha et al., 2000).

Tabela 2.2: Algoritmos que utilizam vetores de protótipos para representar o FCD.

Algoritmo	Referência
Stream	(Guha et al., 2000)
Stream Lsearch	(O'Callaghan et al., 2002)

Coreset Trees

Uma estrutura de dados significativamente diferente para agrupamento em FCD é a árvore de *coreset* utilizada no *StreamKM++* (Ackermann et al., 2012). Esta estrutura de dados é uma árvore binária, na qual cada nó i da árvore contém os seguintes componentes: um conjunto de objetos, E_i ; o protótipo de E_i , xp_i ; o número de objetos em E_i , n_i ; e a soma das distâncias quadráticas dos objetos em E_i para xp_i , sse_i . O conjunto E_i é armazenado apenas em um nó folha da árvore de *coreset*, porque os objetos de um nó interno são implicitamente definidos como a união dos objetos de seus nós filhos.

A estrutura de *coreset* é responsável por reduzir $2m$ objetos em m objetos. Primeiramente, a árvore tem apenas o nó raiz v , o qual contém todos os $2m$ objetos em E_v . O protótipo do nó raiz xp_v é escolhido aleatoriamente de E_v e $n_v = |E_v| = 2m$. A soma dos erros quadráticos, sse_v , é calculada considerando E_v e xp_v . Depois, dois nós folhas para v são criados (v_1 e v_2). Para criar esses nós, é necessário escolher um objeto de E_v com probabilidade proporcional a $\frac{Dist(\mathbf{x}_i, \mathbf{xp}_v)^2}{sse_v}$, $\forall \mathbf{x}_j \in E_v$, i.e., o objeto que é mais distante do protótipo xp_v tem alta probabilidade de ser selecionado. Este objeto selecionado será chamado de xq_v . O próximo passo é distribuir os objetos em E_v para E_{v1} e E_{v2} , de maneira que:

$$E_{v1} = \{\mathbf{x}_i \in E_v | Dist(\mathbf{x}_i, \mathbf{xp}_v) < Dist(\mathbf{x}_i, \mathbf{xq}_v)\} \quad \text{e} \quad E_{v2} = E_v \setminus E_{v1}. \quad (2.12)$$

Em seguida, os sumários estatísticos do nó folha v_1 são atualizados, i.e., $\mathbf{xp}_{v1} = \mathbf{xp}_v$, $n_{v1} = |\mathbf{E}_{v1}|$ e sse_{v1} é calculado conforme a definição de \mathbf{xp}_{v1} . Similarmente, os sumários estatísticos do nó filho v_2 são atualizados, mas note que $\mathbf{xp}_{v2} = \mathbf{xq}_v$. Este é o passo de expansão da árvore, o qual cria dois nós filhos para um dado nó interno. Quando a árvore tem muitos nós folhas, é necessário decidir qual deve ser primeiramente expandido. Para tal, é necessário iniciar do nó raiz e descer a árvore iterativamente selecionando um nó com probabilidade proporcional a $\frac{sse_{filho}}{sse_{pai}}$ até que um nó folha seja alcançado para o passo de expansão ser reiniciado. A expansão da árvore de *coreset* termina quando o número de nós folhas for igual a m .

StreamKM++ (Ackermann et al., 2012) é um algoritmo de duas etapas que realiza passos de redução e união (*merge-and-reduce*). A etapa de redução é realizada por meio da construção da árvore de *coreset*, considerando que sua construção reduz $2m$ objetos em m objetos. Já a etapa de união é realizada com outra estrutura de dados que contém um conjunto de L blocos, onde o valor de L é definido pelo usuário. Cada bloco pode armazenar m objetos. Quando surge um novo objeto, este é armazenado no primeiro bloco. Se o primeiro bloco está cheio, todos os seus objetos são movidos para o segundo bloco. Se o segundo bloco também está cheio, a etapa de união é realizada, i.e., os m objetos do primeiro bloco são unidos com os m objetos do segundo bloco, resultando em $2m$ objetos, que, por sua vez, são reduzidos por meio da construção da árvore de *coreset*. Os m objetos resultados são armazenados no terceiro bloco, ao menos que este esteja também cheio e então novamente é necessário realizar o procedimento *merge-and-reduce*. Este procedimento é ilustrado no pseudocódigo do Algoritmo 2.1.

Algoritmo 2.1: Inserção de um novo objeto no conjunto de blocos B (Ackermann et al., 2012). A função *coresetReduction*(\cdot) (linha 7) recebe $2m$ objetos e retorna m objetos summarizados.

Entrada: Novo objeto \mathbf{x}_i , conjunto de blocos $\mathbf{B} = \bigcup_{j=1}^L \mathbf{B}_j$, tamanho m do bloco.

Saída: Conjunto de blocos B atualizados.

```

1  $B_0 = B_0 \cup \{\mathbf{x}_i\};$ 
2 se  $|B_0| \geq m$  então
3    $Q = B_0;$ 
4    $B_0 = \emptyset;$ 
5    $j = 1;$ 
6   enquanto  $B_j \neq \emptyset$  faça
7      $Q = \text{coresetReduction}(B_j \cup Q);$ 
8      $B_j = \emptyset;$ 
9      $j = j + 1;$ 
10  fim
11   $B_j = Q;$ 
12   $Q = \emptyset;$ 
13 fim

```

Malha (*Grid*)

Outra forma de sumarizar os objetos do FCD é por meio de estruturas que organizam os dados em forma de malha (*grid*). Alguns algoritmos de agrupamento em FCD realizam a sumarização por meio de malhas (Cao et al., 2006; Chen e Tu, 2007; Gama et al., 2011; Park e Lee, 2007), i.e., particionam o espaço de característica l -dimensional em uma malha de células densas. Um algoritmo que utiliza essa estrutura de dados é o *D-Stream* (Chen e Tu, 2007). Este algoritmo mapeia cada objeto do fluxo em células densas. Cada objeto no tempo t está associado a um coeficiente de densidade que decresce com o tempo, conforme apresentado na Equação 2.13, onde o parâmetro $\lambda \in (0,1)$ é um fator de decaimento. A densidade de uma célula o no tempo t , $D(o,t)$, é dada pela soma das densidades ajustadas de cada objeto que está mapeado a o no tempo t (ou após o tempo t) ($E(o,t)$), conforme Equação 2.14. Cada célula da malha é representada por uma tupla $< t_o, t_m, D, label, status >$, onde t_o é o tempo que a última célula foi atualizada, t_m é o tempo em que a última célula foi removida da tabela *hash* que mantém células válidas, D é a densidade da célula, *label* é o rótulo da célula e *status* indica o estado da célula que pode ser *normal* ou *esporádica*, conforme será explicado a seguir.

$$D(\mathbf{x}_i, t) = \lambda^{t-t_i} \quad (2.13)$$

$$D(o,t) = \sum_{x \in E(o,t)} D(x,t) \quad (2.14)$$

A manutenção das células da malha é realizada durante a fase *online* do algoritmo. Uma célula pode se tornar esparsa se esta não recebe novos objetos por um longo período de tempo. Contrariamente, uma célula esparsa pode se tornar densa se a célula receber muitos objetos novos. Em intervalos de tempo determinados pelo parâmetro *gap*, as células são inspecionadas com relação ao seu estado. Considerando que o número de células pode ser grande, especialmente para FCD altamente dimensionais, somente as células que não estão vazias são armazenadas. Adicionalmente, células com poucos objetos são consideradas como *outliers* (*status* = *esporádico*). Células *esporádicas* são removidas periodicamente da lista de células válidas. Além disso, durante a fase *online*, quando um novo objeto l -dimensional surge do FCD, este é mapeado em sua correspondente célula o da malha. Se a célula o não está na lista de células válidas (estruturada como uma tabela *hash*), a célula é inserida e seu sumário estatístico correspondente é atualizado.

DGClust (Gama et al., 2011; Rodrigues et al., 2008) é um algoritmo para agrupamento distribuído de dados de sensores que também utiliza uma estrutura de malha para sumarizar o FCD. O algoritmo recebe dados de diferentes sensores, onde cada sensor produz um FCD univariado. Os dados são processados localmente em cada sensor e quando há uma atualização na célula da malha local (mudança de estado), esta atualização é comunicada

ao sítio global. O sítio local comunica a mudança de estado por meio do número de células que foram atualizadas. O sítio global é uma combinação de estados locais (células da malha) de cada sensor. Cada sítio local i mantém duas camadas de discretização com p_i e b_i blocos, respectivamente, onde $k < b_i < p_i$. O algoritmo *Partition Incremental Discretization* (PID) é utilizado para gerar os blocos para cada camada e assume que as células são de tamanhos iguais. Em cada instante de tempo t_i um objeto x_i é observado e o contador do correspondente bloco do valor lido é incrementado na primeira e segunda camada. O número de blocos na primeira camada pode mudar, dado que a seguinte condição é atendida: se o valor do contador associado ao bloco na primeira camada é maior que um limiar α (definido pelo usuário) o bloco é dividido em dois. A segunda camada discretiza os p_i blocos em b_i , i.e., esta discretização sumariza a informação da primeira camada em uma granularidade maior. O contador de blocos na segunda camada é incrementado quando o correspondente bloco na primeira camada é incrementado. Em seguida, a comunicação com o sítio global é realizada para enviar informações de atualização das células, de maneira que o estado das células do sítio global seja atualizado em cada instante de tempo. Caso ocorra uma divisão, todos os blocos da segunda camada são enviados para o sítio global; caso contrário apenas o bloco atualizado é enviado para o sítio global.

Na Tabela 2.3 são apresentados os algoritmos descritos que utilizam conceitos de malha. Ao todo dois algoritmos adotam esse conceito para representar os sumários do FCD.

Tabela 2.3: Algoritmos que utilizam conceitos de malha para representar o FCD.

Algoritmo	Referência
D-Stream	(Chen e Tu, 2007)
DGClust	(Gama et al., 2011)

2.1.2 Modelos de Janelas

Em muitos cenários de FCD, a informação mais recente do fluxo pode refletir o surgimento de uma tendência ou mudança na distribuição dos dados. Esta informação pode ser usada para explicar a evolução do processo em observação. Sistemas que dão a mesma importância para dados recentes e dados expirados não capturam as características de evolução FCD (Chen e Tu, 2007). Três modelos de janelas tipicamente estudados em FCD (Zhu e Shasha, 2002) são: i) modelos de janelas deslizantes, ii) modelos de janelas amortizada (*damped windows*) e iii) janelas de pontos de referência (*landmark windows*).

Janelas Deslizantes

No modelo de janelas deslizantes, apenas a informação mais recente do FCD é armazenada na estrutura de dados cujo tamanho pode ser variável ou fixo. Esta estrutura de dados é tipicamente do tipo *first in, first out* (FIFO), a qual considera os objetos de um pe-

ríodo atual do tempo ao invés dos objetos mais antigos. A organização e manipulação dos objetos nessa estrutura de dados são baseadas no princípio de processamento em fila, onde o primeiro elemento adicionado na fila será o primeiro elemento a ser removido. Na Figura 2.6 é apresentado um exemplo de janela deslizante.

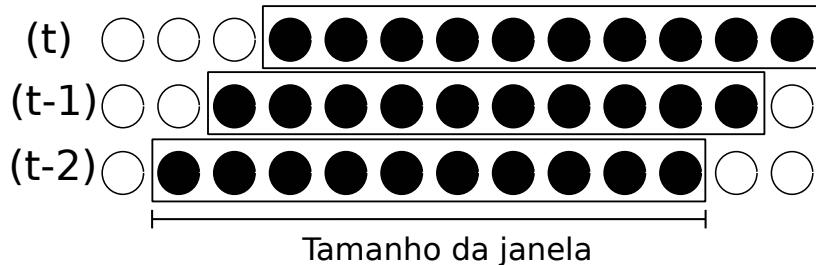


Figura 2.6: Modelo de janela deslizante.

O modelo de janela deslizante é utilizado por diversos algoritmos de agrupamento em FCD — por exemplo, (Babcock et al., 2003; Ren e Ma, 2009; Zhou et al., 2008). Basicamente, esses algoritmos apenas atualizam os sumários estatísticos dos objetos inseridos na janela. O tamanho da janela é definido de acordo com o recurso computacional disponível.

Janelas Amortizadas (*Damped Windows*)

Diferentemente do modelo de janela deslizante, as janelas amortizadas, também conhecidas como modelos de esquecimento temporal (*time-fading model*), consideram a informação mais recente por meio da associação de pesos aos objetos do FCD (Jiang e Gruenwald, 2006). Quanto mais recente for o objeto, maior será o valor do seu peso em relação aos objetos mais antigos. O peso do objeto decresce com o tempo por meio de uma função de decaimento. Um exemplo ilustrativo do modelo de janela amortizada é apresentado na Figura 2.7, onde o peso dos objetos decai exponencialmente — de preto (mais recente) para branco (expirado).

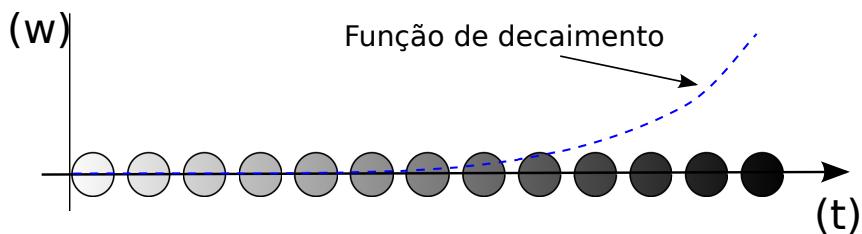


Figura 2.7: Modelo de janela amortizada.

O modelo de janelas amortizadas é tipicamente adotado em algoritmos de agrupamento baseados em densidades (Cao et al., 2006; Chen e Tu, 2007; Isaksson et al., 2012). Esses algoritmos usualmente assumem uma função de decaimento exponencial para os pesos dos objetos do FCD. Em (Cao et al., 2006), por exemplo, a função de decaimento

adotada é baseada numa função exponencial dada pela Equação 2.7, onde o parâmetro $\lambda > 0$ determina a taxa de decaimento e t é o tempo atual. Quanto mais alto o valor de λ , mais baixa é a importância dos objetos mais antigos comparativamente aos mais recentes. O algoritmo *D-Stream* (Chen e Tu, 2007) atribui um coeficiente de densidade para cada elemento que surge do FCD, cujo valor decresce com a idade do objeto. Esse coeficiente de densidade é dado por λ^{t-t_c} , onde t_c é o instante de tempo em que o objeto chegou do FCD.

Janelas por Ponto de Referência (*Landmark Window*)

O terceiro modelo de janela é o modelo de janelas por ponto de referência (*landmark window*). Processar o FCD por meio de janelas de ponto de referência requer lidar com porções disjuntas do fluxo denominados de blocos, os quais são separados por meio de ponto de referências (objetos relevantes). O ponto de referência pode ser definido tanto em termos de tempo (por exemplo, em dias ou semanas) ou em termos do número de elementos observados desde o último ponto de referência (Metwally et al., 2005). Todos os objetos que chegaram após o ponto de referência são mantidos ou summarizados em uma janela com os dados mais recentes. Quando um novo ponto de referência é alcançado, todos os objetos mantidos na janela até então são removidos e os novos objetos do atual ponto de referência são mantidos na janela até um novo ponto de referência ser encontrado. A Figura 2.8 ilustra um exemplo de janela de ponto de referência.

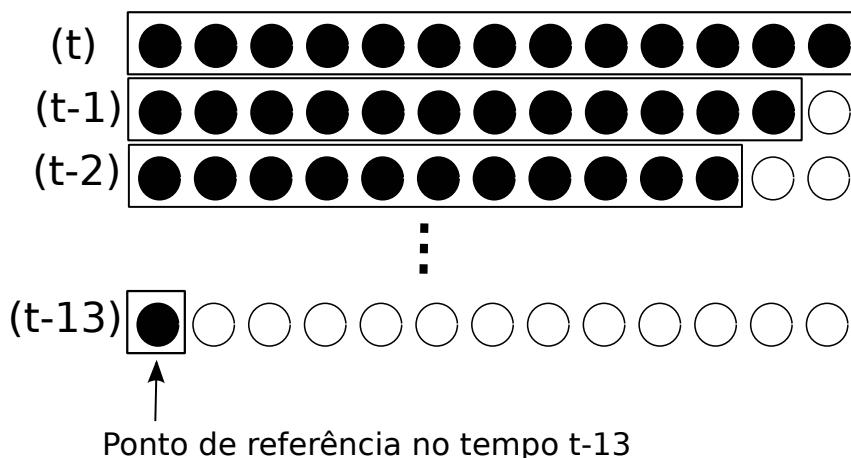


Figura 2.8: Modelo de janela de ponto de referência para um intervalo de tempo de tamanho 13.

Algoritmos de agrupamento em FCD que são baseados no uso de janelas de ponto de referência são encontrados em diversos trabalhos (Ackermann et al., 2012; Aggarwal et al., 2003; Bradley et al., 1998; Farnstrom et al., 2000; O'Callaghan et al., 2002). Em (O'Callaghan et al., 2002), por exemplo, o algoritmo *Stream* adota uma estratégia de divisão e conquista baseada em janelas de ponto de referência cujo ponto de referência é

definido a cada m número de objetos. Note que, neste tipo de modelo de janela, a relação entre os objetos de janelas vizinhas não é considerada.

O problema em utilizar qualquer esquema de janela de tamanho fixo está em encontrar o tamanho ideal de janela a ser empregada. Uma janela de tamanho pequeno garante que o algoritmo seja capaz de reagir rapidamente a eventuais mudanças de conceitos. Ao mesmo tempo, em fases instáveis do FCD, esse tamanho de janela pode afetar o desempenho do algoritmo. Contrariamente, uma janela de tamanho grande é desejável em fases estáveis, embora possa não responder rapidamente a mudança de conceitos (Gama et al., 2004).

Mecanismo de detecção de *outliers*

Além dos requerimentos de ser incremental e rápido, algoritmos de agrupamento em FCD devem também ser capazes de adequadamente lidar com *outliers* (Barbará, 2002). *Outliers* são objetos que desviam do comportamento geral do modelo de dados (Han e Kamber, 2000) e podem ocorrer devido a diferentes razões, tais como problemas na coleta dos dados, erros de armazenamento e transmissão, atividades fraudulentas ou mudança no comportamento do sistema.

Métodos baseados em densidades buscam por regiões de baixa densidade no espaço de entradas, e podem indicar a presença de *outliers*. Por exemplo, o algoritmo BIRCH (Zhang et al., 1996) tem uma fase opcional que faz uma varredura na árvore de CF e armazena os nós folhas com baixa densidade no disco. O número de *bytes* reservado para armazenar os *outliers* no disco é especificado pelo usuário. Os CFs com baixa densidade (estimado de acordo com um limiar) são considerados *outliers*. O valor desse limiar é determinado pelo tamanho médio dos CFs nos nós folhas. Periodicamente, o algoritmo verifica quando os CFs armazenados no disco podem ser inseridos na árvore de CF (mantida na memória). Este monitoramento ocorre quando a capacidade do disco chegou no seu limite ou quando todo o FCD foi processado (assumindo um FCD de tamanho finito). Potencialmente, essa fase opcional pode ser utilizada para monitorar mudanças na distribuição dos dados à medida que os dados são absorvidos pelos CF's.

O algoritmo *DenStream* (Cao et al., 2006) introduz o conceito de *buffer* de *outliers*. A fase online do *DenStream* mantém sumários estatísticos do FCD por meio de *p-microclusters*. A cada período de tempo T_p (Equação 2.11) a fase *online* do *DenStream* verifica os *p-microclusters* para identificar potenciais *outliers*, denominados de *o-microclusters*. Os *o-microclusters* são descritos pela tupla $\langle WLS, WSS, w, t_{init} \rangle$, onde t_{init} é o seu tempo de criação. Um *p-microcluster* torna-se um *o-microcluster* se o seu peso (w) está abaixo do limiar de *outlier* ($w < \beta\eta$), onde β e η são parâmetros definidos pelo usuário. Note que manter todos os *o-microclusters* na memória pode-se tornar inviável após algum tempo. Portanto, alguns *o-microclusters* necessitam ser removidos. A ideia é manter no *buffer* de *outliers* apenas os *o-microclusters* que podem se tornar *p-microclusters* (i.e., *o-microclusters*

cujo peso cresce com o tempo). Para remover os *outliers* reais, em cada instante de tempo t_p , o peso dos *o-microclusters* é verificado, e todos aqueles que estão abaixo do limite são removidos. O limite é calculado por meio da Equação 2.15, onde t é o tempo atual.

$$\xi(t, t_{\text{init}}) = \frac{2^{-\lambda(t-t_{\text{init}}+t_p)-1}}{2^{-\lambda t_p} - 1} \quad (2.15)$$

O algoritmo *D-Stream* (Chen e Tu, 2007) identifica e remove células com baixa densidade que são categorizadas como células esporádicas. Tais células esporádicas (candidatas a *outliers*) podem ocorrer por duas razões: i) células que têm recebido poucos objetos e ii) células populosas que têm sua densidade reduzida por um fator de decaimento. O objetivo é remover células esporádicas que ocorrem pela primeira razão. Uma tabela *hash* armazena a lista de células e o algoritmo verifica periodicamente quais células na tabela *hash* são esporádicas. Essa verificação é realizada a cada intervalo de tempo *gap*, conforme descrito na Equação 2.17. As células cuja densidade está abaixo de um dado limiar são rotuladas como esporádicas. Se no próximo intervalo de tempo *gap* as células continuam rotuladas como esporádicas, elas são removidas da tabela *hash*. O limiar para determinar células com baixa densidade ($D(o,t) < \pi(t_o,t)$) é calculado de acordo com a Equação 2.16, na qual t é o tempo atual, t_o é o tempo da última atualização ($t > t_o$), u_1 e u_2 são parâmetros definidos pelo usuário, G é o número de células e $\lambda \in (0,1)$ é uma constante denominada de fator de decaimento. Ao empregar os valores dos parâmetros sugeridos pelos autores (Chen e Tu, 2007), as células são inspecionadas a cada *gap* < 2 objetos, o que pode ser computacionalmente custoso uma vez que essa verificação na malha é muito frequente.

$$\pi(t_g, t) = \frac{u_1(1 - \lambda^{t-t_g+1})}{G(1 - \lambda)} \quad (2.16)$$

$$gap = \left\lfloor \log_{\lambda} \left(\max \left\{ \frac{u_1}{u_2}, \frac{N-u_2}{N-u_1} \right\} \right) \right\rfloor \quad (2.17)$$

2.1.3 Agrupamento de Dados

A etapa de agrupamento tipicamente envolve a aplicação de algoritmos tradicionais de agrupamento para encontrar grupos de dados nos sumários estatísticos gerados pela etapa de abstração de dados já discutida nas seções anteriores.

Um dos mais populares algoritmos para agrupamento de dados é o *k*-Médias (MacQueen, 1967) devido à sua simplicidade, escalabilidade e ao sucesso empírico em diversas aplicações (Wu et al., 2007). Não é de se surpreender que o algoritmo *k*-Médias e suas variantes sejam amplamente empregados em cenários de FCD (Ackermann et al., 2012; Aggarwal et al., 2003; Bradley et al., 1998; Farnstrom et al., 2000; O'Callaghan et al., 2002; Zhou et al., 2008).

Uma poderosa estrutura de dados utilizada no agrupamento em FCD é o uso de CFs (Gama, 2010), conforme discutido na Seção 2.1.1. Algumas variantes do k -Médias têm sido propostas para lidar com CFs. Zhang et al. (1997), por exemplo, sugerem três maneiras de adaptar o algoritmo k -Médias para lidar com os CFs:

1. Calcular o centroide de cada CF ($\frac{LS}{n}$) e considerar cada centroide como um objeto.
2. Usar a maneira 1. e ponderar cada objeto (centroides dos CFs) proporcionalmente a n , de maneira que os CFs com mais objetos tenham mais influencia no cálculo dos centroides realizado pelo k -Médias.
3. Aplicar o algoritmo de agrupamento diretamente nos CFs, uma vez que seus componentes mantêm estatísticas suficientes para calcular a maioria das distâncias requeridas e as métricas de qualidade.

A primeira e a terceira estratégia são tipicamente adotadas pelos algoritmos de agrupamento baseados nos CFs. A primeira estratégia é simplista e utilizada na prática, desde que nenhuma modificação é necessária no algoritmo de agrupamento. Essa estratégia é adotada pelo algoritmo *ClusTree* (Kranen et al., 2011) para agrupar nós folhas (CFs) para produzir k grupos. A terceira estratégia requer modificações no algoritmo de agrupamento para lidar adequadamente com os CF's. No algoritmo *CluStream* (Aggarwal et al., 2003), as variantes do k -Médias utilizam uma versão adaptada da segunda estratégia que escolhe os protótipos iniciais com probabilidade proporcional ao número de objetos no microgrupo (CF expandido). Essa variante é apresentada no Algoritmo 2.2.

Algoritmo 2.2: Variante do k -Médias para lidar com sumários estatísticos (CFs) (Aggarwal et al., 2003).

Entrada: Número de grupos k e conjunto de microgrupos $Q = \{Q_1, Q_2, \dots, Q_q\}$.

Saída: Partição de dados com k grupos.

- 1 Considerar cada centroide de microgrupo, $\frac{LS}{n}$, como um objeto;
 - 2 Inicialização: k protótipos iniciais são amostrados com probabilidade diretamente proporcional a n ;
 - 3 **enquanto** Protótipos não se estabilizarem **faça**
 - 4 Particionamento: calcular a distância entre os protótipos e os centroides dos microgrupos;
 - 5 Atualização: o novo protótipo é definido como centroide ponderado pelos objetos no grupo;
 - 6 **fim**
-

Em (Bradley et al., 1998), outra variante do k -Médias para lidar com CF's é apresentada. Essa variante, denominada *Extended k-means*, utiliza os CFs e os objetos como entrada para encontrar k grupos. O algoritmo também considera os CFs como objetos ponderados por n . O algoritmo *Extended k-means* é similar ao apresentado no Algoritmo 2.2 exceto que

Algoritmo 2.3: Algoritmo *k-means++* (Arthur e Vassilvitskii, 2007).

Entrada: Número de grupos k e coresset M .

Saída: Partição de dados.

- 1 Escolher um centroide inicial c_1 uniformemente e de maneira aleatória a partir de M ;
- 2 **para** $i=2$ **to** k **faca**
- 3 Considerar que $d(\mathbf{x}_j)$ seja a menor distância do objeto $\mathbf{x}_j \in M$ para o seu centroide já escolhido $\{c_1, \dots, c_{i-1}\}$;
- 4 Escolher o próximo centroide $c_i = \mathbf{x}_z \in M$ com probabilidade $d(\mathbf{x}_z)^2 / \sum_{\mathbf{x}_j \in M} d(\mathbf{x}_j)^2$;
- 5 **fim**
- 6 Proceder com o tradicional algoritmo *k*-Médias;

o algoritmo *Extended k-means* tem um passo adicional para lidar com grupos vazios. Após o passo de convergência (linha 6 do Algoritmo 2.2), os grupos são verificados para detectar grupos vazios. Um grupo vazio tem seu centroide definido para o objeto mais distante e o algoritmo *Extended k-means* é chamado novamente para atualizar os protótipos (Bradley et al., 1998). Em (Farnstrom et al., 2000), o algoritmo *Extended k-means* é também utilizado no framework *Single-pass k-means*.

Outra variante do *k*-Médias para lidar com sumários estatísticos é apresentada em (Ackermann et al., 2012). Esse algoritmo de agrupamento, denominado *k-means++*, pode ser visto como um procedimento de sorteio para o tradicional algoritmo *k*-Médias. Conforme detalhado na Seção 2.1.1, os autores em (Ackermann et al., 2012) propuseram uma maneira de sumarizar o FCD por meio da extração de um pequeno conjunto de objetos, denominado coresset (Agarwal et al., 2004; Bădoiu et al., 2002). Vale relembrar que um coresset é um pequeno conjunto de objetos (ponderados) que aproximam dos objetos do FCD considerando o problema de otimização do *k*-Médias. O algoritmo proposto em (Ackermann et al., 2012) extrai o coresset do FCD por meio da técnica de *merge-and-reduce* (Har-Peled e Mazumdar, 2004) e encontra k grupos via algoritmo *k-means++*, conforme descrito no Algoritmo 2.3.

O algoritmo *LSearch* (O'Callaghan et al., 2002) utiliza o conceito de localização de facilidades (*facility location*) (Meyerson, 2001) para encontrar uma solução para o problema de otimização do algoritmo *k*-Medianas. A ideia principal é encontrar a localização da facilidade (mediana do grupo) que representa os objetos do FCD por meio da minimização de uma função de custo. Cada facilidade (mediana) tem um custo associado para ser "aberta" em uma dada localização e um custo de serviço para atender a demanda dos clientes (objetos). A função de custo é a soma dos custos associados para abrir as facilidades e o custo de serviço. Assim, essa função é a combinação da soma dos erros quadráticos (SSE) e um custo para inserir uma mediana dentro da partição de dados. O algoritmo *LSearch* busca por uma partição de dados com o número de grupos dentro do intervalo $[k, 2k]$ variando o valor do custo de abertura das facilidades. Ackermann et al.

(2012) observaram que o *LSearch* nem sempre encontra o valor k predeterminado e que geralmente a diferença está dentro da margem de 20% do valor de k determinado pelo usuário.

Além do algoritmo k -Médias, algoritmos de agrupamento baseados em densidade tais como o DBSCAN (Ester et al., 1996) são também utilizados na etapa de agrupamento *offline*. Os autores em (Cao et al., 2006) apresentaram o algoritmo *DenStream* que utiliza um método baseado nos vetores de características para sumarização do FCD e uma variante do DBSCAN para realizar o agrupamento. Essa variante recebe como entrada os p -*microclusters* (vetores de características) e dois parâmetros (ϵ e η , anteriormente descritos na Seção 2.1.1) para o agrupamento dos dados. Cada estrutura p -*microcluster* é considerada como um objeto virtual com centro igual à $\frac{LS}{WA_i}$, onde WA_i é o peso do grupo considerando os objetos em uma dada vizinhança do centro. Embora o usuário não necessite especificar exatamente o número de grupos, a definição de ϵ e η tem forte influência na partição de dados resultante.

Como visto na Seção 2.1, outro paradigma para agrupamento em FCD partitiona o espaço de entrada em células discretas. Tipicamente, esses algoritmos criam uma estrutura de malha dividindo o espaço de entrada em células por meio de algoritmos tradicionais de agrupamento. Por exemplo, a componente *offline* do algoritmo *D-Stream* (Chen e Tu, 2007) adota uma estratégia de agrupamento aglomerativa para agrupar as células. O algoritmo inicia associando cada célula densa a um grupo. Em seguida, um procedimento iterativo combina duas células densas que são fortemente correlacionadas em um único grupo. Esse procedimento é repetido até que nenhuma mudança na partição seja observada. Um parâmetro cujo valor é definido pelo usuário determina se duas células são fortemente correlacionadas.

Baseado no uso de células adaptativas, o algoritmo *Distributed Grid Clustering* (DGClust) (Gama et al., 2011; Rodrigues et al., 2008) é um algoritmo distribuído de agrupamento *online* para dados de sensores. O algoritmo DGClust reduz a dimensionalidade por meio do monitoramento e o agrupamento apenas de estados frequentes. Como anteriormente mencionado, o algoritmo *DGClust* é composto de sítios local e global, e cada sensor é relacionado a um FCD univariado, cujos valores são monitorados em um sítio local. O objetivo do sítio global é encontrar k grupos e manter uma partição de dados continuamente atualizada. Para reduzir sua complexidade computacional, o algoritmo *DGClust* mantém apenas os m estados globais mais frequentes (top- m estados, $m > k$). O objeto mais centralizado de cada um desses m estados globais é utilizado no agrupamento de dados. Assim que o sítio local encontra os m estados mais frequentes, um algoritmo de agrupamento tradicional pode ser aplicado nesses objetos centrais que representam os estados para minimizar o raio do grupo (ou de maneira equivalente o diâmetro do grupo). O algoritmo *Furthest Point* (Gonzalez, 1985) é utilizado nessa tarefa. Esse algoritmo seleciona um objeto inicial para o primeiro grupo e iterativamente seleciona o próximo objeto como o

centro de um novo grupo se sua distância ao centro dos grupos restantes é maximizada. Para ajustar dinamicamente a partição de dados, o algoritmo funciona em uma das duas possíveis condições: convergido ou não-convergido. Se o algoritmo está funcionando no modo não-convergido, quando um novo estado $h(t)$ é alcançado, o algoritmo atualiza os centros dos grupos de acordo com os top- m estados. Se o algoritmo já convergiu e o estado atual tem se tornado efetivamente parte dos top- m estados, o algoritmo atualiza os centros e muda seu estado para não-convergido.

2.2 Considerações Finais

Neste capítulo, um levantamento bibliográfico sobre algoritmos para agrupamento em FCD foi apresentado. Dentre os algoritmos de agrupamento apresentados, pode-se observar que o número de grupos a ser encontrado é especificado pelo usuário, apesar deste ser frequentemente desconhecido na prática de Mineração de Dados (MD). Neste sentido, são apresentados, em seguida, dois artigos oriundos do doutorado nos quais são propostos um método capaz de estender alguns algoritmos da literatura para estimar o número de grupos e um algoritmo evolutivo de agrupamento para estimar o valor de k .

Proposta de um Método para Estimação do Número de Grupos

Conforme se pode observar no Capítulo 2, diversos algoritmos de agrupamento em FCD baseados no k -Médias foram propostos na literatura. A grande maioria desses algoritmos assume que o número de grupos, k , é conhecido e definido pelo usuário *a priori*. Com o objetivo de relaxar essa suposição, a qual é frequentemente irrealística em aplicações práticas, é apresentado nesse capítulo um método que permite estender alguns algoritmos da literatura para estimar o valor de k automaticamente a partir dos dados. O potencial do método proposto é ilustrado por meio de três algoritmos do estado-da-arte para agrupamento em FCD – *Stream LSearch* (O’Callaghan et al., 2002), *CluStream* (Aggarwal et al., 2003) e *StreamKM++* (Ackermann et al., 2012) – combinado com dois algoritmos para estimar o número de grupos: Múltiplas Execuções Ordenadas do k -Médias (MEO k) (Naldi et al., 2011) e *Bisecting k-Means* (BkM) (Steinbach et al., 2000).

3.1 Algoritmos Baseados nas k -Médias

Os algoritmos baseados nas k -Médias sofrem de dois principais problemas (Everitt et al., 2001): i) dependendo da inicialização, o algoritmo pode ficar preso em soluções subótimas (mínimos locais); e ii) o número de grupos k precisa ser definido *a priori*. De maneira sucinta, o problema das k -Médias é agrupar N objetos de um conjunto de dados \mathcal{X} em k grupos de maneira que a soma das distâncias quadráticas dos objetos aos centros dos grupos aos quais cada objeto pertence seja minimizada (Jain e Dubes, 1988). Tal problema

de otimização é conhecido por ser NP-difícil. Portanto, algoritmos de aproximação são geralmente empregados.

3.1.1 Stream LSearch

O problema de agrupamento é formulado em O'Callaghan et al. (2002) de tal forma a encontrar uma solução para o problema de otimização do algoritmo k -Medianas. De maneira sucinta, a função objetivo desse problema consiste essencialmente em minimizar a soma quadrática das distâncias dos objetos às suas respectivas medianas. Porém, no problema de otimização do algoritmo *Stream LSearch* não é necessário conhecer o valor exato do número de grupos, apenas uma estimativa inicial. Baseado no conceito de localização de facilidades (ver Seção 2.1.3), a função objetivo do *Stream LSearch* é composta da soma das distâncias quadráticas dos objetos (demanda dos clientes) a suas medianas (facilidades) e de um custo para a criação das facilidades. O número de grupos, k , é um parâmetro do algoritmo definido pelo usuário, o qual é utilizado pela busca da melhor partição de dados considerando o número de grupos dentro do intervalo $[k, 2k]$. Essa busca consiste na escolha do valor para o custo de abertura da facilidade de modo que o número de medianas da solução esteja entre $[k, 2k]$. Ackermann et al. (2010) observaram que o algoritmo *Stream LSearch* nem sempre encontra uma partição com valor de k predefinido e que usualmente essa diferença está dentro de uma margem de 20% do valor de k especificado *a priori*.

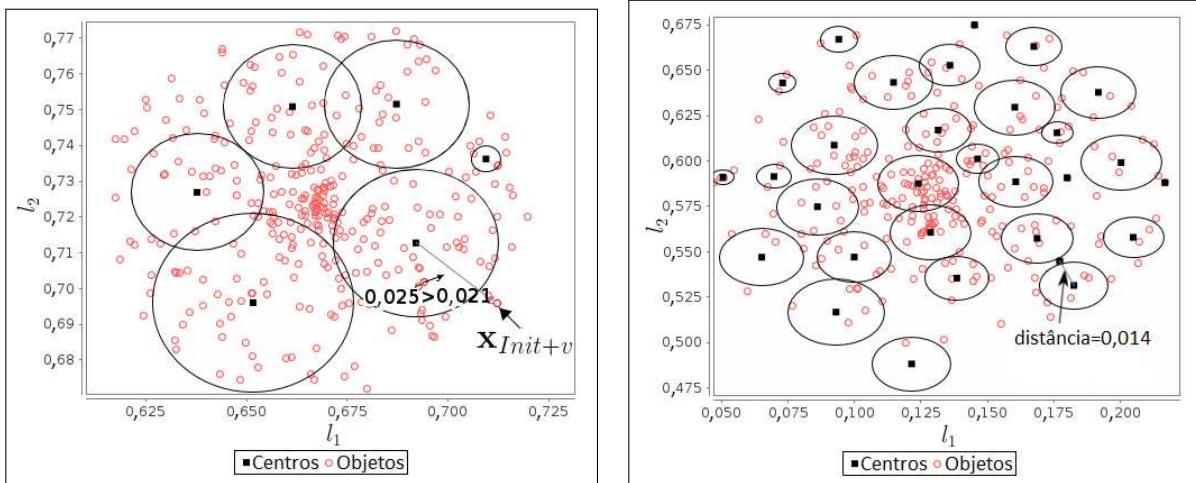
O'Callaghan et al. (2002) observaram que o problema abordado pelo algoritmo *Stream LSearch* é NP-difícil. Entretanto, algoritmos aproximados são usualmente empregados para essa classe de problemas. Aggarwal et al. (2003) notaram que o *Stream LSearch* é implementado como uma versão *online* do algoritmo das k -Médias e assume que os grupos são induzidos sobre todo o FCD. De maneira precisa, o algoritmo assume que os dados surgem em blocos $\mathbf{X}_1, \dots, \mathbf{X}_z$, sendo que cada bloco $\mathbf{X}_i, i \in [1, z]$ pode ser agrupado na memória principal. O algoritmo *Stream LSearch* é aplicado para processar cada bloco \mathbf{X}_i . No i -ésimo bloco do FCD o algoritmo mantém $O(i \times k)$ objetos. Para FCDs muito longos ($z \rightarrow \infty$), manter $O(i \times k)$ objetos pode se tornar inviável sob o ponto de vista de espaço de armazenamento. Para evitar esse problema, o algoritmo *LSearch* (ver Seção 2.1.3) agrupa os $O(i \times k)$ centros dos grupos e mantém apenas os $2k$ centros resultantes, como ilustrado anteriormente na Figura 2.5. Em particular, note que cada bloco \mathbf{X}_i , o qual tem m objetos, será agrupado em $2k$ centros. Devido à sua natureza, o algoritmo *Stream LSearch* não leva em conta a evolução dos dados, no sentido que para os grupos obtidos em cada bloco são dados à mesma importância ao longo do FCD.

3.1.2 CluStream

Este algoritmo considera que os dados evoluem com o tempo e permite explorar os grupos em diferentes porções do FCD. O processo de agrupamento é dividido em duas componentes: (i) componente de micro agrupamento (*online*) que periodicamente armazena sumários estatísticos do FCD e (ii) componente de macro agrupamento (*offline*) que utiliza os sumários estatísticos em conjunto com parâmetros de entradas definidos pelo usuário para prover grupos de dados.

CluStream faz o uso de uma estrutura de dados denominada microgrupo (ver Seção 2.1.1) para manter os sumários estatísticos do FCD. Inicialmente, o *CluStream* executa o algoritmo das k -Médias em um conjunto inicial de objetos predefinido, encontrando q microgrupos. Conforme observado por Aggarwal et al. (2003), o valor de q é determinado pela quantidade de memória que está disponível para armazenar os microgrupos. Entretanto, valores típicos de q são maiores que o número natural de grupos nos dados e menores que o número de objetos que surgem em longos períodos de tempo. Portanto, para cada novo objeto que surge, o algoritmo atualiza os microgrupos para refletir as mudanças nos dados.

Considere que cada objeto é descrito por meio de um vetor de características l -dimensional $\mathbf{x}_v = [x_v^j]_{j=1}^l$, $1 \leq v \leq \infty$. Portanto, após a execução do k -Médias em uma quantidade pre-definida de objetos (*Init*) para criar os q microgrupos, um objeto \mathbf{x}_{Init+v} é inserido em um microgrupo existente ou em um novo microgrupo *i.e.*, o objeto \mathbf{x}_{Init+v} corresponde a um *outlier* ou à semente de um novo grupo (neste caso, devido ao surgimento de novos grupos). Para ilustrar como o *CluStream* funciona, a Figura 3.1 apresenta os objetos como círculos, o centro dos microgrupos como quadrado e um novo objeto \mathbf{x}_{Init+v} . Este novo objeto é inserido em um microgrupo existente se e somente se sua distância ao microgrupo mais próximo estiver dentro de um limite máximo do microgrupo. Caso contrário, um novo microgrupo, que contém inicialmente apenas o objeto \mathbf{x}_{Init+v} , é criado – ver Figura 3.1a, onde a distância do objeto ao microgrupo mais próximo, 0,025, é maior que o raio desse microgrupo, o qual é 0,021. Assim o número de microgrupos (outros) é reduzido por um quando criamos espaço na memória – ver Figura 3.1b, onde dois microgrupos mais próximos, cuja distância entre eles é 0,014, são unidos para liberar espaço na memória. Especificamente, um microgrupo mais antigo é removido ou dois novos microgrupos são unidos. O microgrupo mais antigo é removido se o valor do seu marcador de tempo (t) exceder um dado limiar δ (parâmetro de entrada). Caso contrário, os dois microgrupos mais antigos são unidos. Os q microgrupos são armazenados em um dispositivo secundário de tempos em tempos, tais microgrupos conhecidos como *snapshots*. Estes *snapshots* são armazenados em diferentes níveis de granularidade de tempo, que permitem uma busca de grupos em diferentes horizontes de tempos, através do conceito de janelas de tempo piramidais (Aggarwal et al., 2003). A componente de macro agrupamento utiliza os micro-



(a) Encontrando o microgrupo mais próximo ao novo objeto x_{Init+v} , cuja distância (0,025) de x_{Init+v} ao microgrupo mais próximo é maior que o raio desse microgrupo (0,021).

(b) Unindo dois microgrupos mais próximos para criar um novo microgrupo com o objeto x_{Init+v} .

Figura 3.1: Exemplo ilustrativo do algoritmo *CluStream*.

grupos para encontrar k grupos (por meio de uma variante do k -Médias, conforme descrito na Seção 2.1.3) para um dado horizonte de tempo, h . O usuário fornece (entre outros parâmetros) k e h como entrada.

3.1.3 StreamKM++

Este algoritmo é baseado no k -means++ (ver Algoritmo 2.3), o qual pode ser visto como um procedimento para o sorteio dos protótipos iniciais para o tradicional algoritmo k -Médias. O StreamKM++ possui uma componente denominada de *streaming* que mantém um sumário dos objetos do FCD por meio de um procedimento denominado *merge-and-reduce* (ver Algoritmo 2.1) que consiste de dois passos: união (*merge*) e redução (*reduce*). Basicamente, esse procedimento organiza uma pequena quantidade de amostras (conjunto de L blocos) de tal maneira que um bloco L_i contenha m objetos, onde m é um parâmetro de entrada, que representam $2^i \times m$ objetos do FCD. Quando existirem dois blocos que representam a mesma quantidade de objetos, estes são unidos (passo de *merge*) em $2m$ objetos e um novo bloco é criado contendo m objetos (por meio do passo de *reduce*).

Para ilustrar o procedimento *merge-and-reduce*, considere que há três blocos, conforme ilustrado na Figura 3.2. Os objetos são sempre inseridos no primeiro bloco (L_1), o qual é representado por uma linha tracejada. Inicialmente, alguns objetos (que estão representados como círculos largos no tempo $t = 1$) são inseridos apenas no bloco L_1 . Quando esse bloco estiver totalmente preenchido, seus objetos são deslocados para o bloco vizinho (L_2) quando este estiver vazio, conforme ilustrado no tempo $t = 1$. Entretanto, se os blocos L_1 e L_2 estão cheios, então os objetos desses dois blocos são unidos e o resultado

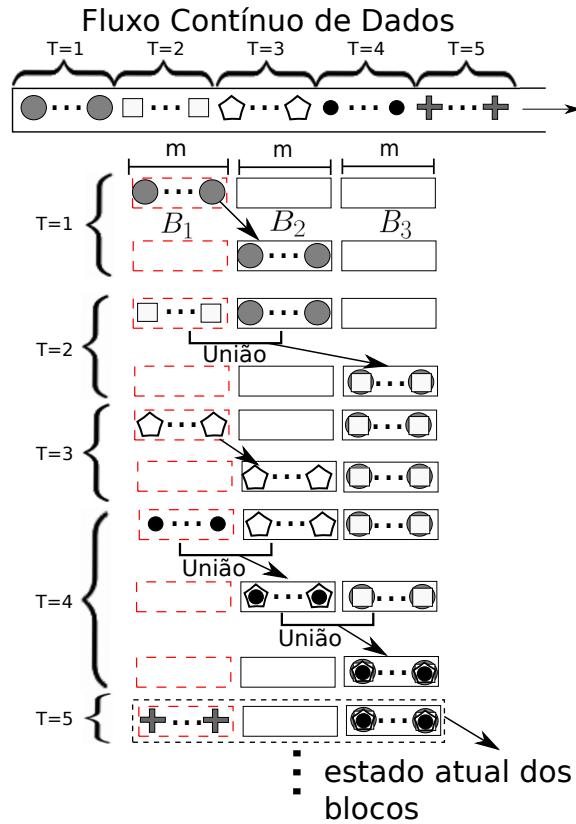
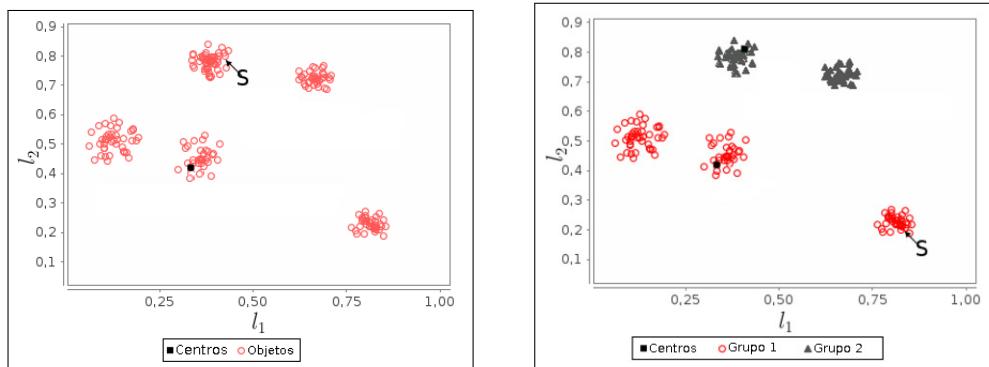


Figura 3.2: Ilustração do passo de redução utilizado no StreamKM++ (Ackermann et al., 2012).

é mantido no terceiro bloco (L_3), conforme ilustrado na Figura 3.2 no tempo $t = 2$. Dessa maneira, é liberado espaço nos blocos L_1 e L_2 para a inserção dos novos objetos (a menos que o terceiro bloco também esteja cheio, neste caso o processo é repetido como ilustrado na Figura 3.2, quando são inseridos os objetos representados pelo símbolo “●” no tempo $t = 4$).

Para o passo de redução, os autores propuseram uma estrutura de dados denominada árvore de *coreset*, conforme descrito na Seção 2.1.1. Basicamente, essa estrutura de dados utiliza os conceitos de agrupamento de dados para reduzir os $2m$ objetos em m objetos, *i.e.*, realiza um agrupamento dos $2m$ objetos em m grupos (*coreset*). A Figura 3.3 ilustra um exemplo do passo de redução, no qual o algoritmo recursivamente partitiona os objetos ($2m$) em dois grupos até encontrar m grupos. Em seguida, o algoritmo *k-means++* é aplicado nesses m objetos (*coreset*) para obter k grupos de dados, sendo k um parâmetro do usuário.



(a) Etapa 1: Partição de dados inicial é formada por um simples grupo. Um objeto x_i é selecionado probabilisticamente para ser o próximo centro de grupo (inicial).

(b) Etapa 2: A partição de dados é formada por dois grupos (triângulos e círculos), cujos custos individuais (soma das distâncias quadráticas) são 2,831 e 12,541, respectivamente. O grupo com maior custo (círculos) é escolhido para ser dividido. Novamente, um novo objeto, o qual é representado por x_i , é escolhido para ser o centroide inicial do grupo.

Figura 3.3: Exemplo ilustrativo do passo de redução usado no algoritmo StreamKM++ (Ackermann et al., 2012).

3.2 Estimando o Número de Grupos

Um dos problemas mais difíceis de lidar no agrupamento de dados é determinar o número apropriado de grupos (Jain et al., 1999). Determinar um número apropriado de grupos é um procedimento usualmente baseado em dois passos principais. Primeiro, os dados são particionados com diferentes valores de k . Segundo, medidas de qualidade, conhecidas como índices de validade relativa (Jain et al., 1999) são utilizados para avaliar a adequação das partições obtidas. Os critérios relativos de agrupamento de dados que são não monotônicos com relação ao número de grupos podem ser potencialmente utilizados como uma função objetivo para um algoritmo que é projetado para otimizar o valor de k . Entre as alternativas que estão disponíveis na literatura, foi escolhida a versão simplificada da Silhueta (Kaufman e Rousseeuw, 1990) como função objetivo. Tal índice tem se destacado entre os melhores em um estudo comparativo abrangente de critérios de validade relativos para agrupamento de dados realizado por Vendramin et al. (2010).

3.2.1 Silhueta Simplificada (ss)

Para explicar esse índice de validade relativo, considere um conjunto de dados X com N objetos e uma partição rígida de X (*crisp*) em k grupos $C = \{C_j\}_{j=1}^k$, conforme definida no Capítulo 2.

Inicialmente, será apresentada a silhueta tradicional que foi proposta por Kaufman e Rousseeuw (1990). Considere que um objeto x_i pertence a um grupo C_a . A dissimilaridade

dade¹ média de \mathbf{x}_i para todos os outros objetos no grupo C_a é descrita como $a(\mathbf{x}_i)$. Em seguida, considere um grupo C_b . A dissimilaridade média de \mathbf{x}_i para todos os objetos em C_b será descrito como $d(\mathbf{x}_i, C_b)$. Para todos os grupos $C_b \neq C_a$, o grupo mais próximo é selecionado, i.e., $b(\mathbf{x}_i) = \min d(\mathbf{x}_i, C_b)$. Este valor representa a dissimilaridade de \mathbf{x}_i para seu grupo vizinho mais próximo e a silhueta $s(\mathbf{x}_i)$ é obtida pela Equação 3.1.

$$s(\mathbf{x}_i) = \frac{b(\mathbf{x}_i) - a(\mathbf{x}_i)}{\max\{a(\mathbf{x}_i), b(\mathbf{x}_i)\}} \quad (3.1)$$

Note que $0 < s(\mathbf{x}_i) < 1$ se os objetos são atribuídos ao grupo mais próximo. Assim, quanto maior é o valor de $s(\mathbf{x}_i)$, melhor situado está o objeto \mathbf{x}_i a um dado grupo. Porém, se $s(\mathbf{x}_i) = 0$, então está indefinido quando um objeto deve ser alocado ao seu grupo atual ou a um grupo vizinho (Everitt et al., 2001). Finalmente, se o grupo C_a é um *singleton* (grupo constituído de apenas um objeto) a escolha mais neutra é atribuir $s(\mathbf{x}_i) = 0$ (Kaufman e Rousseeuw, 1990).

A silhueta proposta por Kaufman e Rousseeuw (1990) depende do cálculo de todos os pares de distâncias entre os objetos, o qual leva a um custo computacional de $O(N^2)$; este custo é frequentemente considerado alto para aplicações em FCD. Para solucionar esta limitação, a SS pode ser utilizada (Hruschka et al., 2006). A SS é baseada no cálculo das distâncias entre os objetos e os centroides dos grupos. Especificamente, o termo $a(\mathbf{x}_i)$ da Equação 3.1 se torna a dissimilaridade do objeto \mathbf{x}_i ao centroide do seu grupo correspondente (C_a). De maneira similar, ao invés de calcular $d(\mathbf{x}_i, C_b)$ como a dissimilaridade média de \mathbf{x}_i para todos os objetos de C_b , $C_b \neq C_a$, as distâncias entre \mathbf{x}_i e o centroide de C_b são calculadas. Assim, para os algoritmos baseados no k -Médias a silhueta pode ser obtida pela Equação 3.2.

$$s(\mathbf{x}_i) = 1 - \frac{a(\mathbf{x}_i)}{b(\mathbf{x}_i)} \quad (3.2)$$

Enquanto essas modificações reduzem o custo computacional estimado de $O(N^2)$ para $O(N)$, resultados empíricos sugerem que a qualidade obtida das partições pode não ser afetada significativamente (Hruschka et al., 2004, 2006). A média de $s(\mathbf{x}_i)$ para todo $i = 1, \dots, N$ é obtida pela Equação 3.3. Esta equação pode ser utilizada como critério para avaliar a qualidade de uma determinada partição. Ao utilizar esse critério, o melhor agrupamento é alcançado quando o valor de SS é maximizado.

$$\text{SS} = \frac{1}{N} \sum_{i=1}^N s(\mathbf{x}_i) \quad (3.3)$$

¹Dissimilaridades são calculadas usualmente a partir de medidas de distância, tais como a bem conhecida distância Euclideana

3.3 Algoritmos para Gerar Partições de Dados

Uma vez que tenha sido estabelecido o índice de validade relativo para avaliar a qualidade das partições, é possível estudar maneiras para estimar o valor de k com o algoritmo das k -Médias. Nas próximas seções serão apresentados dois métodos para estimar o valor de k via k -Médias: MEO k e *Bisecting k-Means* (BkM).

3.3.1 Múltiplas Execuções Ordenadas do k -Médias (MEO k)

Na prática, um método comum para estimar o valor de k envolve obter diferentes partições (possivelmente com diferentes valores de k) e avaliar cada partição obtida para estimar o melhor valor para k (Jain, 2009). Este método é um procedimento que é essencialmente realizado pelo MEO k (Naldi et al., 2011), o qual executa o k -Médias repetitivamente para um número crescente de grupos. Para cada valor de k , um número de partições obtidas pelo k -Médias (de diferentes inicializações) é avaliado por meio de um critério de validade relativo — *e.g.*, a SS (Hruschka et al., 2006) utilizada nesta tese — para estimar o melhor valor de k , que será denotado como \hat{k} e sua correspondente partição de dados. Portanto, após executar o k -Médias para cada valor de k em $\{k_{\min}, \dots, k_{\max}\}$, a melhor partição de dados (com relação ao índice de validade) é selecionada. Nos experimentos foi adotada a seguinte regra prática (Pal e Bezdek, 1995): os valores de k_{\min} e k_{\max} foram definidos em dois e \sqrt{N} , respectivamente, onde N é o número de objetos a ser agrupado.

3.3.2 Bisecting k -Means (BkM)

O algoritmo BkM (Steinbach et al., 2000) é uma variante hierárquica (divisivo) do algoritmo das k -Médias que recursivamente particiona os dados em dois grupos em cada passo, até que o número desejado de grupos seja encontrado. Este algoritmo considera que inicialmente há um único grupo que é formado por todos os objetos do conjunto de dados. Tal grupo é dividido em dois grupos. A partir deste momento, o algoritmo BkM recursivamente escolhe um grupo a ser dividido. Existem diversas maneiras de fazer essa divisão. Nesta tese, foi escolhido o critério de maior largura, onde a largura do grupo é medida como duas vezes a distância do centroide ao objeto mais distante que pertence a este grupo. Para automaticamente estimar o valor de k a partir dos dados, foi utilizado o método descrito em (Naldi et al., 2009), no qual o BkM estima \hat{k} do conjunto $\{2, \dots, k_{\max}\}$ via o critério da SS (Hruschka et al., 2006). Similarmente ao critério adotado para o MEO k , foi adotado o valor de $k_{\max} = \sqrt{N}$.

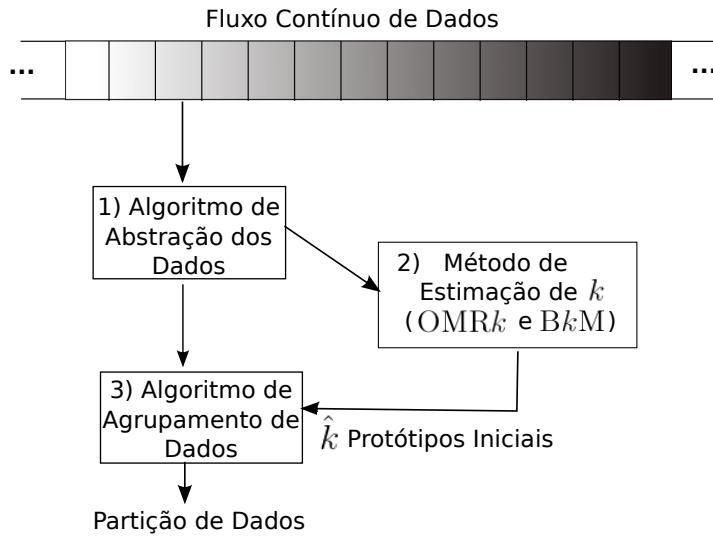


Figura 3.4: Arquitetura do método proposto.

3.4 Proposta de um Método para Estimação de k

Em geral, os algoritmos para agrupamento descritos no Capítulo 2 podem ser resumidos por meio de duas etapas: etapa de abstração dos dados (etapa conhecida no Inglês como *streaming*) e a etapa de agrupamento dos dados. Basicamente, a etapa de abstração dos dados sumariza o FCD com o auxílio de estruturas de dados específicas que foram desenvolvidas para lidar com as restrições de memória que são típicas em aplicações de FCD. A etapa de agrupamento obtém uma partição de dados a partir desses sumários para fornecer uma rápida compreensão do FCD por meio de grupos de dados.

Para estimar o número de grupos a partir dos dados, foi proposto um método que utiliza uma etapa intermediária de estimação do valor de k , etapa de estimação do k (passo 2), como ilustrado na Figura 3.4. Precisamente, esta etapa utiliza os dados sumarizados na etapa de abstração para estimar o número de grupos via os algoritmos MEO k e B k M. Então, não somente o valor de k é estimado, mas os centros de grupos servem como entrada para a etapa de agrupamento. Maiores detalhes de como incorporar os algoritmos da Seção 3.1 no método proposto são apresentados a seguir.

Na Figura 3.5 é ilustrado um exemplo de execução do método. Primeiramente, considere novamente que o FCD surge em blocos (X_1, \dots, X_T). O método processa um bloco por vez. Note que no exemplo de execução, como descrito no Capítulo 2, os algoritmos para agrupamento em FCD criam uma abstração dos dados a partir dos blocos. Em seguida, a etapa de estimativa de k automaticamente estima o número de grupos e então envia ambos os parâmetros \hat{k} e os protótipos obtidos para a etapa de agrupamento. Esta adaptação depende do algoritmo de agrupamento utilizado (relembrando que o objetivo é estender três algoritmos, *Stream LSearch*, *CluStream* e o *StreamKM++*). Por exemplo, ao utilizar o *CluStream*, o k -Médias encontrará \hat{k} grupos a partir dos microgrupos obtidos na

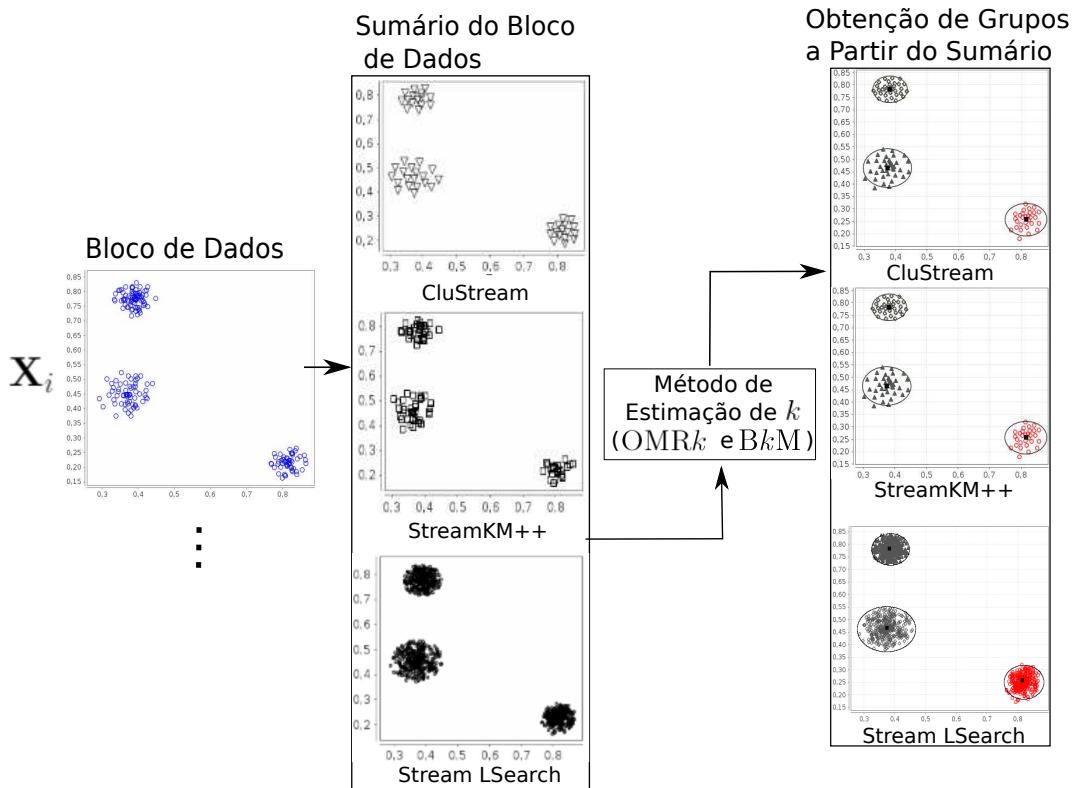


Figura 3.5: Exemplo de execução do método que é utilizado para estimar o número de grupos a partir dos dados.

etapa de abstração. Note também que, ao estimar o valor de k , é possível reduzir a interação do usuário, não necessitando, portanto, informar constantemente o número de grupos. O método permite que o usuário foque apenas nas mudanças do número de grupos, \hat{k} , se assim for desejado. Além disso, o usuário pode ainda monitorar os grupos de dados, embora esse monitoramento seja tedioso e suscetível a erro.

3.4.1 Adaptação para o *Stream LSearch*

A abstração dos dados é construída a partir da estratégia de divisão e conquista e divide o FCD em blocos. Mais precisamente, para cada bloco X_1, \dots, X_T o número de grupos é estimado via os algoritmos MEO k ou BkM. A etapa de agrupamento é realizada pelo algoritmo *LSearch*, o qual refina os centroides que foram obtidos pela etapa da estimação do k .

3.4.2 Adaptação para o *CluStream*

A abstração dos dados é construída a partir da componente *online*, que mantém um conjunto de microgrupos. A componente *offline*, por sua vez, executa uma variante do k -Médias (ver Algoritmo 2.2) para refinar os centroides obtidos na etapa de estimação do valor de k .

3.4.3 Adaptação para o StreamKM++

Um conjunto denominado *coreset* é utilizado na etapa de abstração dos dados para manter uma representação compactada dos objetos do FCD. No Algoritmo 2.3 o *k-means++* é executado no *coreset*, com o valor do número de grupos estimado na etapa \hat{k} . Os protótipos obtidos a partir dessa etapa não são considerados, entretanto, porque o *k-means++* tem seu próprio procedimento para inicializar as \hat{k} sementes iniciais.

3.5 Método Incremental para Estimação de k

3.5.1 Silhueta Simplificada Incremental (SSI)

Em cenários de FCD, um algoritmo idealmente deve ser capaz de atualizar a partição de dados de maneira *online*. Esta alternativa pode economizar os recursos computacionais quando os grupos não mudam significativamente em uma janela de tempo. Note que para atualizar a partição de dados de maneira *online* é necessário avaliar incrementalmente as decisões tomadas durante esse processo de atualização. Nesse caso, o objetivo é estimar o impacto da inserção de um objeto que surge do FCD em um grupo mais próximo. Especificamente, se deseja detectar quando as atribuições (*online*) de um objeto ao grupo mais próximo pioram a silhueta total (ver Equação 3.3). Nesta seção, é apresentada uma proposta para o cálculo da SS incrementalmente.

Considere inicialmente que uma partição de dados em k grupos ($C_c, 1 \leq c \leq k$) para uma predeterminada quantidade de dados N está disponível. Considere também que essa partição de dados foi avaliada com a SS. Em seguida, após inserir um novo objeto x' no grupo mais próximo, por exemplo, C_1 , o novo valor de silhueta total para a partição é calculado, conforme descrito na Equação 3.4. Após a inserção do objeto no grupo C_1 , esse será denominado de C'_1 . Portanto, deseja-se determinar qual a influência desse novo objeto no cálculo da SS. Para isso, assuma que somente o grupo atualizado se alterou. Essa suposição leva em consideração que ao inserir o objeto no grupo C_1 , a posição do centroide se altera, enquanto que para os demais grupos isso não ocorre. Portanto, apenas os valores de silhueta para cada objeto desse grupo modificado podem influenciar o cálculo da SS. Observe que é possível reescrever a Equação 3.4 pela soma das siluetas de cada grupo, conforme descrito na Equação 3.5, na qual C'_1 representa o grupo atualizado, $|C'_1|$ o tamanho desse grupo e $SS_{C'_1}$ o valor de silhueta total considerando apenas os objetos do grupo C'_1 .

$$SS = \frac{s(\mathbf{x}') + \sum_{i=1}^N s(\mathbf{x}_i)}{N+1} \quad (3.4)$$

$$SS = \frac{SS_{C'_1} \times |C'_1| + \sum_{j=2}^k SS_{C_j} \times |C_j|}{|C'_1| + \sum_{j=2}^k |C_j|} \quad (3.5)$$

Após a formulação do problema, pode-se estimar a influência da inserção do objeto \mathbf{x}' no cálculo de silhueta por meio da diferença entre a soma das silhuetas dos grupos C'_1 e C_1 , conforme descrito nas Equações 3.6-3.8. Observe que ao inserir o objeto no grupo C_1 a posição do centroide se altera. Nesse caso, o cálculo de distância entre os objetos desse grupo e o novo protótipo é representado pelo termo $a'(\mathbf{x}_i)$.

$$SS_{C'_1} \times |C'_1| = \sum_{i=1, \forall \mathbf{x}_i \in C'_1}^{|C'_1|} \left(1 - \frac{a'(\mathbf{x}_i)}{b(\mathbf{x}_i)} \right) \quad (3.6)$$

$$SS_{C'_1} \times |C'_1| = \sum_{i=1}^{|C'_1|} 1 - \sum_{i=1, \forall \mathbf{x}_i \in C'_1}^{|C'_1|} \frac{a'(\mathbf{x}_i)}{b(\mathbf{x}_i)}$$

$$SS_{C'_1} \times |C'_1| = |C'_1| - \sum_{i=1, \forall \mathbf{x}_i \in C'_1}^{|C'_1|} \frac{a'(\mathbf{x}_i)}{b(\mathbf{x}_i)} \quad (3.7)$$

$$SS_{C_1} \times |C_1| = \sum_{i=1, \forall \mathbf{x}_i \in C_1}^{|C_1|} \left(1 - \frac{a(\mathbf{x}_i)}{b(\mathbf{x}_i)} \right)$$

$$SS_{C_1} \times |C_1| = \sum_{i=1, \forall \mathbf{x}_i \in C_1}^{|C_1|} 1 - \sum_{i=1}^{|C_1|} \frac{a(\mathbf{x}_i)}{b(\mathbf{x}_i)}$$

$$SS_{C_1} \times |C_1| = |C_1| - \sum_{i=1, \forall \mathbf{x}_i \in C_1}^{|C_1|} \frac{a(\mathbf{x}_i)}{b(\mathbf{x}_i)}$$

$$\Delta SS = SS_{C'_1} \times |C'_1| - SS_{C_1} \times |C_1| \quad (3.8)$$

Portanto, resolvendo a Equação 3.8 por meio das simplificações das Equações 3.6 e 3.7, obtém-se:

$$\Delta SS = \left(|C'_1| - \sum_{i=1, \forall \mathbf{x}_i \in C'_1}^{|C'_1|} \frac{a'(\mathbf{x}_i)}{b(\mathbf{x}_i)} \right) - \left(|C_1| - \sum_{i=1, \forall \mathbf{x}_i \in C_1}^{|C_1|} \frac{a(\mathbf{x}_i)}{b(\mathbf{x}_i)} \right) \quad (3.9)$$

$$\Delta SS = (|C'_1| - |C_1|) + \left(- \sum_{i=1, \forall \mathbf{x}_i \in C'_1}^{|C'_1|} \frac{a'(\mathbf{x}_i)}{b(\mathbf{x}_i)} + \sum_{i=1, \forall \mathbf{x}_i \in C_1}^{|C_1|} \frac{a(\mathbf{x}_i)}{b(\mathbf{x}_i)} \right)$$

$$\Delta SS = \left(1 - \frac{a'(\mathbf{x}')}{b(\mathbf{x}')} \right) + \left(- \sum_{i=1, \forall \mathbf{x}_i \in C_1}^{|C_1|} \frac{a'(\mathbf{x}_i)}{b(\mathbf{x}_i)} + \sum_{i=1, \forall \mathbf{x}_i \in C_1}^{|C_1|} \frac{a(\mathbf{x}_i)}{b(\mathbf{x}_i)} \right)$$

$$\Delta SS = \left(1 - \frac{a'(\mathbf{x}')}{b(\mathbf{x}')} \right) + \left(\sum_{i=1, \forall \mathbf{x}_i \in C_1}^{|C_1|} \frac{a(\mathbf{x}_i) - a'(\mathbf{x}_i)}{b(\mathbf{x}_i)} \right) \quad (3.10)$$

Pode-se observar na Equação 3.10 que o primeiro termo refere-se ao cálculo da silhueta para o objeto \mathbf{x}' , $s(\mathbf{x}')$. Já no segundo termo, tem-se a diferença entre $a(\mathbf{x}_i)$ e $a'(\mathbf{x}_i)$. Vale lembrar que o termo $a(\mathbf{x}_i)$ corresponde à distância Euclidiana entre o objeto \mathbf{x} e o centroide do seu grupo (no exemplo em questão, o grupo C_1 , μ_1), dada por $d(\mathbf{x}, \mu_1) = (\mathbf{x} - \mu_1)^T(\mathbf{x} - \mu_1)$. Portanto, substituindo os termos $a'(\mathbf{x}_i)$ e $a(\mathbf{x}_i)$, respectivamente, pelos termos $d(\mathbf{x}, \mu'_1)$ e $d(\mathbf{x}, \mu_1)$ obtém-se a Equação 3.11, sendo μ'_1 representando o centroide do grupo C'_1 e μ_1 representando o centroide do grupo C_1 .

$$\Delta SS = s(\mathbf{x}') + \left(\sum_{i=1, \forall \mathbf{x}_i \in C_1}^{|C_1|} \frac{d(\mathbf{x}_i, \mu_1) - d(\mathbf{x}_i, \mu'_1)}{b(\mathbf{x}_i)} \right) \quad (3.11)$$

Pode-se observar, ainda, que:

$$\begin{aligned} d(\mathbf{x}, \mu_1) - d(\mathbf{x}, \mu'_1) &= ((\mathbf{x} - \mu_1)^T(\mathbf{x} - \mu_1)) - ((\mathbf{x} - \mu'_1)^T(\mathbf{x} - \mu'_1)) \\ d(\mathbf{x}, \mu_1) - d(\mathbf{x}, \mu'_1) &= (\mathbf{x}^T \mathbf{x} - 2\mathbf{x}^T \mu_1 + \mu_1^T \mu_1) - (\mathbf{x}^T \mathbf{x} - 2\mathbf{x}^T \mu'_1 + \mu'^T \mu'_1) \\ d(\mathbf{x}, \mu_1) - d(\mathbf{x}, \mu'_1) &= (2\mathbf{x}^T \mu'_1 - 2\mathbf{x}^T \mu_1 - \mu'^T \mu'_1 + \mu_1^T \mu_1) \\ d(\mathbf{x}, \mu_1) - d(\mathbf{x}, \mu'_1) &= (2\mathbf{x}^T (\mu'_1 - \mu_1) - \mu'^T \mu'_1 + \mu_1^T \mu_1) \end{aligned} \quad (3.12)$$

Desta forma se obtém:

$$\begin{aligned}
\Delta SS &= s(\mathbf{x}') + \left(\sum_{i=1, \forall \mathbf{x}_i \in C_1}^{|C_1|} \frac{(2\mathbf{x}_i^\top (\boldsymbol{\mu}'_1 - \boldsymbol{\mu}_1) - \boldsymbol{\mu}'_1^\top \boldsymbol{\mu}'_1 + \boldsymbol{\mu}_1^\top \boldsymbol{\mu}_1)}{b(\mathbf{x}_i)} \right) \\
\Delta SS &= s(\mathbf{x}') + 2 \sum_{i=1, \forall \mathbf{x}_i \in C_1}^{|C_1|} \frac{\mathbf{x}_i^\top (\boldsymbol{\mu}'_1 - \boldsymbol{\mu}_1)}{b(\mathbf{x}_i)} - \sum_{i=1, \forall \mathbf{x}_i \in C_1}^{|C_1|} \frac{\boldsymbol{\mu}'_1^\top \boldsymbol{\mu}'_1}{b(\mathbf{x}_i)} \\
&\quad + \sum_{i=1, \forall \mathbf{x}_i \in C_1}^{|C_1|} \frac{\boldsymbol{\mu}_1^\top \boldsymbol{\mu}_1}{b(\mathbf{x}_i)}
\end{aligned} \tag{3.13}$$

Observando os termos da Equação 3.13, nota-se que o termo $s(\mathbf{x}')$ e os protótipos $\boldsymbol{\mu}'_1$ e $\boldsymbol{\mu}_1$ podem ser calculados imediatamente. Note que $\boldsymbol{\mu}'_1 = \frac{\mathbf{x}' + \boldsymbol{\mu}_1 \times |C_1|}{|C_1| + 1}$ e, portanto, pode ser calculado de maneira incremental. Porém, a dependência com o termo $b(\mathbf{x}_i)$ torna impossível calcular de maneira incremental a Equação 3.13. Isto ocorre por que é necessário considerar todos os objetos do grupo para calcular o termo $b(\mathbf{x}_i)$. Para FCD, este cálculo pode ser custoso porque não é possível armazenar todos os dados já processados para realizar esse cálculo a cada atualização. No entanto, uma maneira aproximada de calcular o impacto da inserção de um objeto na qualidade do agrupamento consiste em observar o valor de silhueta do grupo atualizado — nesse caso mantendo a suposição de que as silhuetas dos demais grupos não se modificam.

Considere que para cada grupo C_c , $1 \leq c \leq k$ é armazenado o valor mínimo de silhueta ss_c considerando todos os objetos que pertencem a este grupo. De acordo com a Equação 3.2, o valor mínimo de silhueta corresponde aos valores máximos para os termos $a(\mathbf{x}_i)$ e $b(\mathbf{x}_i)$, que geralmente são obtidos pelos objetos alocados nas bordas dos grupos. O valor ss_c pode ser utilizado como um limiar para determinar quando um novo objeto deve ser inserido em um grupo mais próximo C_c ou quando um novo grupo está surgindo. Dessa maneira, para cada novo objeto \mathbf{x}' é calculado $s(\mathbf{x}')$. Em seguida, é comparado esse valor com ss_c do grupo mais próximo correspondente C_c .

3.5.2 Método Incremental

Nesta seção, será descrita a variante incremental do método proposto na Seção 3.4. Especificamente, será descrito uma adaptação incremental dos algoritmos *Stream LSearch*, *CluStream* e *StreamKM++* para estimar o valor de k .

De acordo com o passo 2 da Figura 3.4, o número de grupos é estimado via MEO $_k$ ou BkM. O custo de processamento desse passo pode ser reduzido ao se introduzir um método para atualizar os grupos incrementalmente, tal como o método descrito na Seção 3.5.1. O método incremental resultante é ilustrado na Figura 3.6, onde Múltiplas Execuções Ordenadas do k -Médias Incremental (MEO $_k$) e Incremental Bisecting k -Means (IBkM) são os algoritmos resultantes da combinação dos passos 2 e 3.

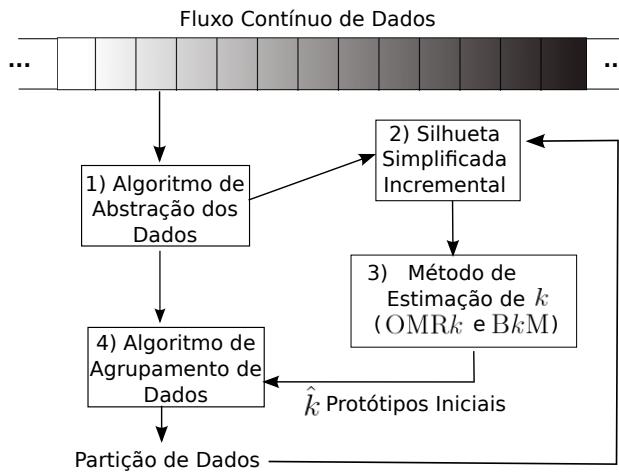


Figura 3.6: Arquitetura do método incremental proposto.

Adaptação Incremental para o *Stream LSearch*

Considere que uma janela deslizante de tamanho fixo é utilizada para armazenar os dados que surgem um por vez no passo 2. Relembre que, na estrutura de dados de janelas deslizantes, quando um novo objeto é inserido na janela outro objeto mais antigo é removido (ver Seção 2.1.2). Para o primeiro bloco de objetos do FCD, é obtida a partição de dados inicial (com o algoritmo MEO k ou B k M), e então calculada a silhueta dessa partição (ver Equação 3.2) e a silhueta ss_c de cada grupo. Em seguida, para cada novo objeto, x' , que surge, este é inserido na janela seguindo a estratégia *FIFO*. Para este objeto, é verificado o grupo mais próximo e calculado o valor de silhueta $s(x')$. Em seguida, é verificado se $s(x') > ss_c$. Se for atendida essa condição, então x' está contribuindo para aumentar a qualidade do grupo c , e, portanto deve ser inserido no grupo C_c . Caso contrário, assume-se que um novo grupo está surgindo. Nesse caso, o algoritmo das k -Médias é utilizado para agrupar os dados da janela com k e $k + 1$ grupos. Em seguida, é mantida a partição que obteve o maior valor de silhueta total. Por fim, o algoritmo MEO k ou B k M e o algoritmo de agrupamento do *Stream LSearch* são executado apenas nos blocos de objetos em que foram detectadas mudanças no número de grupos (de k para $k + 1$).

Adaptação Incremental para o *CluStream*

O algoritmo *CluStream*, conforme descrito na Seção 3.1.2, adota o conceito de microgrupos na sua componente de abstração de dados. Portanto, ao invés de observar o impacto da inserção dos objetos nos grupos de dados, é observado o impacto da inserção dos microgrupos.

Conforme descrito na Seção 3.1.2, inicialmente são obtidos os microgrupos com os primeiros $Init$ objetos do FCD, a partição de dados via MEO k ou B k M, o valor de silhueta da partição e a silhueta ss_c de cada grupo. Em seguida, para cada novo objeto, x' , que surge, este é inserido no microgrupo mais próximo ou é criado um novo microgrupo para

este objeto. Dado esse microgrupo atualizado ou o novo microgrupo criado, o passo 2 da Figura 3.6 identifica qual o grupo mais próximo desse microgrupo. Em seguida, é calculado o valor de silhueta desse objeto x' e comparado com o valor de silhueta ss_c do grupo mais próximo ($s(x') > ss_c$). Se o valor de silhueta do grupo diminuiu com a inserção desse microgrupo no grupo, então é possível que um novo grupo esteja surgindo. Portanto, o algoritmo das k -Médias é utilizado para agrupar os microgrupos com k e $k + 1$ grupos. Em seguida, é mantida a partição com maior valor de silhueta total. De maneira similar ao algoritmo *Stream LSearch*, o algoritmo MEO $_k$ ou BkM é executado a cada *Init* objetos processados somente se forem detectadas mudanças no número de grupos (de k para $k + 1$).

Adaptação Incremental para o *StreamKM++*

A adaptação do *StreamKM++* é similar àquela descrita anteriormente para o algoritmo *CluStream*. A exceção é que, ao invés de monitorar o impacto dos microgrupos, no *StreamKM++* é monitorado o impacto do *coreset* (conjunto de objetos representativos do FCD). Nessa estrutura de dados, cada objeto do FCD está relacionado com um objeto do *coreset*. Portanto, uma partição de dados é obtida via o agrupamento com MEO $_k$ ou BkM no *coreset*.

Considere que, inicialmente está disponível o *coreset* para uma quantidade inicial de objetos processados do FCD. Nesse momento, é obtida uma partição de dados com o MEO $_k$ ou BkM, o valor de silhueta total dessa partição e o valor ss_c de cada grupo de dados. Para cada novo objeto x' que surge do FCD é encontrado o objeto mais próximo em *coreset*, i.e., o objeto representativo cuja distância Euclidiana é menor em relação aos demais objetos representativos em *coreset*. Esse objeto representativo absorve o objeto x' de maneira similar à estrutura de microgrupos utilizada no *CluStream*. Portanto, o passo 2 da Figura recebe o objeto representativo atualizado. Em seguida, é identificado o grupo mais próximo desse objeto representativo, para o qual é calculado o seu valor de silhueta. Esse valor de silhueta é comparado com o valor ss_c do grupo mais próximo. Se $s(x') > ss_c$ então esse objeto representativo deve ser inserido no grupo C_c . Caso contrário, um novo grupo está surgindo. Nesse caso, o algoritmo das k -Médias é utilizado para agrupar *coreset* com k e $k + 1$ grupos. Em seguida, é mantida a partição que obteve o maior valor de silhueta total. Por fim, o algoritmo MEO $_k$ ou BkM e o algoritmo *k-Means++* são executados no *coreset* apenas em momentos que foram detectadas mudanças no número de grupos (de k para $k + 1$).

3.6 Considerações Finais

Neste capítulo foi apresentado um método para estimação do número de grupos. Tal método foi desenvolvido para permitir que os algoritmos do estado-da-arte em agrupamento

de FCD (*CluStream*, *Stream LSearch* e *StreamKM++*) possam estimar o número de grupos de maneira automática a partir dos dados. As variantes propostas para estimar o valor de k totalizam doze algoritmos:

- SLS-MEO k : *Stream LSearch* combinado com o MEO k ;
- SLS-B k M: *Stream LSearch* combinado com o B k M;
- CLS-MEO k : *CluStream* combinado com o MEO k ;
- CLS-B k M: *CluStream* combinado com o B k M;
- SKM-MEO k : *StreamKM++* combinado com o MEO k ;
- SKM-B k M: *StreamKM++* combinado com o B k M;
- SLS-MEO k I: *Stream LSearch* combinado com o MEO k I;
- SLS-IB k M: *Stream LSearch* combinado com o IB k M;
- CLS-MEO k I: *CluStream* combinado com o MEO k I;
- CLS-IB k M: *CluStream* combinado com o IB k M;
- SKM-MEO k I: *StreamKM++* combinado com o MEO k I;
- SKM-IB k M: *StreamKM++* combinado com o IB k M.

Tais algoritmos se baseiam em dois métodos bastante conhecidos para estimar o valor de k em bases estáticas (MEO k e B k M). No entanto, esses métodos realizam uma busca sistemática para encontrar o melhor valor de k , o que pode acarretar em um custo computacional elevado dependendo do número de soluções candidatas (partições de dados com diferentes valores de k) a serem analisadas. No contexto de agrupamento de dados em bases estáticas, Algoritmos Evolutivos (AEs) tem se mostrado úteis e eficientes para estimar o número de grupos e para “otimizar” as partições geradas. Por tal razão, com o projeto de AEs para FCD busca-se algoritmos com compromisso entre estimar corretamente o número de grupos e atender as restrições de tempo e armazenamento imposta em aplicações de FCD. Neste sentido, no próximo capítulo são apresentados AEs capazes de estimar o número de grupos em FCD.

Algoritmos Evolutivos para o Agrupamento em FCD com Número de Grupos Variável

Sob a perspectiva da otimização, agrupamento de dados pode ser considerado como um tipo específico de problema NP-difícil (Falkenauer, 1998). Algoritmos Evolutivos (AEs) são meta-heurísticas usualmente efetivas em problemas NP-difíceis, capazes de prover boas soluções para tais problemas em tempo razoável. Nesse sentido, diversos Algoritmo Evolutivos (AEs) foram propostos na literatura para resolver problemas de agrupamento de dados (Hruschka et al., 2009). A motivação para aplicar algoritmos evolutivos para mineração de dados é devido à robustez no sentido de realizar uma busca global no espaço de soluções candidatas, ao contrário dos métodos convencionais de mineração de dados que usualmente realizam uma busca local (Freitas, 2005). AEs são baseados na otimização de uma função-objetivo, também conhecida como função de aptidão, que guia a busca evolutiva (Eiben e Smith, 2003; Freitas, 2005).

Diversos trabalhos na literatura empregam AEs em problemas de mineração de dados (Freitas, 2002). Em problemas de classificação em FCD há trabalhos utilizando algoritmos evolutivos para geração de regras (Shafi e Abbass, 2009; Vivekanandan e Nedunchezhian, 2011) e *ensemble* de classificadores (Folino et al., 2007). Em relação ao problema de agrupamento em FCD com algoritmos evolutivos, nenhum trabalho foi encontrado até o momento. Diferentemente do cenário de FCD, diversos AEs para agrupamento de dados foram desenvolvidos para bases de dados estáticas — uma revisão bibliográfica sobre o

assunto pode ser encontrada em Hruschka et al. (2009); Naldi (2011). Em Naldi (2011); Naldi et al. (2009) foram realizados diversos experimentos empíricos com algoritmos de agrupamento para otimizar partições obtidas por meio do popular algoritmo k -Médias e seus resultados indicaram que AEs bem projetados podem ser eficientes computacionalmente quando comparado com estratégias de amostragem comumente utilizadas na prática — como execuções sistemáticas do k -Médias via, por exemplo o algoritmo Múltiplas Execuções Ordenadas do k -Médias (MEO k) (Naldi et al., 2011).

Considerando o potencial dos AEs para minerar dados e, principalmente, para agrupar dados, nesta tese foi considerada o uso de AEs para agrupamento em FCD. Em particular, foi considerado o algoritmo denominado *Fast Evolutionary Algorithm for Clustering* (FEAC), proposto em Alves et al. (2006), o qual tem se apresentado promissor na identificação de grupos automaticamente. Em particular, esse algoritmo apresenta custo computacional relativamente baixo e qualidade de agrupamento superior ou igual a alguns métodos sistemáticos para executar o k -Médias (Naldi, 2011). Entretanto, este algoritmo não apresenta a capacidade de lidar com FCD porque, neste tipo de cenário, os algoritmos devem idealmente realizar uma única leitura do FCD. Nesta tese, são apresentadas duas variantes do FEAC para lidar com FCD. As principais características do FEAC (Alves et al., 2006) consistem em: um esquema de codificação genotípica (representação interna da solução ao algoritmo evolutivo) baseada em vetores de valores inteiros de $N + 1$ posições, i.e., uma determinada posição i do vetor contém o rótulo de um grupo (variando de 1 a k) para o i -ésimo objeto do conjunto de dados (de tamanho N) e uma posição para armazenar o número de grupos da partição; uma função de adequabilidade baseada na Silhueta Simplificada (ss) (Hruschka et al., 2006) que avalia quão promissor é o indivíduo (solução candidata); seleção de indivíduos realizada por meio da estratégia da roleta (seleção proporcional) (Eiben e Smith, 2003); dois operadores de mutação, sendo um para remoção de grupos e outro para divisão de grupos. Ambos são probabilisticamente guiados pela informação da qualidade do agrupamento (função de adequabilidade) de um dado indivíduo; e o uso de um procedimento de busca local com o algoritmo das k -Médias. Na seção seguinte, os algoritmos propostos *Fast Evolutionary Algorithm for Clustering Data Stream based on Page Hinkley Test* (FEACS-PHT) e *Fast Evolutionary Algorithm for Clustering Data Stream based on Incremental Simplified Silhouette* (FEACS-ISS) são descritos em maiores detalhes.

4.1 ***Fast Evolutionary Algorithm for Clustering Data Streams* (FEACS)**

Conforme observado no Capítulo 2, os algoritmos de agrupamento em FCD geralmente utilizam duas componentes: uma para abstração dos dados e outra para o agrupamento

de dados em k grupos. Adicionalmente, uma componente intermediária pode ser inserida para a estimação automática do número de grupos a partir dos dados, conforme apresentado no Capítulo 3. Diferentemente dessas estratégias, os algoritmos evolutivos propostos nesta tese não necessitam de uma componente de abstração dos dados, porque nosso interesse recai em monitorar o comportamento dos grupos ao invés dos dados. Portanto, o FEACS utiliza apenas uma componente que mantém a partição de dados atualizada, o que reduz o custo de memória quando o tamanho do sumário (dados mantidos na componente de abstração de dados) é muito maior que o número de grupos – suposição tipicamente adotada pelos algoritmos da literatura (Ackermann et al., 2012; Aggarwal et al., 2003; O'Callaghan et al., 2002). Nas Seções 4.1.1 e 4.1.2 são descritas visões gerais dos algoritmos FEACS-PHT e FEACS-ISS, respectivamente. Em seguida, são descritas as componentes que são comuns nesses dois algoritmos: esquema de codificação (Seção 4.1.3) e evolução da partição dos dados (Seção 4.1.4). Por fim, são detalhadas as componentes de detecção de mudanças específicas de cada um dos algoritmos.

4.1.1 FEACS-PHT

O FEACS-PHT é baseado em um algoritmo de detecção ponto a ponto que utiliza o *Page-Hinkley Test* (PHT) (Mouss et al., 2004; Page, 1954). Basicamente, o algoritmo PHT foi desenvolvido para monitorar a variação no valor médio de um sinal Gaussiano e utiliza um limiar com valor constante (definido pelo usuário) para determinar a ocorrência de mudança no sinal. Nesta tese, o PHT foi adaptado para detectar em qual instante de tempo a partição de dados se altera significativamente. Portanto, assume-se que mudanças na distribuição dos dados podem refletir também em mudanças na partição de dados de maneira que qualquer grupo de dados pode aparecer ou desaparecer ao longo do tempo.

Na Figura 4.1, é apresentada uma visão geral do FEACS-PHT, o qual assume que os objetos vão surgindo um por vez. Inicialmente, o FEACS-PHT estima o número de grupos com os primeiros *Init* objetos do FCD. Uma vez que os grupos iniciais têm sido estabelecidos, a manutenção *online* dos grupos (inspirada em Aggarwal et al. (2003)) é iniciada. O FEACS-PHT incrementalmente atualiza os grupos de dados ao absorver o objeto mais próximo e remover o grupo mais antigo (Passo 1). O PHT monitora as distâncias médias entre os objetos processados e seus respectivos centroides (Passo 2). Quando esta distância média exceder o valor de um limiar de alerta (tw), o algoritmo entra em estado de alerta e inicia o armazenamento dos objetos em um *buffer* (F), até que um estado de alarme seja alcançado. Sempre que um estado de alarme (de acordo com o limiar ta) é declarado, o FEACS-PHT produz uma partição de dados para os objetos do *buffer*, conforme ilustrado no Passo 3 da Figura 4.1.

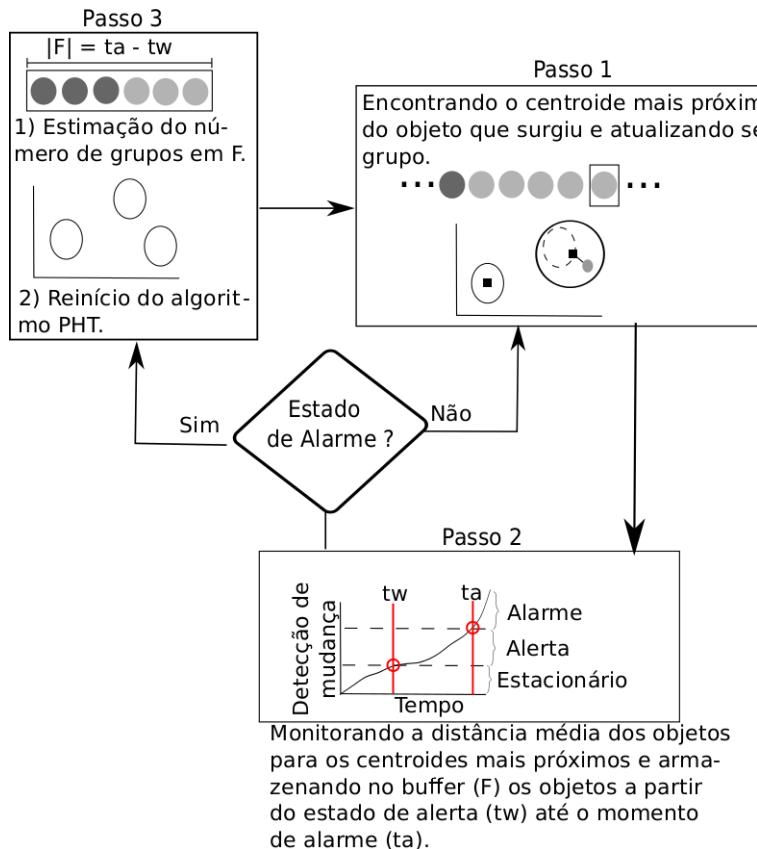


Figura 4.1: Visão geral da execução do FEACS-PHT. O Passo 3 está relacionado à estimativa do número de grupos. Os passos 1 e 2 estão relacionados à manutenção dos grupos de dados (encontrar o grupo mais próximo e remover o grupo mais antigo) e detectar mudanças (monitorar as distâncias entre os objetos e seus respectivos centroides), respectivamente.

4.1.2 FEACS-ISS

O FEACS-ISS é baseado no uso de janelas deslizantes (Seção 2.1.2) e na silhueta simplificada incremental (Seção 3.5.1). A estrutura de dados com janelas deslizantes é utilizada para armazenar os objetos mais recentes do FCD. A ideia é que a partição de dados esteja coerente com os novos padrões que vem surgindo do FCD. Dessa maneira, os dados mais recentes podem ser utilizados para capturar essas mudanças via agrupamento de dados. Neste contexto, a versão incremental da SS foi desenvolvida para monitorar a variação na qualidade da partição dos dados que está sendo monitorada (conforme detalhada na Seção 3.5.1). Especificamente, busca-se detectar um decaimento na qualidade da partição na medida em que os grupos são atualizados incrementalmente.

A Figura 4.2 apresenta uma visão geral do FEACS-ISS, o qual assume que os objetos vão surgindo um por vez. Inicialmente, o FEACS-ISS estima o número de grupos com os primeiros *Init* objetos do FCD que estão armazenados em uma janela deslizante. Uma vez que os grupos iniciais foram estabelecidos, a manutenção *online* dos grupos (inspirada em Aggarwal et al. (2003)) é iniciada. O FEACS-ISS incrementalmente atualiza os grupos de dados ao absorver o objeto mais próximo e remover o grupo mais antigo (Passo 1). Nesse processo a janela de dados é deslizada de modo a inserir o objeto mais recente e remover o objeto mais antigo. A SSI monitora a qualidade da partição observando a influência da inserção do objeto no grupo mais próximo (Passo 2). Quando esta influência for negativa *i.e.*, houver uma redução na qualidade da partição, o algoritmo produz uma partição de dados para os objetos armazenados na janela deslizante, conforme ilustrado no Passo 3 da Figura 4.2. Os detalhes algorítmicos da Figura 4.2 para evoluir partições de dados são apresentados na Seção 4.1.4. A manutenção dos grupos e os procedimentos para detecção de mudanças são apresentados na Seção 4.1.5.

4.1.3 Esquema de Codificação

O FEACS-PHT e o FEACS-ISS têm um esquema de codificação simples para descrever soluções candidatas (indivíduos¹) para o problema de agrupamento, conforme proposto em (Alves et al., 2006). Para explicar esse esquema, considere um conjunto de dados X com N objetos e uma partição rígida de X (*crisp*) em k grupos $C = \{C_j\}_{j=1}^k$. Uma partição de dados C é codificada como um vetor de inteiros de N posições. Cada posição desse vetor corresponde a um objeto do conjunto de dados, *i.e.*, a i -ésima posição corresponde ao i -ésimo objeto de X . Dessa maneira, cada elemento do vetor tem um valor sobre os possíveis valores de rótulo de grupo $\{1, \dots, k\}$. Por exemplo, considere a Figura 4.3, que ilustra tal esquema de codificação para o conjunto de dados hipotético formado por doze objetos ($x_i, i = \{1, \dots, 12\}$) que codifica três grupos (por conveniência, o número de grupos, k , é apresentado na última posição do vetor).

¹Também conhecido como genótipos ou cromossomos

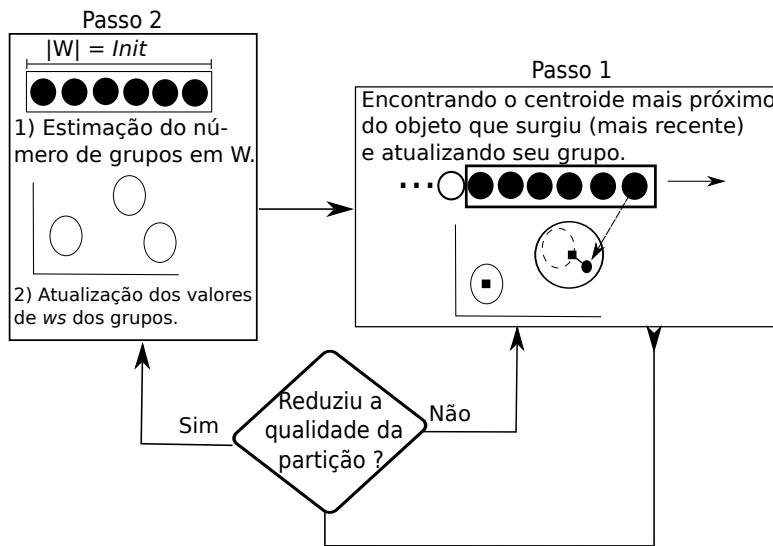


Figura 4.2: Visão geral da execução do FEACS-ISS. O Passo 2 está relacionado à estimção do número de grupos. O passo 1 está relacionado à manutenção dos grupos de dados (encontrar o grupo mais próximo) e detectar mudanças (monitorar as distâncias entre os objetos e seus respectivos centroides), respectivamente.

1	1	1	1	3	1	2	3	3	2	3	2	3
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	k

Figura 4.3: Esquema de codificação para uma partição de dados.

4.1.4 Evoluindo Partições de Dados

Para criar um conjunto de soluções candidatas para o problema de agrupamento, são utilizadas duas estratégias: i) inicializar a população de indivíduos aleatoriamente, e ii) utilizar operadores de mutação. No início do processamento do FCD, o conjunto de dados com os primeiros (*Init*) objetos do FCD é utilizado para criar uma partição de dados. Nesse momento, a primeira estratégia (i) é adotada. Esta estratégia compreende escolher aleatoriamente o valor de $k \in \{2, \dots, k_{\max}\}$ e os centroides (k objetos do conjunto de dados). Em seguida, o algoritmo das k -Médias é executado com esses parâmetros para gerar uma partição de dados que será codificada por um indivíduo de acordo com o esquema de codificação descrito anteriormente (Seção 4.1.3). A segunda estratégia é utilizada quando o FEACS-PHT entra em estado de alarme ou quando a FEACS-ISS observa uma redução na qualidade da partição, *i.e.*, quando os grupos atuais sendo atualizados não refletem mais as mudanças no FCD (grupos desatualizados). Baseado nos grupos desatualizados e naqueles objetos armazenados no *buffer* entre um estado de alerta e de alarme (para o caso do FEACS-PHT) ou na janela deslizante (para o caso do FEACS-ISS), uma partição de dados é construída. Então, uma população é composta de cópias replicadas desta partição de dados. A população de indivíduos que representa essas cópias replicadas será mutada (por meio de operadores de mutação) para promover a diversidade. Em seguida, a busca

evolutiva se inicia. Dois operadores de mutação são utilizados para essa tarefa (um para divisão de grupos e outro para remoção de grupos). Esses operadores são aplicados de maneira aleatória em cada indivíduo.

O Algoritmo 4.1 descreve o procedimento para estimativa do número de grupos do FEACS. Especificamente, cada iteração (geração) é repetida até que um critério de parada predefinido seja satisfeito (S_C) (passo 3 do Algoritmo 4.1). No Passo 4, o algoritmo executa um número ($iter$) de iterações do k -Médias para cada indivíduo da população, e esses indivíduos são avaliados por meio da SS (Equação 3.3). Múltiplos indivíduos são probabilisticamente selecionados e mutados para formar uma nova população para a próxima geração. A seleção é realizada via dois operadores bem conhecidos: elitismo (Passo 5) e estratégia da roleta (Passo 6). O elitismo foi adotado na implementação para manter o melhor indivíduo na população. Os indivíduos restantes ($P_{size} - 1$) são selecionados de acordo com a estratégia da roleta com reposição. Essa estratégia garante que quanto melhores os indivíduos são (maior valor de adequabilidade) maiores são as probabilidades de serem selecionados para a próxima geração. Especificamente, os indivíduos foram ordenados por meio de um procedimento de *ranking* de modo que o indivíduo com o melhor valor de adequabilidade receba o valor de *ranking* P_{size} o segundo melhor receba o valor de *ranking* $P_{size} - 1$, e assim sucessivamente até o último indivíduo, cujo valor de *ranking* seja 1. Essa seleção baseada em *ranking* pode evitar a convergência prematura do algoritmo evolutivo (Horta e Campello, 2009). Após esta etapa, o algoritmo aplica o operador de mutação no indivíduo selecionado.

Algoritmo 4.1: Algoritmo Evolutivo para Agrupamento de Dados.

```

/* Dado  $P(g)$  ser a população em uma geração  $g$ , dado  $S_C$  ser um
   critério de convergência e dado  $iter$  ser o número de iterações do
    $k$ -Médias */  

1  $g \leftarrow 1$ ;  

2 Inicializar a população  $P(g)$ ;  

3 enquanto  $S_C$  não é satisfeito faça  

4   Executar  $iter$  iterações do  $k$ -Médias em cada indivíduo em  $P(g)$  e avalie sua  

    adequabilidade de acordo com a SS (Equação 3.3);  

5   Aplicar elitismo;  

6   Selecionar os indivíduos a serem mutados por meio da estratégia da roleta;  

7   para cada indivíduo selecionado faça  

8     Selecionar o operador de mutação;  

9     Selecionar o grupo para mutação;  

10    Aplicar o operador de mutação;  

11  fim  

12  Copiar os indivíduos para a próxima população  $P(g)$ ;  

13   $g \leftarrow g + 1$ ;  

14 fim
```

De acordo com a taxonomia apresentada em Hruschka et al. (2009), os operadores de

mutação do FEACS-PHT são *orientados a grupos* (dependentes da tarefa específica para a qual foram projetados – nesse caso, projetados para eliminar, dividir e unir grupos) e *guiados* por meio da qualidade individual dos grupos – calculando a SS por grupo e que será utilizada para guiar o procedimento de busca evolutiva. O algoritmo utiliza dois operadores de mutação, os quais eliminam e dividem grupos. Estes operadores adotam uma busca informada (probabilística) que é guiada por meio da qualidade individual dos grupos. Especificamente, os grupos de menores qualidades, de acordo com a Equação 3.3 que permite avaliar a qualidade de cada grupo, são mais prováveis de serem mutados do que grupos de melhor qualidade. Dado que grupos com valores baixos de SS devem estar associados com probabilidades altas de mutação, a probabilidade de mutação de um grupo C_i é descrita pela Equação 4.1, sendo *rank* uma função que provê uma classificação ordinal (*ranking*) baseada no complemento das adequabilidades parciais de todos os grupos. Nesse caso, para k grupos, o pior grupo recebe valor de *ranking* k , o segundo pior grupo recebe valor de *ranking* $k - 1$ e assim sucessivamente até o último grupo, cujo valor de *ranking* é 1. A normalização baseada em *ranking* permite evitar que um grupo com qualidade muito inferior aos demais, de acordo com a SS, domine as probabilidades (Bäck et al., 2000).

$$p_m(C_i) = \frac{\text{rank}(1 - SS_{C_i})}{\sum_{j=1}^k \text{rank}(1 - SS_{C_j})} \quad (4.1)$$

O primeiro operador de mutação (OM_1) elimina um ou mais grupos selecionados de um dado indivíduo. Este operador é aplicado nos indivíduos que codificam mais que dois grupos. Em conformidade com esta condição, esse operador seleciona aleatoriamente $z \in \{1, \dots, k-2\}$ grupos a serem removidos de maneira que restem pelo menos 2 grupos na partição (indivíduo) – Algoritmo 4.2. A estratégia da roleta (seleção proporcional) (Eiben e Smith, 2003), sem reposição, é adotada para selecionar os grupos com probabilidade de acordo com a Equação 4.1. Em seguida, para cada objeto de um grupo selecionado C_v , $1 \leq v \leq z$ calculam-se as distâncias deste aos centroides dos grupos restantes. Por meio dessas distâncias, os objetos do grupo selecionado são atribuídos aos grupos mais próximos.

O segundo operador de mutação (OM_2) divide um ou mais grupos em dois grupos cada. Assim, o OM_2 permite aumentar o número de grupos. Esse operador somente divide grupos com no mínimo dois objetos. Similarmente ao OM_1 , os grupos são probabilisticamente selecionados com a estratégia da roleta, sem reposição. Considere C_v um grupo selecionado. Inicialmente, um centroide x^s é selecionado aleatoriamente para ser a semente de um novo grupo C'_v . Então, o objeto x^j mais distante de x^s é selecionado como semente de um novo grupo C''_v . Finalmente, o grupo C_v é eliminado, substituindo-se seus

Algoritmo 4.2: Operador de mutação OM₁ (adaptado de Naldi (2011)).

```

/* Seja  $C_1, \dots, C_k$  os grupos que compõem a partição  $C$  */  

1 se  $k > 2$  então  

2   Selecionar aleatoriamente um número  $z \in \{1, \dots, k - 2\}$ ;  

3   para  $o = 1, \dots, z$  faz  

4     Selecionar um grupo  $C_v \in C$  com probabilidade  $p_m(C_v)$ ;  

5     Alocar cada objeto do grupo  $C_v$  para o grupo restante mais próximo  

6      $C_r \in C, r \neq v$ ;  

7     Remover o grupo vazio  $C_v$ ;  

8 fim  

9 fim

```

objetos para os respectivos centroides \mathbf{x}^j e \mathbf{x}^s . Os passos do OM₂ são apresentados no Algoritmo 4.3.

Algoritmo 4.3: Operador de mutação OM₂ (adaptado de Naldi (2011)).

```

/* Seja  $C_1, \dots, C_k$  os grupos que compõem a partição  $C$  */  

1 Selecionar aleatoriamente um número  $z \in \{1, \dots, k\}$ ;  

2 para  $o = 1, \dots, z$  faz  

3   Selecionar um grupo  $C_v, v \in \{1, \dots, k\}$  da partição  $C$  com probabilidade  $p_m(C_v)$ ;  

4   se  $|C_v| \geq 2$  então  

5     Selecionar aleatoriamente um objeto  $\mathbf{x}^s \in C_v$ ;  

6     Encontrar o objeto  $\mathbf{x}^j \in C_v$  mais distante de  $\mathbf{x}^s$ ;  

7     Gerar dois novos grupos  $C'_v$  e  $C''_v$ , colocando os objetos de  $C_v$  mais próximos  

8     de  $\mathbf{x}^s$  em  $C'_v$  e os objetos de  $C_v$  mais próximos de  $\mathbf{x}^j$  em  $C''_v$ ;  

9 fim  

9 fim

```

A estratégia para a escolha dos operadores de mutação em estágios futuros da busca evolutiva é determinada pelo seu sucesso ao ser aplicado a um determinado indivíduo (Horta e Campello, 2009). Para melhor compreensão dessa estratégia, considere que para o i -ésimo indivíduo da população foi observado um aumento no valor de adequabilidade após ser submetido a um operador, por exemplo, o operador OM₁, em uma geração h . Esse operador é continuamente aplicado ao i -ésimo indivíduo da população, para as gerações seguintes enquanto não houver um decaimento no valor de adequabilidade do indivíduo. Caso contrário, o operador de mutação OM₂ é selecionado. Se o i -ésimo indivíduo não foi submetido a nenhum operador de mutação (preservado pela estratégia elitista ou no caso da geração atual ser a primeira), então os operadores OM₁ e OM₂ são selecionados aleatoriamente.

A seleção dos indivíduos é realizada por meio de uma estratégia elitista (Fogel, 1999) seguida da estratégia da roleta (Eiben e Smith, 2003). Devido à estratégia elitista, o melhor indivíduo da população é mantido na população para próxima geração. Para os demais indivíduos da população, é aplicada a seleção baseada na estratégia da roleta com repos-

sição. Esta estratégia utiliza o procedimento de normalização baseado em *ranking* para a seleção dos indivíduos, conforme apresentada na Equação 4.2, onde \mathbf{p} representa um indivíduo da população. Em particular, considere uma população com P_{size} indivíduos a serem submetidos aos dois operadores de mutação. Esses indivíduos serão ordenados de forma que o indivíduo com melhor valor de adequabilidade, de acordo com SS, receba valor de *ranking* P_{size} , o segundo melhor recebe o valor de *ranking* $P_{\text{size}} - 1$, e assim sucessivamente até o último indivíduo, cujo valor de *ranking* seja 1.

$$p_m(\mathbf{p}_i) = \frac{\text{rank}(SS_{\mathbf{p}_i})}{\sum_{j=1}^{P_{\text{size}}} \text{rank}(SS_{\mathbf{p}_j})} \quad (4.2)$$

4.1.5 Detecção de Mudanças

FEACS-PHT

Uma vez que a partição de dados inicial foi estabelecida com os primeiros objetos do FCD, os objetos subsequentes que surgem são absorvidos pelo grupo mais próximo, sendo que a proximidade é definida pela mínima distância Euclidiana. Para representar cada grupo, foi utilizada uma variante do vetor de características para armazenar os sumários dos grupos de dados (Aggarwal et al., 2003). Portanto, cada grupo é descrito por um vetor de características que é composto de quatro componentes: n , o número de objetos, S_1 , a soma linear dos vetores de objetos do grupo, S_2 , a sua soma quadrática e o marcador de tempo t_l do objeto mais recente do grupo (o último objeto que foi absorvido pelo grupo). Os três primeiros componentes permitem calcular medidas dos grupos de maneira incremental, tais como o centroide e o raio, que serão utilizados pelo algoritmo. O último componente é utilizado para ponderar a importância do grupo de acordo com seu tempo de vida, cujo valor decresce com o tempo via função de decaimento. O peso w de um grupo no tempo t é calculado conforme descrito na Equação 4.4, sendo λ um parâmetro definido pelo usuário que controla a taxa de decaimento².

$$w(t) = e^{-u(t)}, \quad (4.3)$$

$$u(t) = \frac{t - t_l}{\lambda} \quad (4.4)$$

A detecção de mudanças é relacionada a quantificar diferenças entre dois objetos, ou dois conjuntos de objetos e determinar quando as mudanças tem significância estatística (Gama, 2010). Para detectar mudanças na partição de dados, a distância euclidiana média

²Grupos com nenhuma atividade no FCD ($w < 0.1$) são removidos da partição de dados.

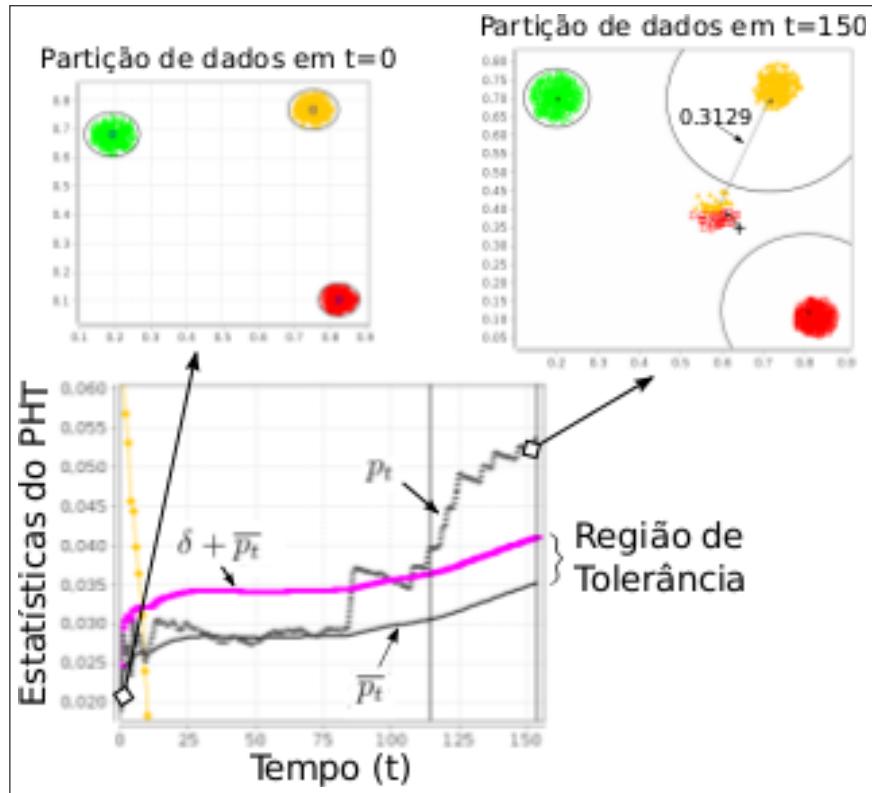
entre os objetos e seus centroides mais próximos é monitorada por meio de uma variante do PHT (Page, 1954). O PHT é um teste não paramétrico que é tipicamente utilizado para monitorar mudanças no processamento de sinais (Mouss et al., 2004; Page, 1954). Formalmente, considere p_r como sendo a variável cujo valor será monitorado no tempo r . Esse teste considera uma variável acumulativa $m(t)$, que é definida como a soma dos desvios entre o valor observado e sua média até o tempo t , conforme descrito na Equação 4.6.

$$m(t) = \sum_{r=1}^t (p_r - \bar{p}_r - \delta) \quad (4.5)$$

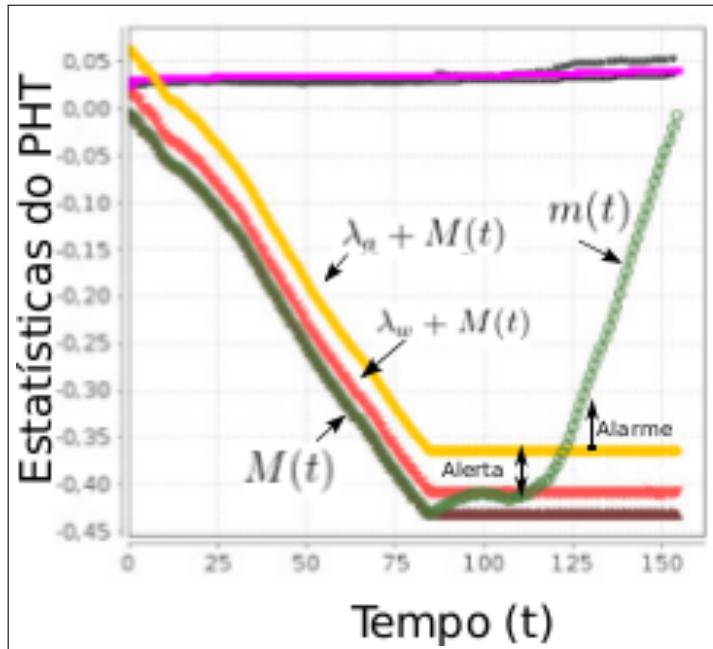
$$\bar{p}_r = \frac{\sum_{y=1}^r p_y}{r} \quad (4.6)$$

O parâmetro δ corresponde a um limiar de tolerância. Além disso, foram utilizados dois limiares para definir os estados de alerta e alarme, respectivamente, λ_w e λ_a . O PHT entra em estado de alarme quando a diferença entre duas variáveis $m(t)$ e $M(t) = \min\{m(i), i = 1, \dots, t\}$ está acima do limiar de detecção λ_a , i.e., quando $m(t) - M(t) > \lambda_a$.

Considere o exemplo de execução do PHT para detectar mudanças na partição de dados com três grupos, conforme ilustrado na Figura 4.4. Na Figura 4.4a, é apresentado como o PHT pode ser utilizado para monitorar as distâncias médias entre os objetos e os centroides de seus respectivos grupos. A variável p_t monitora as distâncias médias dos objetos processados até o tempo t . De acordo com a Equação 4.6, quando o valor de p_t está abaixo de $\bar{p}_t + \delta$, os valores de $m(t)$ são negativos (devido à soma de valores negativos); consequentemente, as curvas para $m(t)$ e $M(t)$ decrescem de $t = 0$ até $t = 80$ (ver Figura 4.4b). Entretanto, o espaço entre $m(t)$ e $M(t)$ se mantém crescente após a ocorrência de uma mudança no tempo $t = 80$ (um novo grupo de dados está surgindo), e PHT entra em estado de alerta quando o valor de $m(t)$ está acima de $M(t) + \lambda_w$, que ocorre no tempo $t = 114$. A partir desse momento, os objetos são armazenados em um *buffer* até que um estado de alarme seja declarado. Se o PHT entra em estado normal, então o *buffer* é esvaziado, evitando falsos alarmes (estado de alerta seguido de um estado normal). Sempre que o PHT entra em estado de alarme, o algoritmo evolutivo 4.1 é aplicado para encontrar uma nova partição de dados para os objetos no *buffer* e os parâmetros do PHT são reiniciados. Neste exemplo, δ , λ_w e λ_a estão definidos com os seguintes valores 0,006, 0,022 e 0,069, respectivamente. Observe que os valores para esses parâmetros não são definidos *a priori*, ao invés disso, eles são estimados diretamente a partir dos dados, conforme está descrito nos experimentos do Capítulo 5.



(a) Estatísticas do PHT para a variável p_t que corresponde a distância média entre o objeto e centroide mais próximo.



(b) Estatísticas do PHT para $m(t)$ (a soma dos desvios de p_t e sua média \bar{p}_t) e $M(t)$ (valor mínimo de $m(t)$ até o tempo t). λ_w (com valor igual 0,022) e λ_a (definido como 0,069) são parâmetros relacionados ao limiar para os estados de alerta e alarme, respectivamente.

Figura 4.4: Exemplo ilustrativo da execução do PHT. Este teste monitora as distâncias médias entre os objetos e os seus centroides mais próximos.

FEACS-ISS

No FEACS-ISS, a partição inicial do algoritmo é obtida com os *Init* objetos da janela deslizante (similar ao realizado com o FEACS-PHT). Após a criação da partição inicial (via procedimento descrito na Seção 4.1.4), são calculados os valores de silhueta total (SS_C) e mínimos (ss_c) de cada grupo. A partir desse momento, para cada novo objeto no fluxo é iniciado o processo de atualização incremental dos grupos. Para cada novo objeto que surge no fluxo a janela deslizante é descolada, inserindo o novo objeto e removendo o objeto mais antigo. Dessa maneira a janela se mantém sempre preenchida. Para cada novo objeto que surge este é inserido no grupo vizinho mais próximo temporariamente. Isto, porque não sabe-se ainda se é melhor inserir esse objeto no grupo mais próximo ou criar um novo grupo.

Para o objeto que foi inserido no grupo vizinho mais próximo, é calculado o seu valor de silhueta. Este cálculo é realizado via Equação 3.2. O valor de silhueta é comparado com o valor de silhueta mínimo do grupo. Caso o valor de silhueta do objeto seja menor que o valor mínimo de silhueta do grupo é verificado se a inclusão de um novo grupo pode melhorar a qualidade da partição. Isso ocorre porque ao permitir que esse objeto seja atribuído ao grupo mais próximo, o seu valor de silhueta que é relativamente baixo (comparado com o menor valor de silhueta do grupo) vai interferir na silhueta total da partição. Caso contrário, o objeto é inserido no grupo definitivamente sem interferir negativamente na qualidade da partição.

O processo de inclusão de um novo grupo na partição de dados consiste em criar um novo grupo considerando o objeto que ocasionou a diminuição na qualidade do grupo como protótipo desse grupo. Nessa situação, é verificado se a SS total da nova partição de dados ainda é maior que a antiga. Se essa condição for atendida, a nova partição é então considerada e uma mudança na partição é sinalizada. Caso contrário, a partição antiga ainda é mantida. Nessa situação, assume-se que os objetos observados ainda não são suficientes para criar um novo grupo. Portanto, continua-se a inclusão de novos objetos aos grupos. Ao observar que uma mudança na partição foi realizada no fim de cada *Init* objetos processados é aplicado o algoritmo evolutivo (Seção 4.1.4) para refinar as partições com os objetos presentes na janela deslizante.

4.2 Considerações Finais

Um algoritmo já existente na literatura, denominado de FEAC (Alves et al., 2006; Naldi et al., 2009), é capaz de eficientemente amostrar partições via o popular algoritmo k-means, incluindo a estimação automática do número de grupos. Entretanto, esse algoritmo não é apropriado para FCD. Esse capítulo descreveu adaptações necessárias para que o FEAC se torne apropriado para lidar com FCD. Em particular foram apresentados dois AEs inspi-

rados no FEAC e que se mostraram eficientes para induzir partições em FCD. Especificamente, foram estudadas estruturas de dados, estratégias de manutenção dos grupos de maneira incremental e a detecção de mudanças nas partições para a concepção dos algoritmos FEACS-PHT e FEACS-ISS. Foi desenvolvida uma estrutura de dados complementando o esquema de codificação (Seção 4.1.3) para armazenar as informações dos grupos: número de objetos, soma dos vetores de objetos, a sua soma quadrática, valor de silhueta e o instante de tempo da última atualização do grupo. As três primeiras características são úteis para calcular medidas como o raio do grupo e o centroide, de maneira incremental. O valor de silhueta contém o valor de qualidade do grupo. O instante de tempo foi utilizando para determinar o tempo de atividade do grupo ao longo do tempo para permitir que grupos possam desaparecer quando necessário. Para determinar esse tempo de atividade foi proposta um função de ponderação para determinar pesos aos grupos. Além disso, foi estudado o uso de janelas deslizantes (comumente utilizadas pelos algoritmos de agrupamento em FCD) para gerenciar o FCD. Também foram propostos dois métodos para detectar mudanças nas partições de dados: uma versão adaptada do PHT ([Page, 1954](#)) e a Silhueta Simplificada Incremental (SSI) ([Silva e Hruschka, 2014](#)). Esses dois métodos foram propostos com o intuito de detectar os instantes de tempo em que as partições mudam significativamente a ponto de permitir o uso do algoritmo evolutivo para encontrar o novo valor de k : de acordo com o critério da silhueta, o que pode reduzir o tempo computacional dos algoritmos ao invés de utilizar um AE frequentemente. Diferentemente dos algoritmos da literatura de FCD que fazem a manutenção dos dados do FCD, o FEACS-PHT e o FEACS-ISS fazem a manutenção dos grupos de dados, o que é de grande interesse quando se deseja interpretar o FCD por meio de grupos de dados. Essa característica possibilita também uma redução de armazenamento uma vez que o número de grupos é geralmente menor que o número de dados mantidos na componente de abstração (também conhecida como componente online) dos algoritmos da literatura.

Avaliação Experimental

Nos Capítulos 3 e 4 foram apresentados ao todo 14 algoritmos para agrupamento em FCD. Neste capítulo são apresentados os experimentos realizados com os algoritmos desenvolvidos, e os respectivos resultados experimentais obtidos.

5.1 Bases de Dados

Para a avaliar a capacidade dos algoritmos em detectar a evolução dos grupos ao longo do tempo foram geradas seis bases de dados artificiais (com diferentes números de atributos) e também foram utilizadas duas bases de dados reais. Nas Seções 5.1.1 e 5.1.2 são descritas as bases de dados artificiais e reais, respectivamente.

5.1.1 Bases de Dados Artificiais

Para a realização dos experimentos controlados nessa tese, empregou-se o uso de partições de referência, ou seja, partições em que são conhecidos os grupos de dados. Tipicamente, tais partições são utilizadas para se escolher um algoritmo de agrupamento mais adequado à uma aplicação ou para calibrar seus parâmetros. Nesse sentido, na literatura é possível encontrar vários experimentos que utilizam a partição de referência para avaliar algoritmos de agrupamento para FCD em termos de qualidade, escalabilidade e sensibilidade (Ackermann et al., 2010, 2012; Aggarwal et al., 2003, 2004; Cao et al., 2006; Chen e Tu, 2007; O'Callaghan et al., 2002; Silva e Hruschka, 2011). Nesse contexto, os grupos de dados das partições de referência são normalmente obtidos a partir de dados gerados artificialmente, de acordo com alguma distribuição de probabilidades.

As bases de dados artificiais foram geradas de distribuições gaussianas que têm diferentes dimensões $l = \{2, 4, 8, 16, 32, 64\}$. Para simular a evolução dos grupos ao longo do tempo, o número de grupos foi aleatoriamente variado entre 2 e 8 grupos. Além disso, a média e o desvio padrão de cada grupo foram aleatoriamente modificados. Cada base de dados artificial tem 200.000 objetos. A ordem de geração dos dados dos grupos é aleatória. O gerador de dados artificial é uma extensão do gerador *RBFGenerator*, o qual está disponível no software MOA¹ (Bifet et al., 2010), para gerar grupos hiper-esféricos de distribuições gaussianas. Especificamente, o gerador *RBFGenerator* foi modificado para gerar partições rígidas de dados (*crisp*), que são especialmente adequadas para avaliar algoritmos de agrupamento tais como Stream LSearch (SLS), CluStream (CLS) e StreamKM++ (SKM). Da mesma maneira como disponível no software MOA, o gerador utilizado nos experimentos permite escolher o número de parâmetros, tais como o valor mínimo e máximo de grupos, o raio do grupo, número de atributos, a velocidade do fluxo e, também, tornar possível o controle para inserção e remoção de grupos. Na Figura 5.1 é possível observar a dinâmica dos grupos de dados obtidos pelo gerador.

5.1.2 Bases de Dados Reais

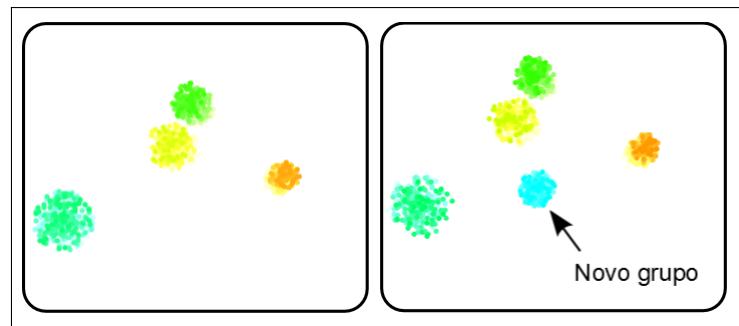
A primeira base de dados real utilizada foi a versão com 10% da base de detecção de intrusos KDDCup99², amplamente utilizada na literatura (Ackermann et al., 2010; Aggarwal et al., 2003; O'Callaghan et al., 2002; Silva e Hruschka, 2011), e que é mais desafiadora do que a sua versão completa (Masud et al., 2011). Como em (Ackermann et al., 2010; Aggarwal et al., 2003; O'Callaghan et al., 2002; Silva e Hruschka, 2011) foram utilizados os 34 atributos contínuos deste conjunto de dados, que contém 494.020 registros de conexões em uma rede de computadores e 23 diferentes classes de dados. Cada objeto se refere a uma conexão normal ou um entre 22 diferentes tipos de ataque. A base de dados foi convertida em um FCD ao se considerar a ordem dos dados como um fluxo. Os atributos foram normalizados para o intervalo [0,1].

A segunda base de dados real utilizada foi a base *Forest Cover Type*³, que contém 581.012 descrições geoespaciais de 7 tipos de cobertura florestal. Os objetos são descritos por 54 atributos: 10 atributos quantitativos e 44 atributos binários (4 para descrever áreas selvagens e 40 atributos para descrever tipos de solo). Como realizado por diversos experimentos da literatura (Aggarwal et al., 2004; Masud et al., 2011) foram utilizados todos os 10 atributos quantitativos. Similar ao realizado com a base KDDCup99, o conjunto de dados foi convertido em um FCD ao se considerar a ordem dos dados como um fluxo. Os atributos foram normalizados para o intervalo [0,1].

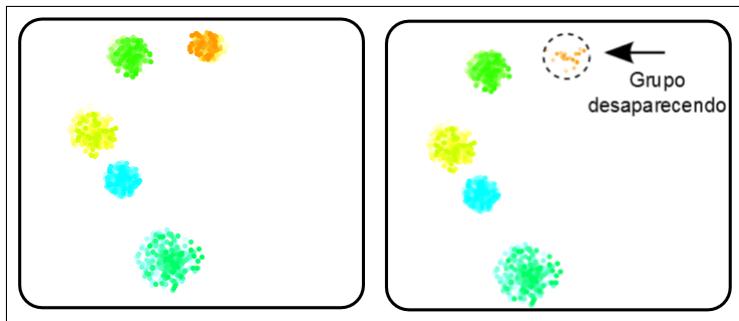
¹Disponível em: <http://moa.cms.waikato.ac.nz/>

²Disponível em: <https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data>

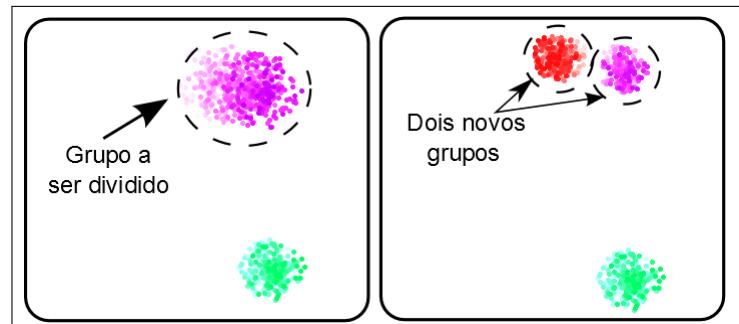
³Disponível em: <https://archive.ics.uci.edu/ml/datasets/Covertype>



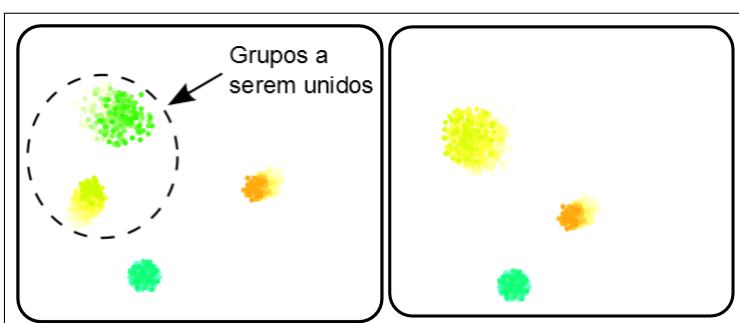
(a) Exemplo ilustrativo do surgimento de um novo grupo, formando uma partição com 5 grupos.



(b) Exemplo ilustrativo do desaparecimento de um grupo.



(c) Exemplo ilustrativo da divisão de um grupo em dois novos grupos.



(d) Exemplo ilustrativo da união de dois grupos.

Figura 5.1: Exemplo ilustrativo da evolução dos grupos ao longo do tempo — simulação realizada via gerador de dados sintético. As imagens da esquerda ilustram os grupos antes das modificações, enquanto que as imagens da direita ilustram o resultado das modificações.

5.2 Configuração Experimental

5.2.1 Algoritmos

Nos experimentos foram avaliados os 14 algoritmos propostos para agrupamento em FCD capazes de estimar o número de grupos:

- CLS-MEO k (combinação do algoritmo CluStream com o algoritmo de estimação de k , MEO k)— Seção 3.4.2;
- CLS-BkM (combinação do algoritmo CluStream com o algoritmo de estimação de k , BkM)— Seção 3.4.2;
- CLS-MEO k I (combinação do algoritmo CluStream com a versão incremental do MEO k via SSI)— Seção 3.5.2;
- CLS-IBkM (combinação do algoritmo CluStream com a versão incremental do BkM via SSI)— Seção 3.5.2;
- SLS-MEO k (combinação do algoritmo Stream LSearch com o algoritmo de estimação de k , MEO k)— Seção 3.4.1;
- SLS-BkM (combinação do algoritmo Stream LSearch com o algoritmo de estimação de k , BkM)— Seção 3.4.1;
- SLS-MEO k I (combinação do algoritmo Stream LSearch com a versão incremental do MEO k via SSI)— Seção 3.5.2;
- SLS-IBkM (combinação do algoritmo Stream LSearch com a versão incremental do BkM via SSI)— Seção 3.5.2;
- SKM-MEO k (combinação do algoritmo StreamKM++ com o algoritmo de estimação de k , MEO k)— Seção 3.4.3;
- SKM-BkM (combinação do algoritmo StreamKM++ com o algoritmo de estimação de k , BkM)— Seção 3.4.3;
- SKM-MEO k I (combinação do algoritmo StreamKM++ com a versão incremental do MEO k via SSI)— Seção 3.5.2;
- SKM-IBkM (combinação do algoritmo StreamKM++ com a versão incremental do BkM via SSI)— Seção 3.5.2;
- FEAC-PHT (combinação do algoritmo FEAC com o método de detecção de mudança PHT)— Seção 4.1.1;

- FEAC-SSI (combinação do algoritmo FEAC com o método de detecção de mudança via SSI)— Seção 4.1.1;

Devido à natureza aleatória dos algoritmos, esses foram executados 10 vezes para cada base de dados. Da mesma maneira que em ([Vendramin et al., 2010](#)), MEO k foi executado de modo a inicializar 10 diferentes partições de dados para cada valor de k . O parâmetro relacionado ao B k M (número de iterações) foi também definido como sendo igual a 10. Para ambos os algoritmos, o k -Médias foi programado para parar quando um dos seguintes critérios for satisfeito: i) 5 iterações tenham sido concluídas ou ii) a diferença máxima absoluta entre centroides de duas iterações consecutivas for menor ou igual a $10^{(-3)}$.

Os parâmetros dos algoritmos foram definidos de modo a permitir uma comparação mais justa possível entre eles. Para os algoritmos baseados no CluStream, foram definidas as seguintes configurações: $Init = 1.000$ para bases artificiais e $Init = 2.000$ para base real; $m = 50$ e $\delta = 512$ para bases artificiais e reais; q igual a 100 e 200 microgrupos, respectivamente para as bases artificiais e reais. O parâmetro relacionado com o limite máximo de raio (*radius factor*) foi definido em 2. Os valores dos parâmetros do CluStream foram determinados de acordo com ([Aggarwal et al., 2003](#)). Para as versões dos algoritmos baseados no SKM, o número de coresets foi definido em 100 para bases artificiais e 200 para as bases reais. Com relação às versões do SLS, devido à natureza dinâmica do FCD, não foi realizado o armazenamento dos protótipos de grupos obtidos de blocos já processados para evitar que protótipos desatualizados (antigos) dominem a análise do comportamento do FCD, o que pode deteriorar a qualidade do agrupamento.

Os parâmetros do FEACS-PHT e FEACS-ISS foram definidos da seguinte maneira: população formada por 10 indivíduos e máximo de 10 iterações para o k -Means. Os algoritmos foram programados para encerrar a busca evolutiva quando um dos seguintes critérios forem satisfeitos: i) adequabilidade do melhor indivíduo não melhorar mais do que 0,001 após 10 gerações consecutivas; ou ii) 50 gerações tiverem sido completadas. O parâmetro $Init$ foi definido em 1.000 para bases artificiais e 2.000 para as bases reais.

No FEACS-PHT, os parâmetros foram definidos da seguinte maneira: o parâmetro λ (Equação 4.4) foi definido igual a $Init$. Este parâmetro está relacionado à taxa de decaimento da função exponencial. O algoritmo auto ajusta os parâmetros do PHT pela seguinte heurística: como é comum em configurações *online* ([Agarwal et al., 2004; Chen e Tu, 2007](#)), o valor de λ_a foi definido como o raio médio dos grupos⁴; λ_w é a fração do raio $\frac{\lambda_a}{3}$, e $\sigma = 0.1 * \lambda_a$. A ideia é utilizar as informações do limites dos grupos para determinar os parâmetros do PHT, o que faz sentido porque o PHT monitora a distância média entre os objetos e os centroides dos grupos mais próximos. Estes parâmetros são estimados após a

⁴Por meio dos sumários estatísticos do vetor de característica dos grupos, o raio foi calculado da seguinte maneira: $\sqrt{\frac{S_2}{n} - \frac{S_1^2}{n}}$

partição de dados estar disponível, *i.e.*, no início do processamento do FCD e quando o PHT dispara um alarme. Nos experimentos, o tamanho do *buffer* foi definido como $10\% * Init$. Note que o usuário pode determinar o tamanho mínimo do *buffer* de acordo com o recurso computacional disponível.

Os experimentos foram conduzidos em um computador com 2.20GHz i7 quad-core (8GB de memória) executando o sistema operacional Ubuntu. Todos os algoritmos foram implementados na linguagem de programação Java.

5.2.2 Avaliação do Agrupamento

Para as bases artificiais, os dados foram processados em blocos formados por 1.000 objetos. Para cada bloco foram calculados os valores de Índice Rand Ajustado (IRA) ([Arabie e Hubert, 1996](#)) e os respectivos tempos computacionais dos algoritmos em milissegundos.

Nas bases reais, blocos de dados contendo 2.000 objetos foram utilizados para avaliar os algoritmos. Nessas bases, o número de grupos não é conhecido *a priori*. Além disso, a premissa de que classes correspondem a grupos é usualmente irrealista. Essa é provavelmente uma das razões para que muitos dos experimentos reportados na literatura, por exemplo ([Ackermann et al., 2010](#); [Aggarwal et al., 2003](#); [O'Callaghan et al., 2002](#)) adotem a popular medida de validação de agrupamento baseada no erro quadrático médio das partições. Entretanto, é conhecido que este critério não é apropriado para avaliar partições com diferentes números de grupos. Portanto, as partições de dados podem ser avaliadas por meio de um índice relativo de validade. Por conta dessas observações, nos experimentos realizados foi adotada a Silhueta Simplificada (SS).

Vale ressaltar que, em cenários de FCD, assume-se que se o bloco de dados está cheio, os demais dados não são enviados até que o bloco seja processado. Além disso, na prática, ajustes são feitos nos parâmetros dos algoritmos para levar em conta o recurso computacional disponível.

5.3 Resultados

5.3.1 Bases de Dados Artificiais

O uso de bases de dados artificiais permite um controle nas condições experimentais, em termos de variação dos grupos gerados e número de atributos, possibilitando a criação de cenários desafiadores para os algoritmos. Além disso, há a vantagem de estar disponível o conhecimento das partições verdadeiras, o que permite uma avaliação apropriada de qualquer outra partição de dados encontrada pelos algoritmos. Dessa maneira, índices de validade externos como o Índice Rand Ajustado (IRA) podem ser utilizados para avaliar as partições de dados.

Análise dos Resultados de IRA e Tempo de Processamento

A Tabela 5.1 apresenta os resultados médios obtidos para o IRA de 10 diferentes inicializações dos algoritmos para cada base de dados artificial com 2,4,8,16,32 e 64 dimensões. Devido ao tamanho do bloco ser de 1.000 objetos e haver 200.000 objetos em cada base de dados, há então 200 valores de IRA e tempo de processamento para cada par algoritmo e base dados artificial. Dessa maneira, são apresentados os valores de média e desvio padrão calculados sobre os 200 valores. Pode-se observar na Tabela 5.1 que todos os algoritmos estudados provêm bons resultados de IRA, sugerindo que partições de dados de alta qualidade foram obtidas em todas bases de dados — valores de IRA variando entre 0,95 e 1,00.

Tabela 5.1: Valores médios de IRA - Base de Dados Sintética.

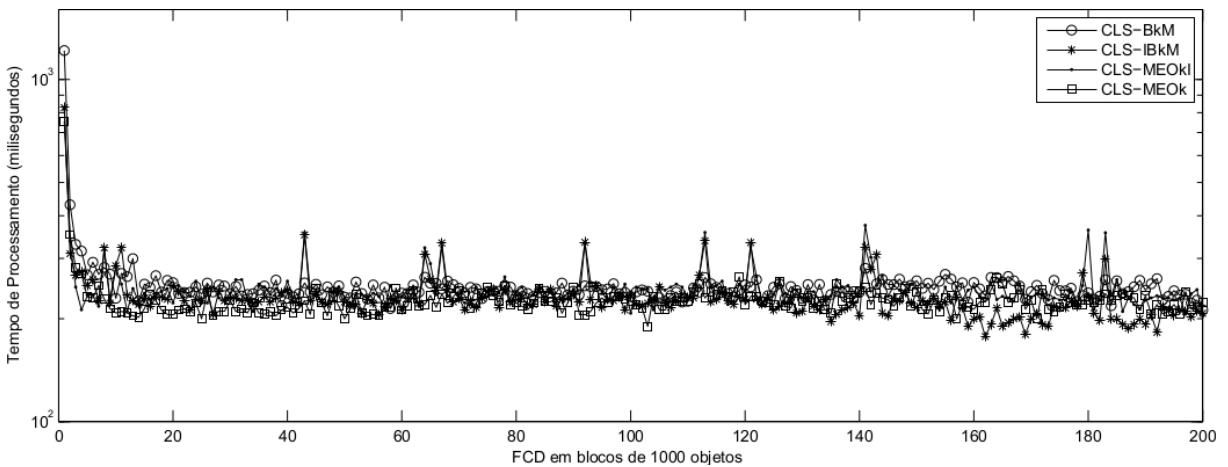
Algoritmo	Base - 2D $\mu(\pm\sigma)$	Base - 4D $\mu(\pm\sigma)$	Base - 8D $\mu(\pm\sigma)$	Base - 16D $\mu(\pm\sigma)$	Base - 32D $\mu(\pm\sigma)$	Base - 64D $\mu(\pm\sigma)$
CLS-B _k M	0,97 ($\pm 0,10$)	0,98 ($\pm 0,08$)	0,97 ($\pm 0,09$)	0,97 ($\pm 0,09$)	0,99 ($\pm 0,04$)	0,98 ($\pm 0,06$)
CLS-IB _k M	0,98 ($\pm 0,08$)	0,97 ($\pm 0,10$)	0,96 ($\pm 0,14$)	0,98 ($\pm 0,10$)	0,99 ($\pm 0,05$)	0,98 ($\pm 0,06$)
CLS-MEO _k	0,99 ($\pm 0,04$)	0,99 ($\pm 0,02$)	0,99 ($\pm 0,01$)	0,99 ($\pm 0,03$)	0,99 ($\pm 0,02$)	0,99 ($\pm 0,02$)
CLS-MEO _k I	0,98 ($\pm 0,06$)	0,99 ($\pm 0,03$)	0,99 ($\pm 0,03$)	0,99 ($\pm 0,06$)	0,99 ($\pm 0,04$)	0,99 ($\pm 0,03$)
SKM-B _k M	0,95 ($\pm 0,14$)	0,96 ($\pm 0,10$)	0,97 ($\pm 0,09$)	0,96 ($\pm 0,10$)	0,97 ($\pm 0,08$)	0,99 ($\pm 0,05$)
SKM-IB _k M	0,96 ($\pm 0,12$)	0,97 ($\pm 0,10$)	0,98 ($\pm 0,06$)	0,98 ($\pm 0,08$)	0,98 ($\pm 0,07$)	0,99 ($\pm 0,05$)
SKM-MEO _k	0,98 ($\pm 0,08$)	0,98 ($\pm 0,08$)	0,99 ($\pm 0,04$)	0,99 ($\pm 0,06$)	0,98 ($\pm 0,07$)	0,99 ($\pm 0,03$)
SKM-MEO _k I	0,98 ($\pm 0,08$)	0,98 ($\pm 0,09$)	0,99 ($\pm 0,04$)	0,99 ($\pm 0,06$)	0,98 ($\pm 0,07$)	0,99 ($\pm 0,04$)
SLS-B _k M	0,96 ($\pm 0,12$)	0,98 ($\pm 0,07$)	0,98 ($\pm 0,11$)	0,98 ($\pm 0,07$)	0,99 ($\pm 0,02$)	1,00 ($\pm 0,00$)
SLS-IB _k M	0,96 ($\pm 0,12$)	0,97 ($\pm 0,09$)	0,95 ($\pm 0,14$)	0,99 ($\pm 0,01$)	0,99 ($\pm 0,04$)	1,00 ($\pm 0,00$)
SLS-MEO _k	0,99 ($\pm 0,04$)	0,99 ($\pm 0,01$)	0,99 ($\pm 0,02$)	0,99 ($\pm 0,01$)	0,99 ($\pm 0,01$)	0,99 ($\pm 0,01$)
SLS-MEO _k I	0,96 ($\pm 0,11$)	0,98 ($\pm 0,04$)	0,98 ($\pm 0,05$)	0,99 ($\pm 0,01$)	0,99 ($\pm 0,04$)	0,99 ($\pm 0,01$)
FEACS-PHT	0,97 ($\pm 0,14$)	0,99 ($\pm 0,03$)	0,99 ($\pm 0,02$)	0,99 ($\pm 0,02$)	1,00 ($\pm 0,01$)	0,99 ($\pm 0,04$)
FEACS-SSI	0,97 ($\pm 0,11$)	0,98 ($\pm 0,07$)	0,98 ($\pm 0,05$)	0,98 ($\pm 0,05$)	0,99 ($\pm 0,05$)	0,99 ($\pm 0,09$)

De maneira análoga à metodologia para obtenção dos resultados na Tabela 5.1, a Tabela 5.2 apresenta os resultados médios de tempo de processamento. Pode-se observar que, em geral, os algoritmos FEACS-PHT e SKM-B_kM são os mais rápidos. O sucesso em termos de tempo de processamento desses algoritmos está relacionado com a quantidade de dados a serem agrupados (100 objetos). Além disso, para o algoritmo FEACS-PHT o motivo do seu sucesso é também devido ao número de vezes que a estimativa do valor de \hat{k} é solicitada (quando uma mudança é detectada com o PHT). Pode-se observar também que os algoritmos baseados no B_kM se mostraram mais rápidos do que os algoritmos baseados no MEO_k, até mesmo nas versões incrementais.

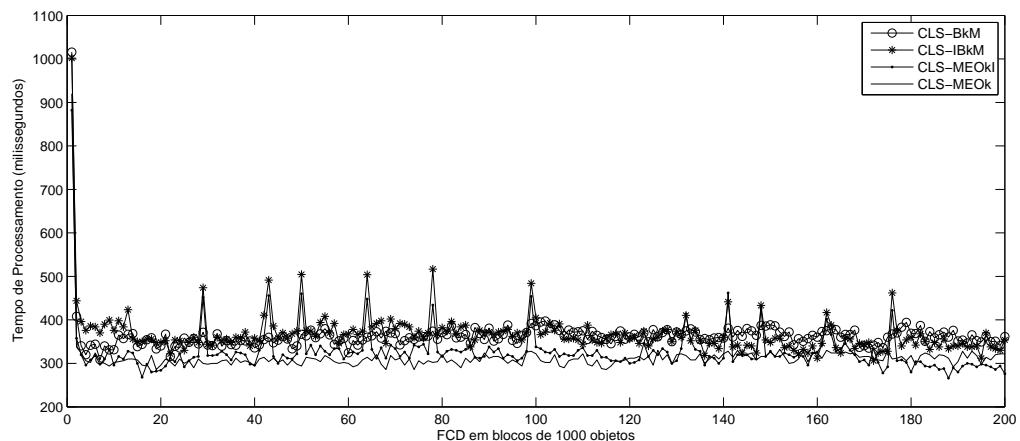
Tabela 5.2: Valores médios de tempo de processamento em milissegundos- Bases de Dados Artificiais. Em negrito estão destacados os menores valores para cada base de dados.

Algoritmo	Base - 2D $\mu(\pm\sigma)$	Base - 4D $\mu(\pm\sigma)$	Base - 8D $\mu(\pm\sigma)$
CLS-B _k M	231,64 (\pm 41,76)	325,08 (\pm 44,64)	590,13 (\pm 69,20)
CLS-IB _k M	227,12 (\pm 51,18)	242,47 (\pm 51,47)	565,81 (\pm 93,85)
CLS-MEO _k	250,75 (\pm 71,48)	368,61 (\pm 43,18)	683,20 (\pm 63,26)
CLS-MEO _k I	245,66 (\pm 49,75)	355,82 (\pm 58,95)	601,28 (\pm 70,98)
SKM-B _k M	27,04 (\pm 19,82)	35,19 (\pm 8,65)	53,08 (\pm 11,88)
SKM-IB _k M	30,06 (\pm 20,76)	48,41 (\pm 10,06)	73,45 (\pm 15,48)
SKM-MEO _k	44,71 (\pm 12,70)	58,2 (\pm 8,10)	83,14 (\pm 10,50)
SKM-MEO _k I	45,20 (\pm 16,03)	55,55 (\pm 13,33)	77,5 (\pm 17,64)
SLS-B _k M	1.018,57 (\pm 955,19)	1.139,61 (\pm 1000,60)	1.678,17 (\pm 1.602,90)
SLS-IB _k M	265,69 (\pm 540,51)	485,144 (\pm 723,70)	425,92 (\pm 583,88)
SLS-MEO _k	2.487,47 (\pm 485,11)	3.296,73 (\pm 410,91)	5.326,53 (\pm 1.188,89)
SLS-MEO _k I	525,72 (\pm 1.094,10)	511,84 (\pm 932,53)	775,38 (\pm 1.555,35)
FEACS-PHT	12,44 (\pm 19,71)	16,44 (\pm 24,21)	29,70 (\pm 39,83)
FEACS-SSI	54,68 (\pm 88,00)	70,11 (\pm 81,70)	143,5 (\pm 172,93)
Algoritmo	Base - 16D $\mu(\pm\sigma)$	Base - 32D $\mu(\pm\sigma)$	Base - 64D $\mu(\pm\sigma)$
CLS-B _k M	1.031,77 (\pm 107,01)	2.068,76 (\pm 274,57)	3.675,80 (\pm 383,22)
CLS-IB _k M	907,35 (\pm 126,98)	1.829,71 (\pm 277,76)	3.263,85 (\pm 415,10)
CLS-MEO _k	1.065,49 (\pm 127,09)	2.194,05 (\pm 264,93)	4.197,68 (\pm 350,30)
CLS-MEO _k I	991,59 (\pm 131,20)	2.124,89 (\pm 285,20)	4.186,15 (\pm 520,04)
SKM-B _k M	90,58 (\pm 17,81)	116,19 (\pm 21,93)	292,45 (\pm 50,63)
SKM-IB _k M	117,45 (\pm 21,84)	229,08 (\pm 49,65)	409,37 (\pm 74,65)
SKM-MEO _k	125,25 (\pm 14,27)	349,09 (\pm 44,97)	774,55 (\pm 68,92)
SKM-MEO _k I	125,07 (\pm 32,56)	158,50 (\pm 52,41)	480,87 (\pm 184,91)
SLS-B _k M	2.821,31 (\pm 2.025,36)	1.895,94 (\pm 1.163,49)	3.425,53 (\pm 2.112,43)
SLS-IB _k M	1.439,67 (\pm 1.151,17)	1.630,72 (\pm 1.102,40)	3.013,58 (\pm 1.779,65)
SLS-MEO _k	11.075,49 (\pm 3.382,68)	13.784,04 (\pm 2.995,95)	71.344,87 (\pm 8.957,05)
SLS-MEO _k I	2.021,29 (\pm 3.593,40)	2.521,52 (\pm 4.836,73)	6.443,60 (\pm 15.995,36)
FEACS-PHT	128,69 (\pm 81,04)	285,92 (\pm 196,04)	303,43 (\pm 183,09)
FEACS-SSI	295,88 (\pm 206,01)	310,68 (\pm 215,82)	1.340,45 (\pm 914,53)

Para analisar os custos computacionais dos algoritmos baseados no CluStream ao longo do FCD para todas as bases de dados artificiais, considere a Figura 5.2, na qual são apresentados os resultados apenas para as bases de dados com 2 e 4 dimensões. É conveniente salientar que os algoritmos apresentaram comportamento similares para as demais bases de dados e, portanto, foram apresentados apenas os resultados para as bases com 2 e 4 dimensões. Com relação à Figura 5.2, pode-se observar que, inicialmente, os algoritmos apresentam um pico nas curvas de custo de processamento devido ao custo para a criação dos microgrupos no início do FCD. O procedimento para a criação dos microgrupos consiste da execução do algoritmo das k -Médias em uma quantidade $Init$ (definido com o valor 1000) de objetos do início do FCD para encontrar q microgrupos (definido com o valor 100). Em seguida, são executadas as etapas de abstração dos dados para atualização dos microgrupos e estimação de \hat{k} a partir dos microgrupos. Os demais picos nas curvas estão relacionados com a execução dos algoritmos para estimação de \hat{k} para as versões incrementais dos algoritmos, que ocorrem com pouca frequência. Por exemplo, para a base de dados bidimensional (Figura 5.2a) a frequência de uso da componente de estimação de \hat{k} para os algoritmos CLS-IB_kM e CLS-MEO_kI é de 12 vezes, ao invés de executá-la a todo momento no FCD.



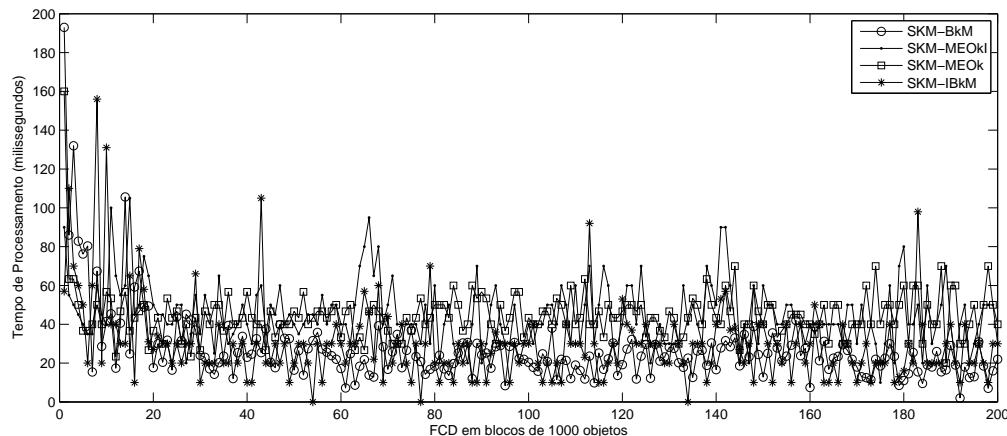
(a) Tempo de processamento (escala logarítmica) dos algoritmos CLS-BkM, CLS-IBkM, CLS-MEOkl e CLS-MEOk - Base de dados artificial com 2 dimensões.



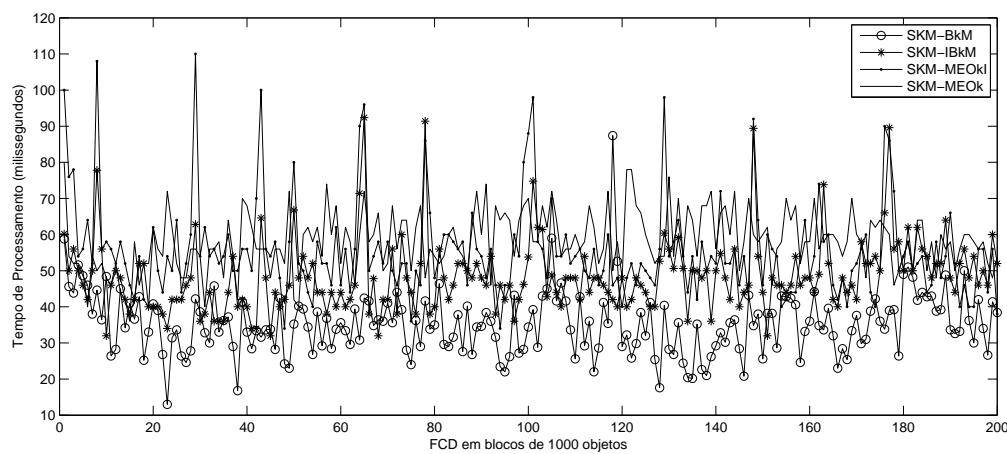
(b) Tempo de processamento dos algoritmos CLS-BkM, CLS-IBkM, CLS-MEOkl e CLS-MEOk - Base de dados artificial com 4 dimensões.

Figura 5.2: Tempo de processamento dos algoritmos CLS-BkM, CLS-IBkM, CLS-MEOkl e CLS-MEOk.

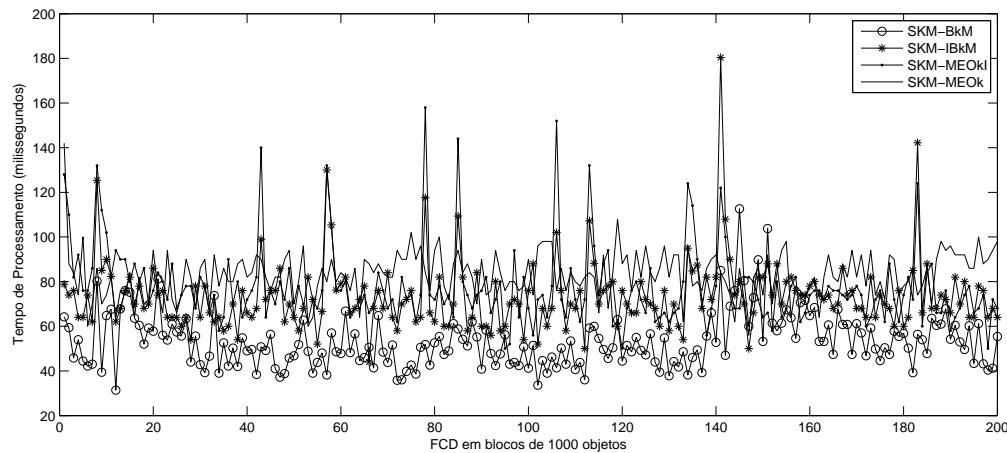
Para os algoritmos baseados no StreamKM++, a Figura 5.3 apresenta os valores médios de tempo de processamento para as bases de dados artificiais com 2, 4 e 8 dimensões. Pode-se observar que os algoritmos têm comportamento similares durante todo o FCD, apresentando resultados competitivos entre si. Pode-se observar também que os menores valores de tempo de processamento são obtidos frequentemente pelo algoritmo SKM-BkM. Os resultados para as demais bases de dados foram omitidos em razão dos algoritmos apresentarem comportamento similares.



(a) Tempo de processamento (escala logarítmica) dos algoritmos SKM-B_kM, SKM-IB_kM, SKM-MEO_k e SKM-MEO_kI - Base de dados artificial com 2 dimensões.



(b) Tempo de processamento (escala logarítmica) dos algoritmos SKM-B_kM, SKM-IB_kM, SKM-MEO_k e SKM-MEO_kI - Base de dados artificial com 4 dimensões.



(c) Tempo de processamento (escala logarítmica) dos algoritmos SKM-B_kM, SKM-IB_kM, SKM-MEO_k e SKM-MEO_kI - Base de dados artificial com 8 dimensões.

Figura 5.3: Tempo de processamento dos algoritmos SKM-B_kM, SKM-IB_kM, SKM-MEO_k e SKM-MEO_kI.

Os resultados obtidos para os algoritmos baseados no *Stream LSearch* são apresentados nas Figuras 5.4, 5.5, 5.6 e 5.7 para as bases de dados com 2, 4, 8, 16, 32 e 64 dimensões, respectivamente. Esses resultados mostram que o algoritmo SLS-IBkM apresentou frequentemente os menores valores de tempo de processamento. Também pode-se observar que as versões incrementais dos algoritmos obtiveram menores valores de tempo de processamento comparativamente aos algoritmos não-incrementais. Essa redução no tempo de processamento dos algoritmos SLS-IBkM e SLS-MEO k I ocorreu devido à redução na frequência de execução da etapa de estimativa de \hat{k} comparada com os algoritmos SLS-BkM e SLS-MEO k , os quais não contêm um método de detecção de mudança, cuja etapa de estimativa de \hat{k} é executada em todo o momento no FCD. Além disso, pode-se observar que na Figura 5.7 as curvas de tempo de processamento dos algoritmos SLS-BkM, SLS-IBkM e SLS-MEO k I apresentam uma variação ao longo do FCD, as quais são proporcionais ao número de grupos encontrados pelos algoritmos, *i.e.*, quanto maior o valor de k maior é o tempo de processamento.

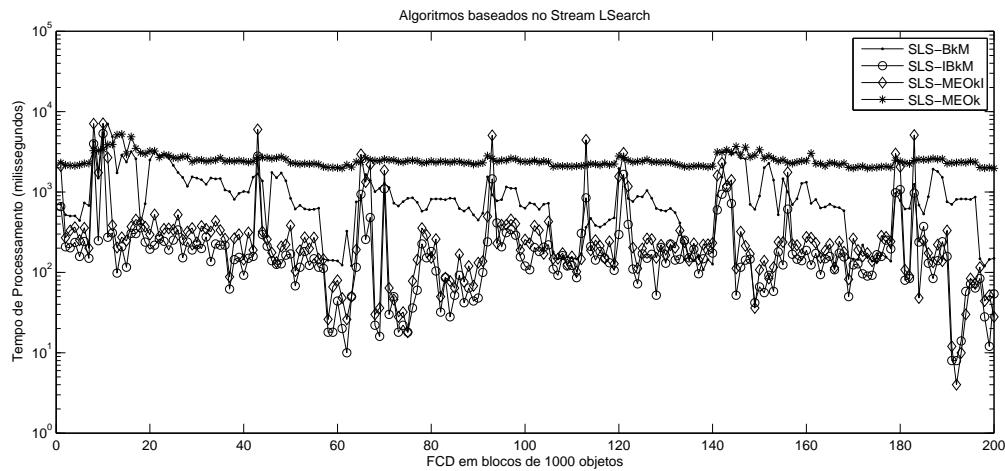


Figura 5.4: Tempo de processamento (escala logarítmica) dos algoritmos SLS-B k M, SLS-IB k M, SLS-MEO k e SLS-MEO k I - Base de dados artificial com 2 dimensões.

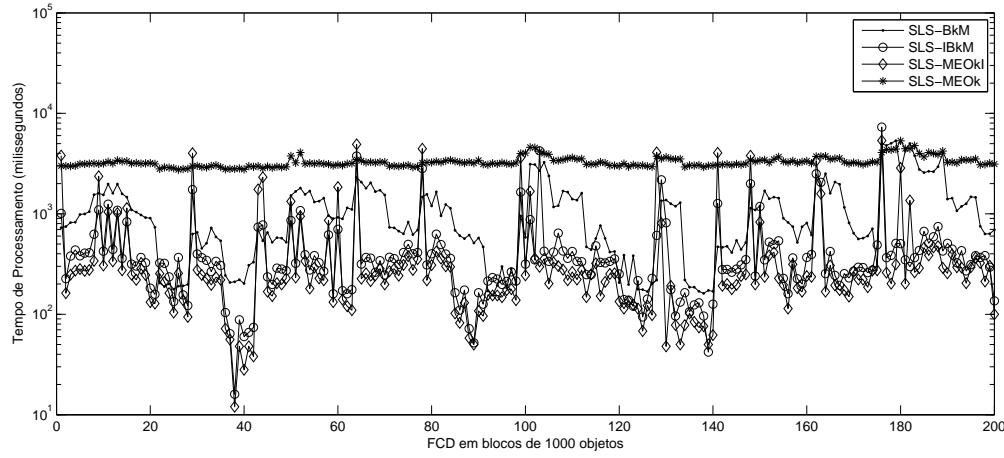


Figura 5.5: Tempo de processamento (escala logarítmica) dos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOk e SLS-MEOkI - Base de dados artificial com 4 dimensões.

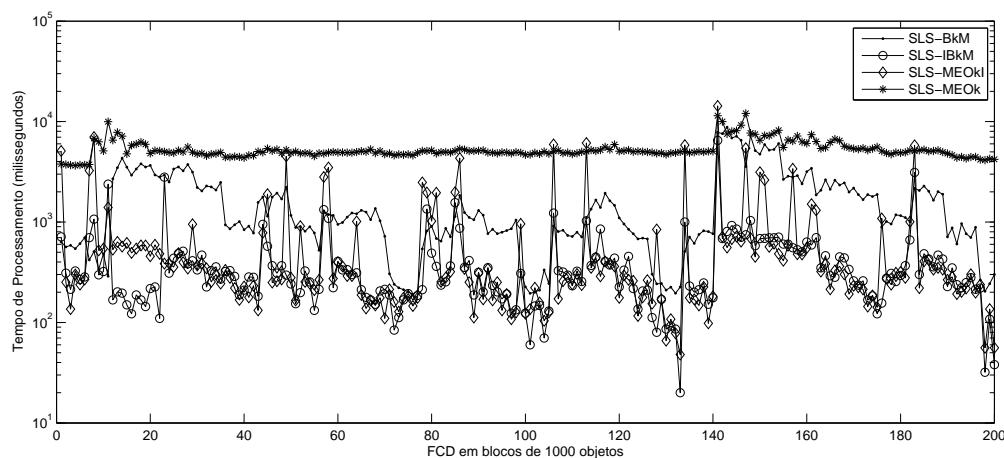
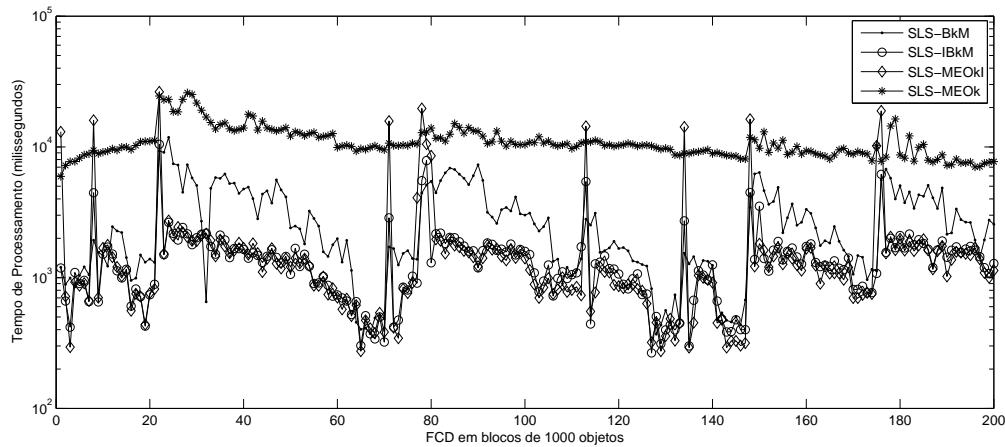
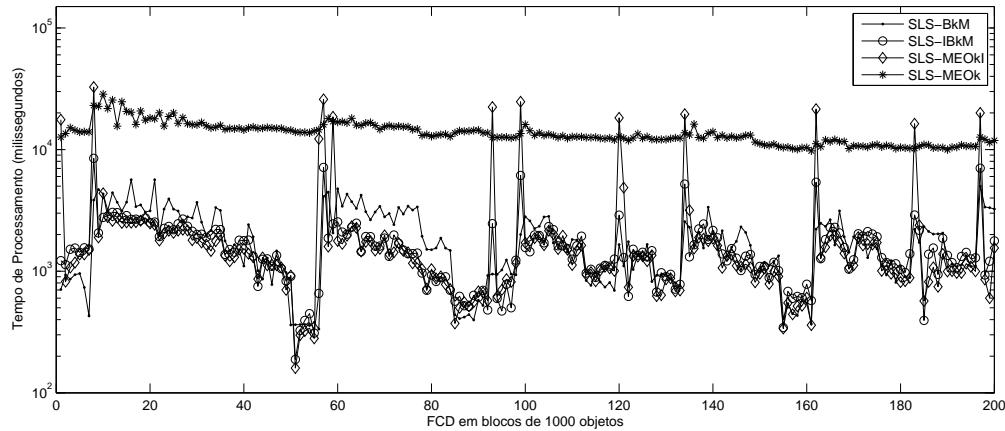


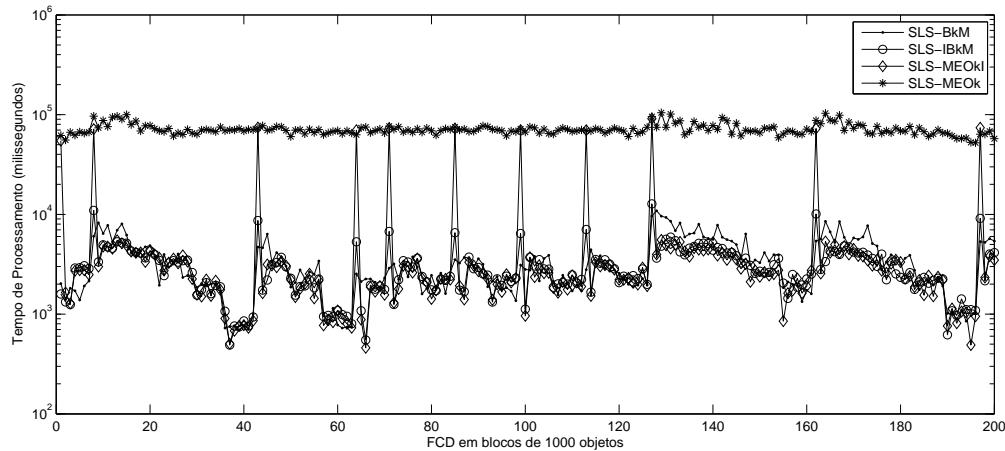
Figura 5.6: Tempo de processamento (escala logarítmica) dos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOk e SLS-MEOkI - Base de dados artificial com 8 dimensões.



(a) Tempo de processamento (escala logarítmica) dos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOkI e SLS-MEOk - Base de dados artificial com 16 dimensões.



(b) Tempo de processamento (escala logarítmica) dos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOk e SLS-MEOkI - Base de dados artificial com 32 dimensões.



(c) Tempo de processamento (escala logarítmica) dos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOk e SLS-MEOkI - Base de dados artificial com 64 dimensões.

Figura 5.7: Tempo de processamento (escala logarítmica) dos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOk e SLS-MEOkI.

Nas Figuras 5.8 e 5.9, pode-se observar que os algoritmos FEACS-PHT e FEACS-SSI têm comportamento similares ao longo do FCD com 2 e 4 dimensões, com um destaque para o algoritmo FEACS-PHT que apresenta os menores valores de tempo de processamento. A diferença dos tempos de processamento dos algoritmos FEACS-PHT e FEACS-SSI está relacionada ao fato de que o FEACS-SSI aplica o algoritmo das k -Médias para encontrar a melhor partição (de acordo com SS) com k e $k + 1$ grupos a todo momento que uma mudança for detectada. No final do processamento do bloco de dados do FCD (a cada 1.000 objetos processados para a base de dados sintética), o FEACS-SSI também executa algoritmo evolutivo para estimativa de \hat{k} como uma medida preventiva de ajustar as partições quando for modificado o número de grupos da partição de k para $k + 1$. Contrariamente, o algoritmo FEACS-PHT executa apenas o algoritmo evolutivo para estimativa de \hat{k} quando uma mudança no PHT for detectada. Além disso, para o algoritmo FEACS-PHT o tamanho do conjunto de objetos entre um sinal de alarme e alerta é definido nesses experimentos com o valor de 100 (10% do tamanho do bloco que é igual 1000, conforme descrito na Seção 5.2). Nesse caso 100 objetos são agrupados. É conveniente antecipar que conclusões semelhantes são obtidas para as demais bases sintéticas com 8, 16, 32 e 64 dimensões. Por conta disso, optou-se por apresentar apenas as análises para as bases com 2 e 4 dimensões.

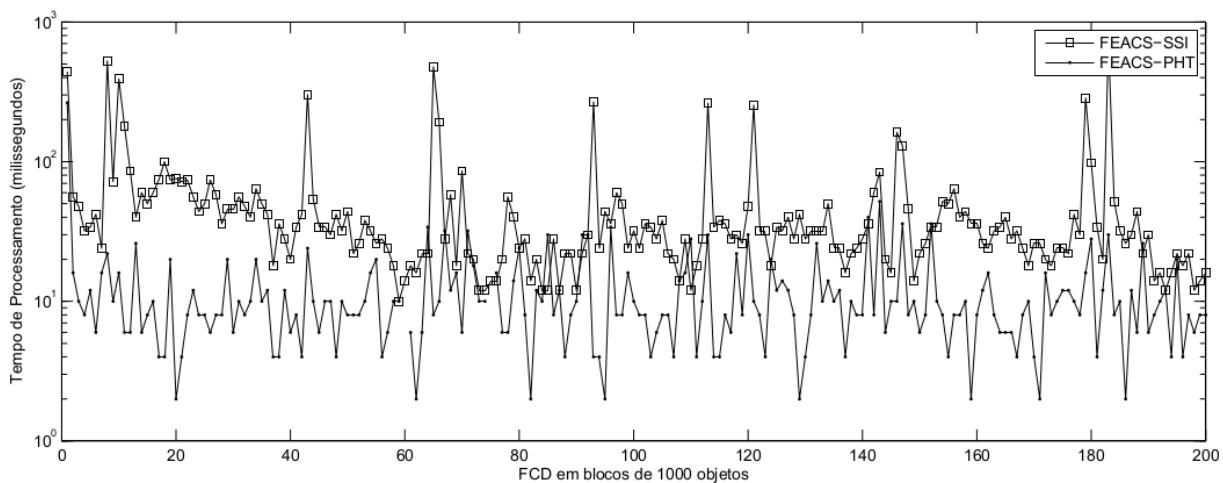


Figura 5.8: Tempo de processamento dos algoritmos FEACS-SSI e FEACS-PHT - Base de dados artificial com 2 dimensões.

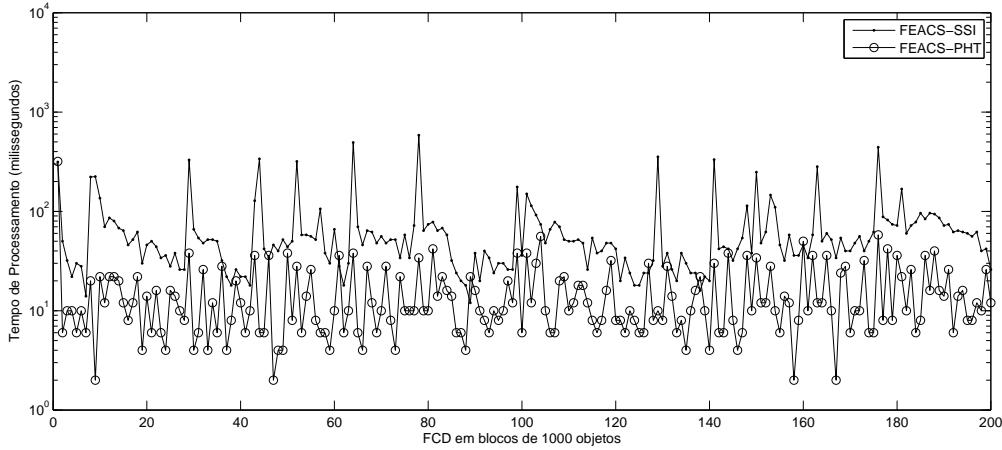
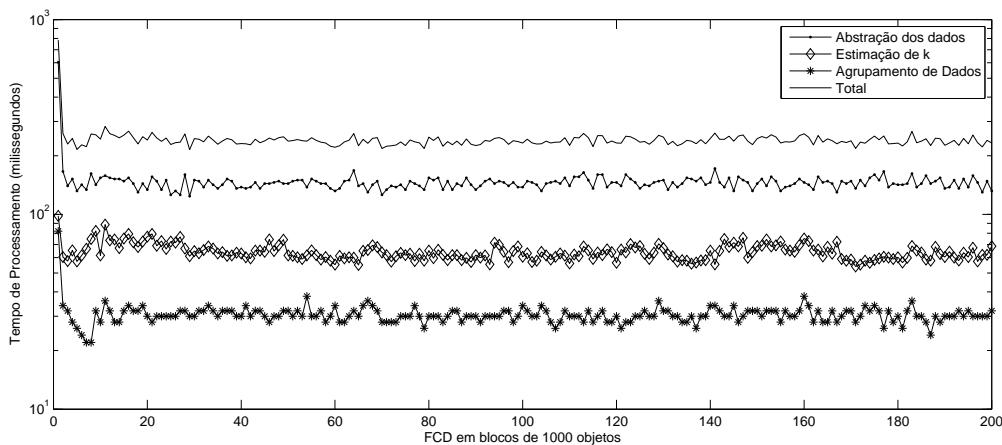


Figura 5.9: Tempo de processamento dos algoritmos FEACS-SSI e FEACS-PHT - Base de dados artificial com 4 dimensões.

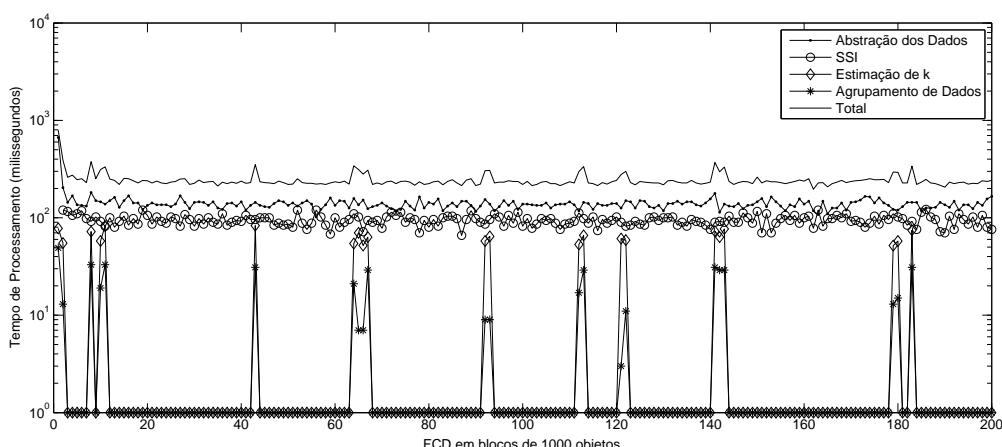
Análise dos Resultados de Tempo de Processamento para Cada Componente dos Algoritmos

Pode-se observar dos resultados reportados na seção anterior que os algoritmos baseados no CluStream apresentaram comportamento similares em todas as bases de dados. Para entender a razão desse comportamento dos algoritmos, até mesmo para as versões baseados na SSI, considere a Figura 5.10 para visualizar os tempos de processamento médios de cada componente para os algoritmos CLS-BkM e CLS-IBkM, e a Figura 5.11 para visualizar os tempos de processamento médios de cada componente para os algoritmos CLS-MEO_k e CLS-MEO_{k|l} para a base de dados com 2 dimensões. Pode-se observar que a componente de abstração de dados dos algoritmos apresenta um custo computacional superior (aproximadamente 140 milissegundos) comparado às demais componentes, sendo significativa no tempo total (soma dos tempos de processamento de cada componente) dos algoritmos. Relembrando da Seção 3.1.2 que a componente de abstração de dados consiste em atualizar os microgrupos inserindo os objetos subsequentes do FCD no microgrupo mais próximo disponível (cujo custo computacional é da ordem de $O(q \times l)$, onde l é o número de atributos) ou criando um novo microgrupo de duas possíveis maneiras: i) remover um microgrupo cujo tempo de atividade média seja menor que um limiar definido pelo usuário, o qual apresenta complexidade computacional da ordem de $O(q)$, ou ii) unir dois microgrupos mais próximos, com complexidade de $O(q^2 \times l)$. Para observar quais dessas operações para atualização dos microgrupos são realizadas frequentemente pelos algoritmos, considere as Figuras 5.12 e 5.13. Pode-se observar que a componente de união de grupos é executada em aproximadamente 40% dos dados de cada bloco, ou seja, dado que cada bloco para a base sintética contém 1.000 objetos, em 400 objetos desse bloco é necessário realizar a operação de união dos microgrupos, sendo que cada operação possui custo computacional quadrático com o número de microgrupos, re-

sultando assim em alto custo computacional para a componente de abstração dos dados quando comparado às demais componentes. É conveniente salientar que para as bases de dados com 4, 8 16, 32 e 64 dimensões os comportamentos de cada componente dos algoritmos foram similares (a etapa de abstração de dados apresentou maiores valores de tempo de processamento devido ao número elevado de operações de união de grupos, aproximadamente 800 operações) e, portanto, foram apresentados apenas os resultados para a base de dados com 2 dimensões.

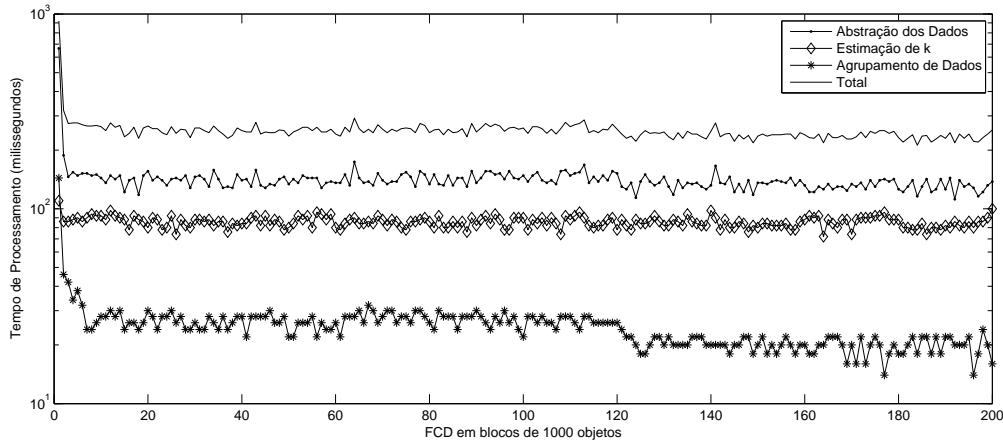


(a) Tempo de processamento de cada componente do algoritmo CLS-BkM - Base de dados artificial com 2 dimensões.

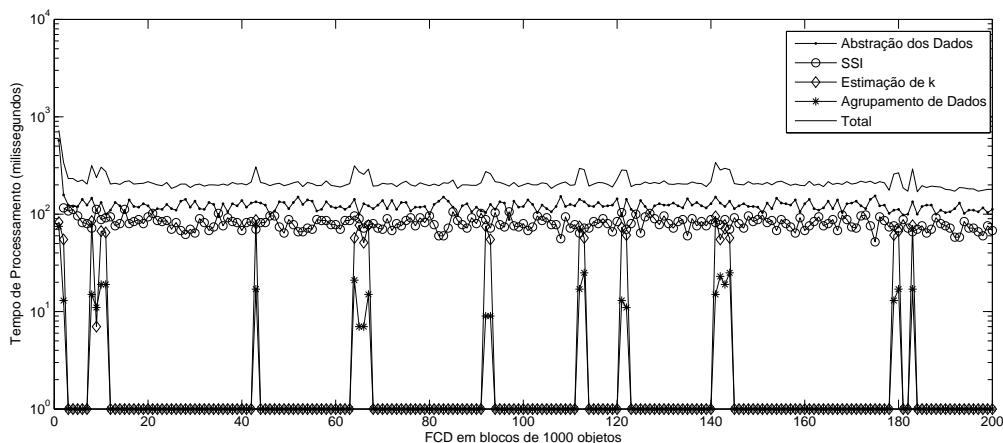


(b) Tempo de processamento de cada componente do algoritmo CLS-IBkM - Base de dados artificial com 2 dimensões.

Figura 5.10: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (ssi), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos CLS-BkM (Figura 5.10a) e CLS-IBkM (Figura 5.10b).

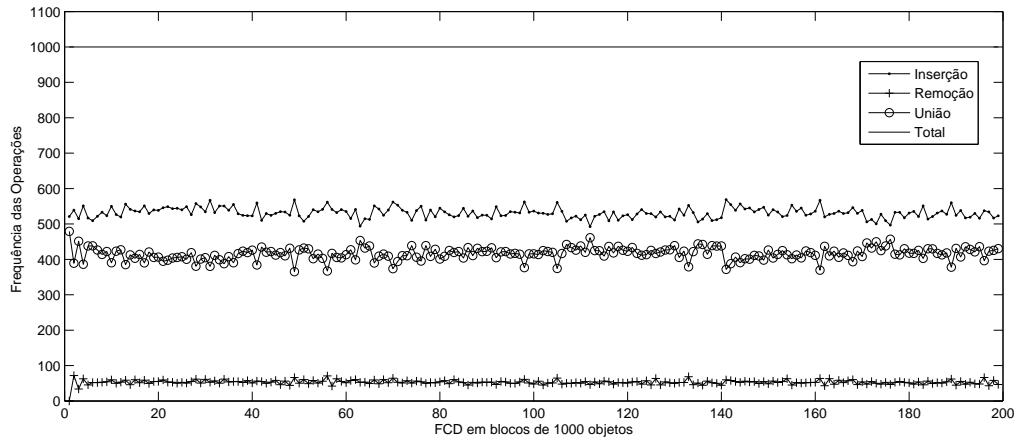


(a) Tempo de processamento de cada componente do algoritmo CLS-MEO k - Base de dados artificial com 2 dimensões.

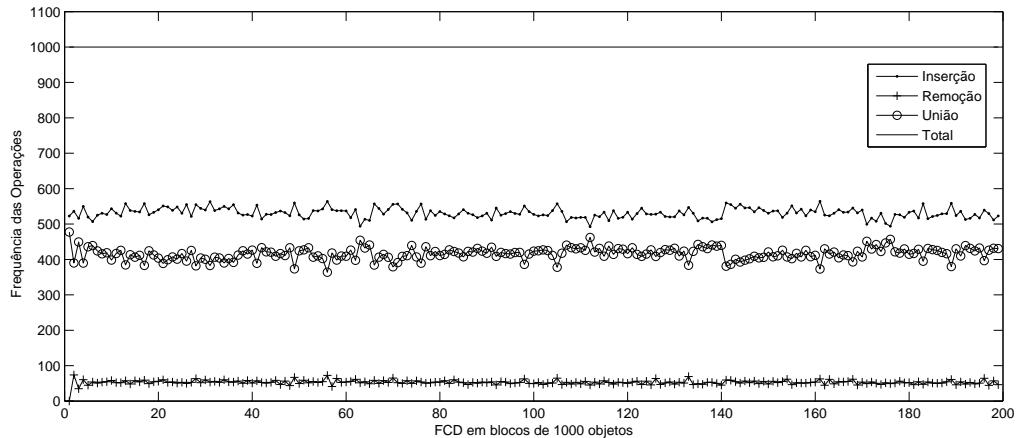


(b) Tempo de processamento de cada componente do algoritmo CLS-MEO k I - Base de dados artificial com 2 dimensões.

Figura 5.11: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos CLS-MEO k (Figura 5.11a) e CLS-MEO k I (Figura 5.11b).

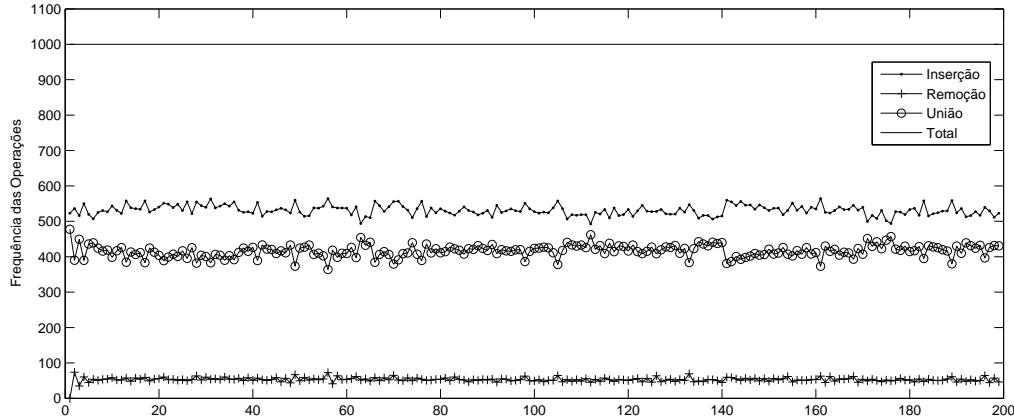


(a) Frequência de cada operação de inserção, remoção e união de microgrupos para o algoritmo CLS-B_kM - Base de dados artificial com 2 dimensões.

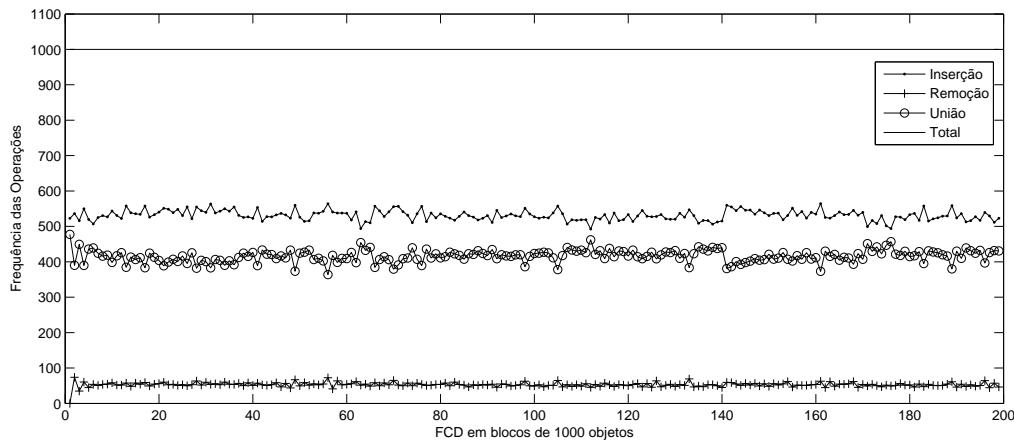


(b) Frequência de cada operação de inserção, remoção e união de microgrupos para o algoritmo CLS-IB_kM - Base de dados artificial com 2 dimensões.

Figura 5.12: Frequência de cada operação de inserção, remoção e união de microgrupos e sua soma (total) para os algoritmos CLS-B_kM (Figura 5.12a) e CLS-IB_kM (Figura 5.12b).



(a) Frequência de cada operação de inserção, remoção e união de microgrupos para o algoritmo CLS-MEO k
- Base de dados artificial com 2 dimensões.



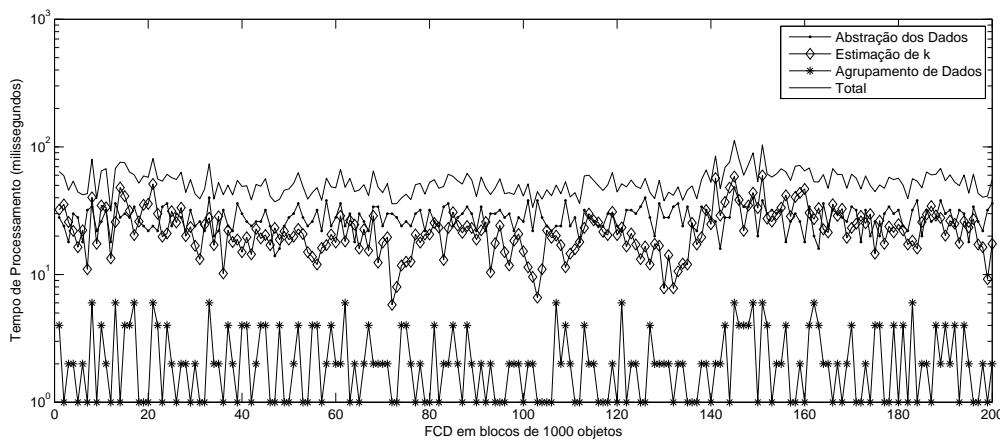
(b) Frequência de cada operação de inserção, remoção e união de microgrupos para o algoritmo CLS-MEO kl
- Base de dados artificial com 2 dimensões.

Figura 5.13: Frequência de cada operação de inserção, remoção e união de microgrupos e sua soma (total) para os algoritmos CLS-MEO k (Figura 5.13a) e CLS-MEO kl (Figura 5.13b).

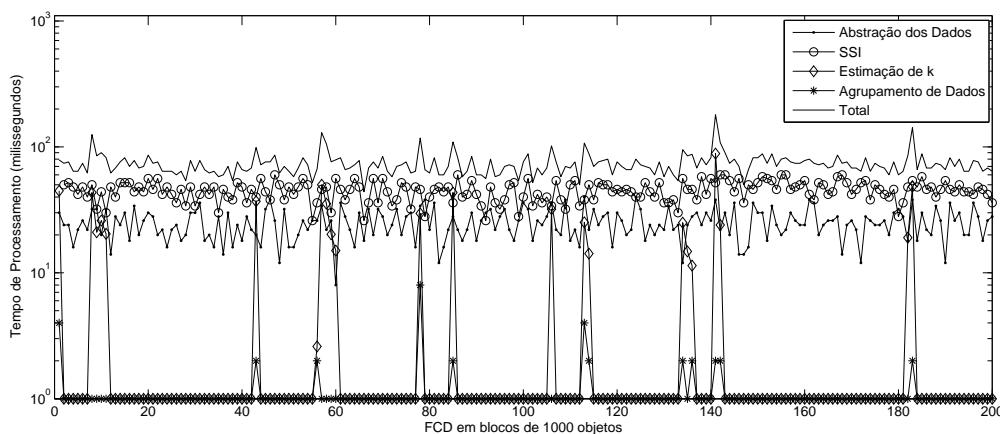
Na Figura 5.14 são apresentados os tempos de processamento médios de cada componente para os algoritmos SKM-B k M e SKM-IB k M e na Figura 5.15 são apresentados os resultados de cada componente para os algoritmos SKM-MEO k e SKM-MEO kl , ambos para a base de dados com 8 dimensões. Esta base foi selecionada dentre as demais para facilitar a visualização dos resultados para cada componente. É importante salientar que as conclusões para as demais bases de dados foram similares e por esse motivo foram omitidas.

Pode-se observar nas Figuras 5.14b e 5.15b que para os algoritmos SKM-IB k M e SKM-MEO kl a componente de estimativa de k e a componente de agrupamento de dados são utilizadas com pouca frequência no FCD. Além disso, para os algoritmos com método de estimativa de \hat{k} baseados no B k M o custo de processamento dessa componente é inferior

ao da componente SSI, enquanto que os algoritmos com método de estimativa de \hat{k} com o algoritmo MEO k apresentam custo computacional superior à componente SSI. A razão para essa diferença está no fato de que o algoritmo BkM é mais rápido do que o algoritmo MEO k .

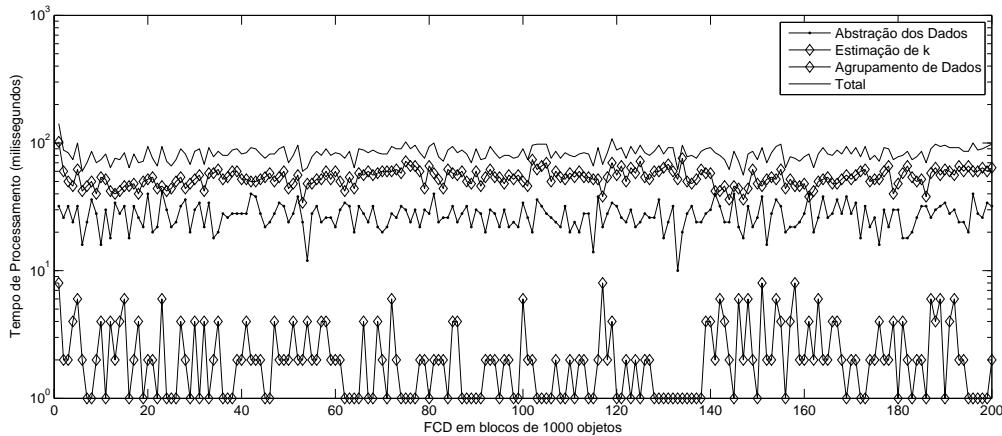


(a) Tempo de processamento de cada componente do algoritmo SKM-B k M - Base de dados artificial com 8 dimensões.

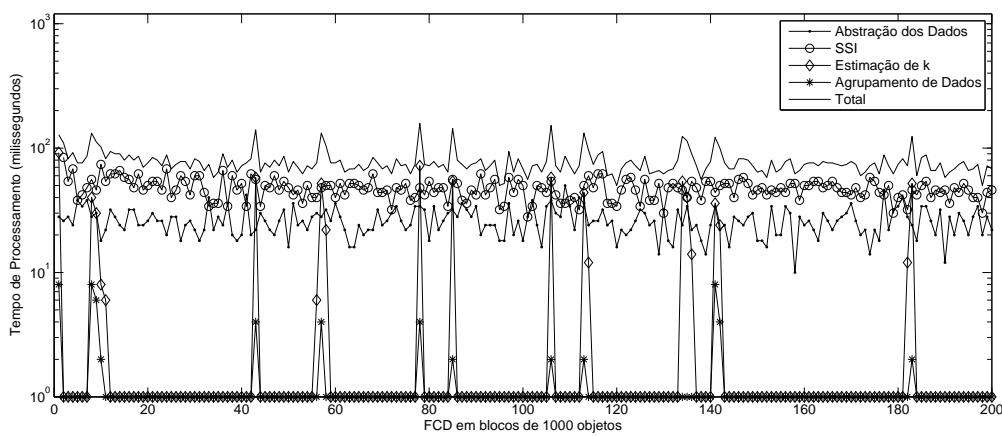


(b) Tempo de processamento de cada componente do algoritmo SKM-IB k M - Base de dados artificial com 8 dimensões.

Figura 5.14: Tempo de processamento em milissegundos de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) para os algoritmos SKM-B k M (Figura 5.14a) e SKM-IB k M (Figura 5.14b).



(a) Tempo de processamento de cada componente do algoritmo SKM-MEO k - Base de dados artificial com 8 dimensões.



(b) Tempo de processamento de cada componente do algoritmo SKM-MEO k I - Base de dados artificial com 8 dimensões.

Figura 5.15: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (ssi), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos SKM-MEO k (Figura 5.15a) e SKM-MEO k I (Figura 5.15b).

Na Figura 5.16 são apresentados os tempos de processamento médios de cada componente para os algoritmos SLS-B k M e SLS-IB k M e na Figura 5.17 são apresentados os tempos de processamentos médios de cada componente para os algoritmos SLS-MEO k e SLS-MEO k I. Pode-se observar na Figura 5.16a que a etapa de estimativa de \hat{k} apresenta um custo computacional inferior ao da etapa de agrupamento de dados para o algoritmo SLS-B k M. Porém, para o algoritmo SLS-MEO k (ver Figura 5.17a) o custo da etapa de estimativa de \hat{k} é superior ao da etapa de agrupamento de dados. Nesses dois algoritmos, o custo da etapa de agrupamento de dados é similar. Porém, o custo das componentes de estimativa de \hat{k} dos dois algoritmos diferem, de modo que a versão B k M é mais rápida que a versão MEO k , resultando nessa diferença observada nos tempos de processamento. Considerando os algoritmos SLS-IB k M e SLS-MEO k I, pode-se notar uma redução

de custo das etapas de estimativa de \hat{k} e de agrupamento de dados, sendo estas as etapas mais custosas dos algoritmos. Adicionalmente, tem-se o custo do uso da SSI para detecção de mudanças que para os algoritmos é de aproximadamente 140 milissegundos. A componente SSI aplica o algoritmo das k -Médias com o número de grupos igual a k e $k + 1$ no bloco de dados, neste caso com 1000 objetos. Para essa quantidade de objetos, pode-se notar a diferença entre o custo para a estimativa de \hat{k} e a SSI. Entretanto, para os algoritmos baseados no *StreamKM++*, os custos são semelhantes para a estimativa de \hat{k} e a SSI para o caso com 100 objetos. Além disso, pode-se notar o ganho no tempo de processamento para as versões incrementais (SLS-IB k M e SLS-MEO k) com relação aos algoritmos SLS-B k M e SLS-MEO k . Conclusões similares foram obtidas para as demais bases de dados e portanto foram apresentados apenas os resultados para base de dados com duas dimensões.

Nas Figuras 5.18 e 5.19 são apresentados os custos computacionais de cada componente dos algoritmos FEACS-SSI e FEACS-PHT para a base de dados com duas dimensões, respectivamente. Pode-se observar que as componentes de estimativa de \hat{k} são utilizadas em poucos momentos do FCD. Além disso, pode-se observar pelos picos na curva da componente PHT que o algoritmo FEACS-PHT utiliza com maior frequência a etapa de estimativa de \hat{k} do que o algoritmo FEACS-SSI. Nas Figuras 5.20 e 5.21 são apresentados os resultados dos algoritmos FEACS-SSI e FEACS-PHT para a base de dados com 16 dimensões. Pode-se observar que as variações nas componentes de estimativa de \hat{k} dos dois algoritmos têm comportamentos similares, os quais estão diretamente relacionados com a variação do número de grupos encontrado pelos algoritmos. A Figura 5.18 também permite observar esse comportamento.

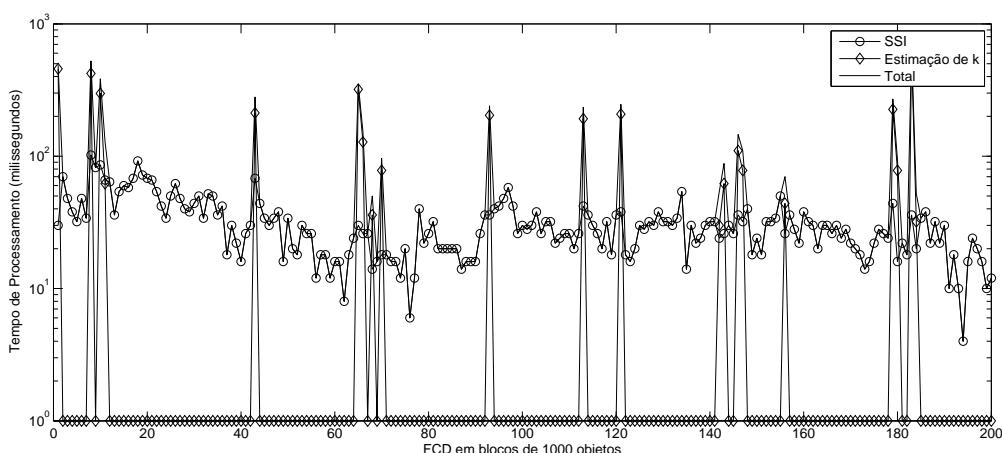
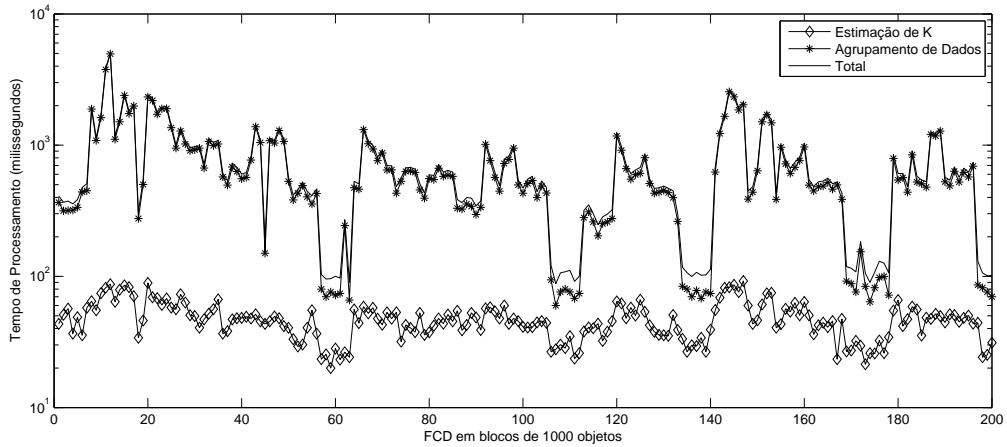
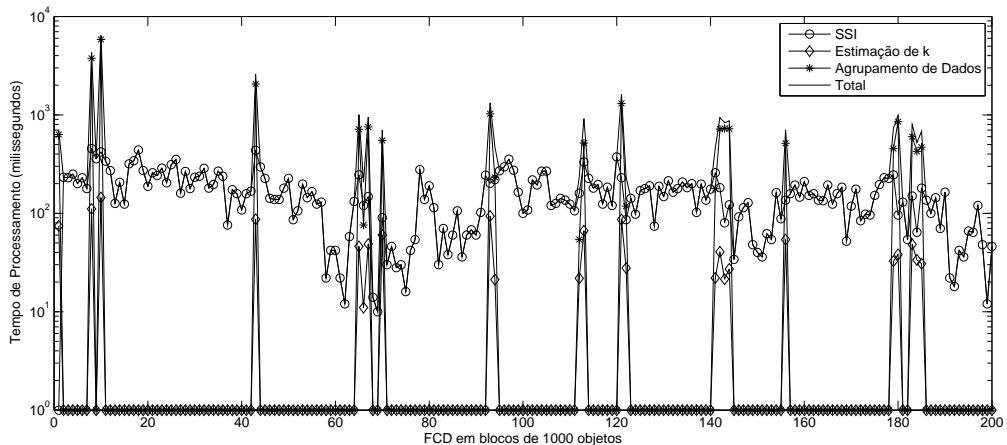


Figura 5.18: Tempo de processamento de cada componente do algoritmo FEACS-SSI - Base de dados artificial com 2 dimensões.

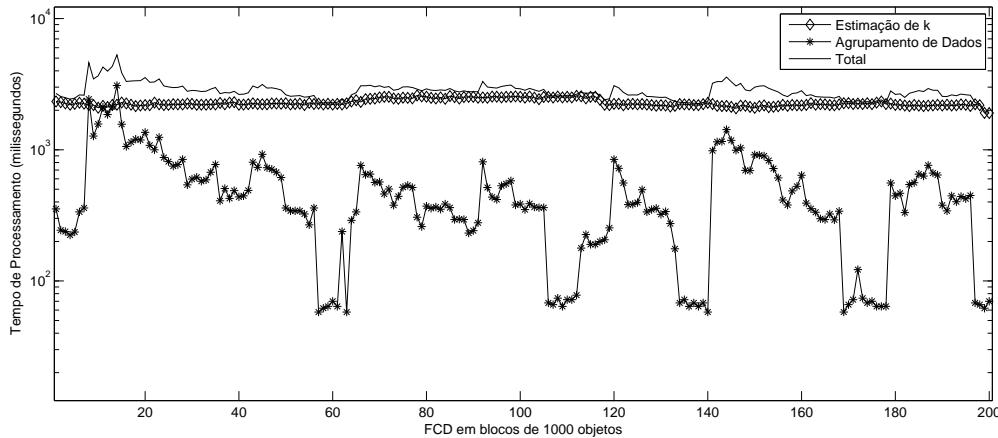


(a) Tempo de processamento de cada componente do algoritmo SLS-B k M - Base de dados artificial com 2 dimensões.

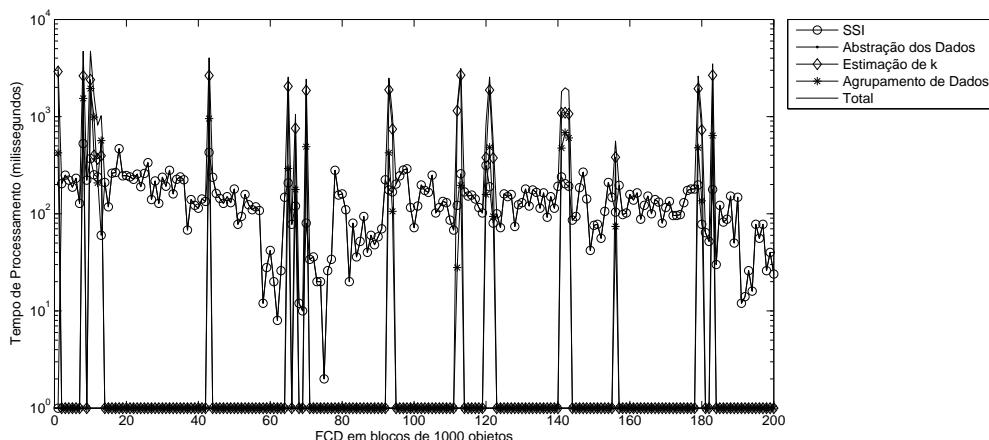


(b) Tempo de processamento de cada componente do algoritmo SLS-IB k M - Base de dados artificial com 2 dimensões.

Figura 5.16: Tempo de processamento em milissegundos de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) para os algoritmos SLS-B k M (Figura 5.16a) e SLS-IB k M (Figura 5.16b) - para a base de dados artificial com 2 dimensões.



(a) Tempo de processamento de cada componente do algoritmo SLS-MEO k - Base de dados artificial com 2 dimensões.



(b) Tempo de processamento de cada componente do algoritmo SLS-MEO k I - Base de dados artificial com 2 dimensões.

Figura 5.17: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimação de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos SLS-MEO k (Figura 5.17a) e SLS-MEO k I (Figura 5.17b) - Base de dados artificial com 2 dimensões.

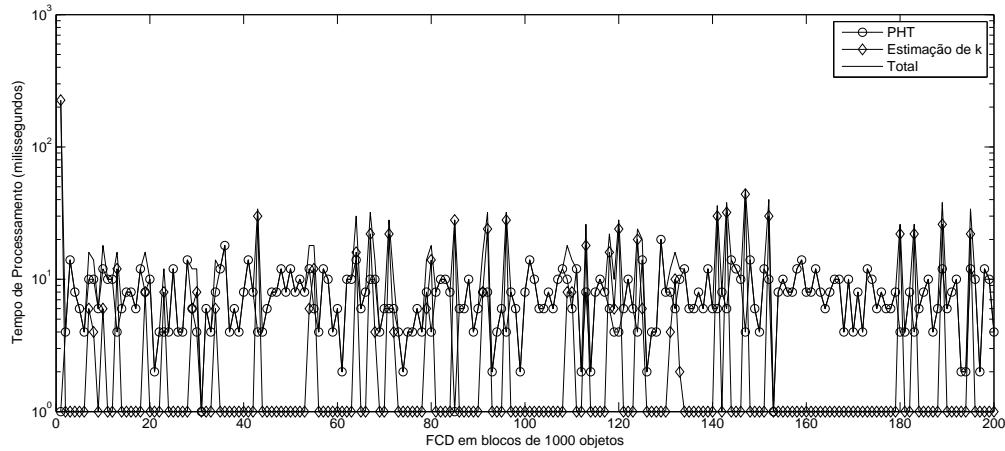


Figura 5.19: Tempo de processamento de cada componente do algoritmo FEACS-PHT - Base de dados artificial com 2 dimensões.

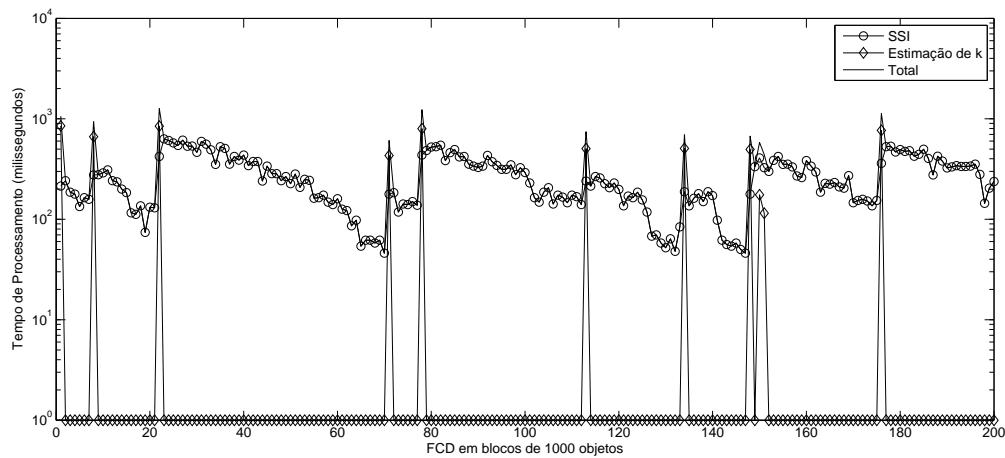


Figura 5.20: Tempo de processamento de cada componente do algoritmo FEACS-SSI - Base de dados artificial com 16 dimensões.

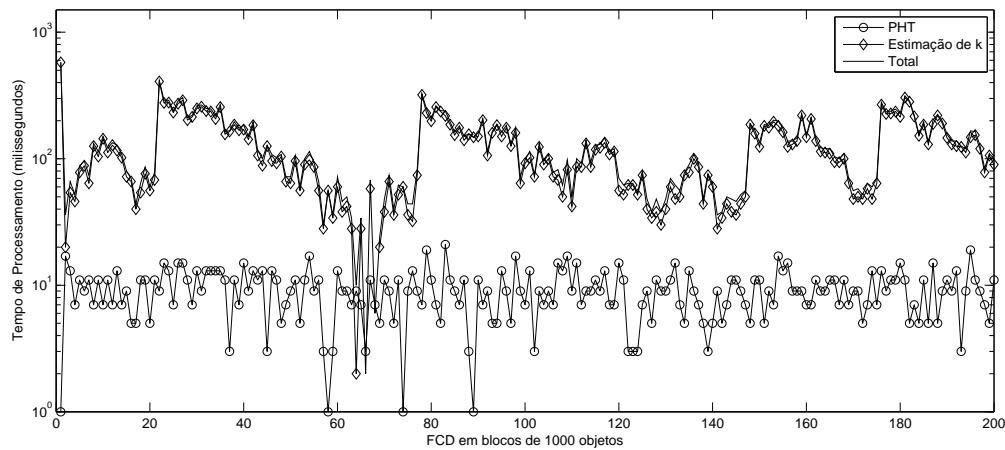


Figura 5.21: Tempo de processamento de cada componente do algoritmo FEACS-PHT - Base de dados artificial com 16 dimensões.

Análise da Sensibilidade dos Algoritmos

Para analisar a sensibilidade dos algoritmos baseados no CluStream para detectar e reagir a mudanças no FCD artificial, nas Figuras 5.22, 5.23 e 5.24 são apresentados os resultados para os números de grupos encontrados pelos algoritmos no decorrer do fluxo dos dados. Esses resultados são provenientes da média de 10 execuções dos algoritmos. Pode-se observar que os algoritmos, na maioria dos casos, reagem às mudanças de número de grupos, e frequentemente encontram o valor correto de k . Especificamente, a menor taxa de acerto para o valor de k foi obtida pelo algoritmo CLS-IBkM, com 77% (base de dados com duas dimensões) e a maior taxa foi obtida pelo algoritmo CLS-MEO k com 97% para a base de dados com oito dimensões.

Na Tabela 5.3 são apresentadas as taxas de acertos dos algoritmos para cada base de dados. Em geral, os algoritmos que utilizam o MEO k obtiveram taxas de acerto variando de 90% a 97% enquanto que os algoritmos que utilizam o BkM obtiveram taxas de acerto variando de 85% a 94% *i.e.*, as taxas de acerto para os algoritmos que utilizam o MEO k são maiores do que com o algoritmo de estimação de \hat{k} via BkM. Além disso, os altos valores para a taxa de acerto se refletem na alta qualidade das partições obtidas, conforme observado anteriormente na Tabela 5.1.

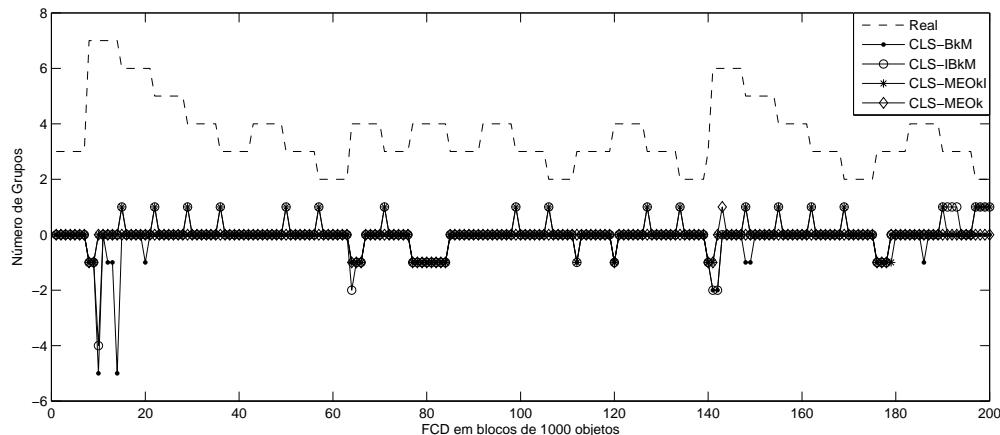
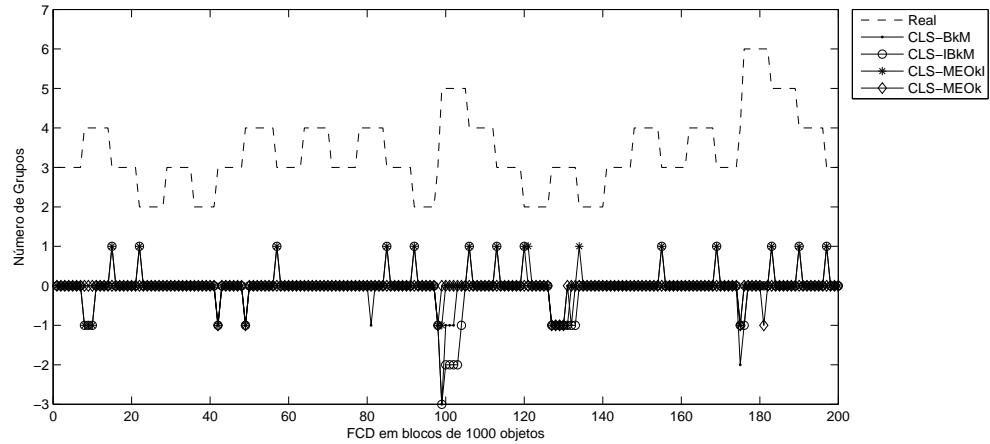
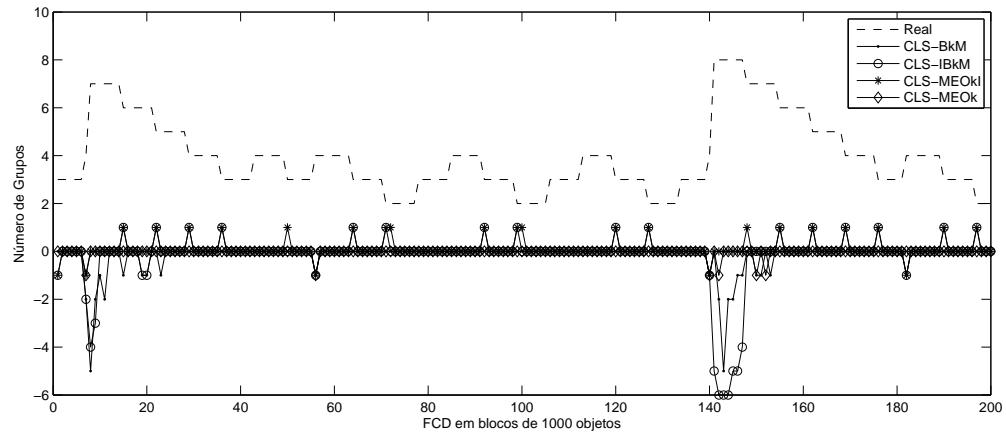


Figura 5.22: Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos CLS-BkM, CLS-IBkM, CLS-MEO k e CLS-MEO k I - Base de dados artificial com 2 dimensões. Valores positivos indicam que o valor de \hat{k} foi superestimado e valores negativos indicam que o número de grupos foi subestimado.

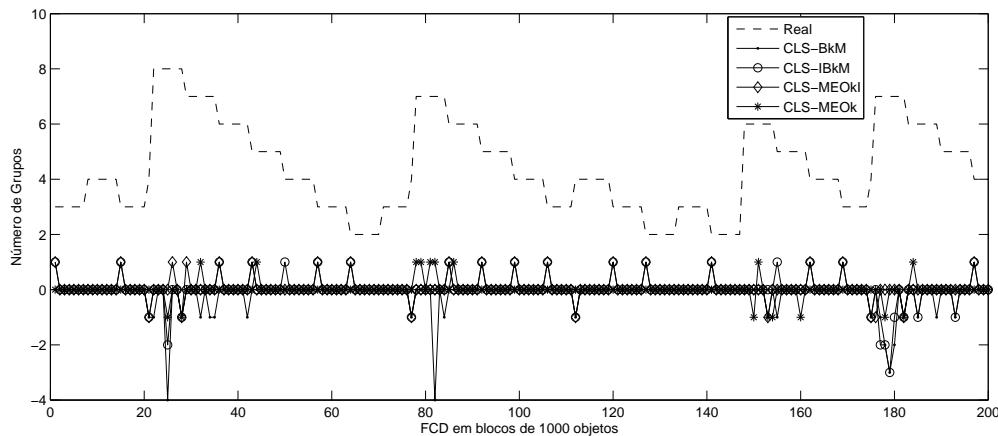


(a) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos CLS-BkM, CLS-IBkM, CLS-MEOk e CLS-MEOkl - Base de dados artificial com 4 dimensões.

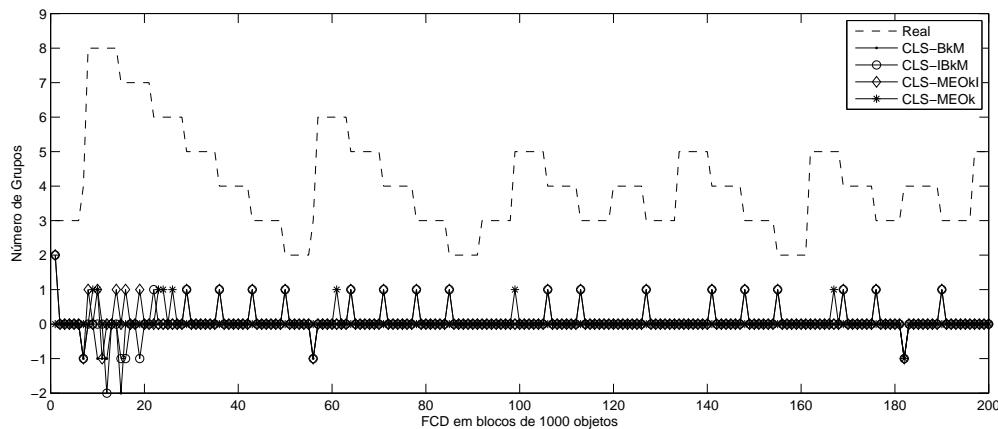


(b) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos CLS-BkM, CLS-IBkM, CLS-MEOk e CLS-MEOkl - Base de dados artificial com 8 dimensões.

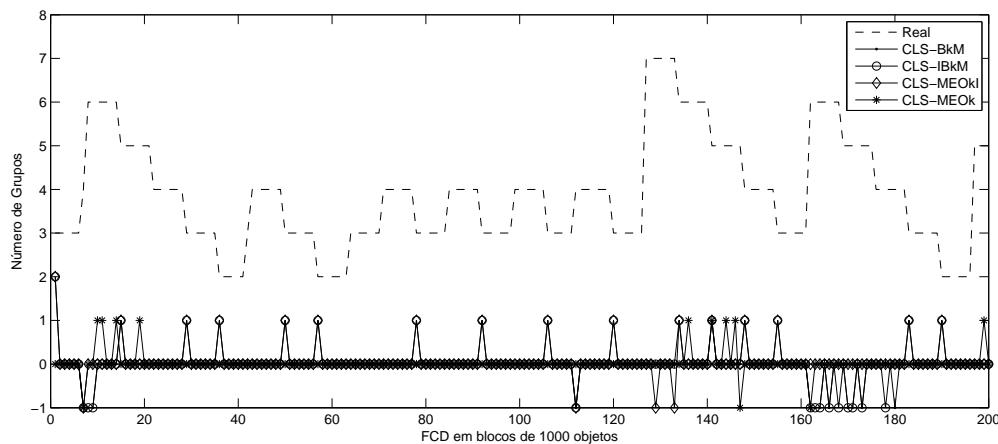
Figura 5.23: Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos CLS-BkM, CLS-IBkM, CLS-MEOk e CLS-MEOkl para as bases de dados artificiais com 4 e 8 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k .



(a) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos CLS-B_kM, CLS-IB_kM, CLS-MEO_k e CLS-MEO_kI - Base de dados artificial com 16 dimensões.



(b) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos CLS-B_kM, CLS-IB_kM, CLS-MEO_k e CLS-MEO_kI - Base de dados artificial com 32 dimensões.



(c) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos CLS-B_kM, CLS-IB_kM, CLS-MEO_k e CLS-MEO_kI - Base de dados artificial com 64 dimensões.

Figura 5.24: Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos CLS-B_kM, CLS-IB_kM, CLS-MEO_k e CLS-MEO_kI para as bases de dados artificiais com 16, 32 e 64 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k .

Tabela 5.3: Taxas de acerto do número correto de grupos para os algoritmos baseados no CluStream - Bases de Dados Artificiais.

Algoritmo	Base - 2D	Base - 4D	Base - 8D	Base - 16D	Base - 32D	Base - 64D
CLS-BkM	85%	92%	89%	88%	96%	94%
CLS-IBkM	77%	83%	84%	84%	87%	86%
CLS-MEOkl	79%	86%	88%	87%	87%	90%
CLS-MEOk	90%	95%	97%	92%	96%	94%

Para analisar a sensibilidade dos algoritmos baseados no StreamKM++ para detectar e reagir a mudanças no FCD artificial, nas Figuras 5.25, 5.26 e 5.27 são apresentados os resultados para os números de grupos encontrados pelos algoritmos no decorrer do fluxo dos dados artificiais para duas dimensões. Pode-se observar que os algoritmos, na maioria dos casos, reagem às mudanças de número de grupos, e frequentemente encontram o valor correto de k . Especificamente, a menor taxa de acerto para o valor de k foi obtida pelo algoritmo SKM-BkM (72% para a base de dados com duas dimensões) e a maior taxa foi obtida pelo algoritmo SKM-MEOkl (97%), conforme pode-se observar na Tabela 5.4.

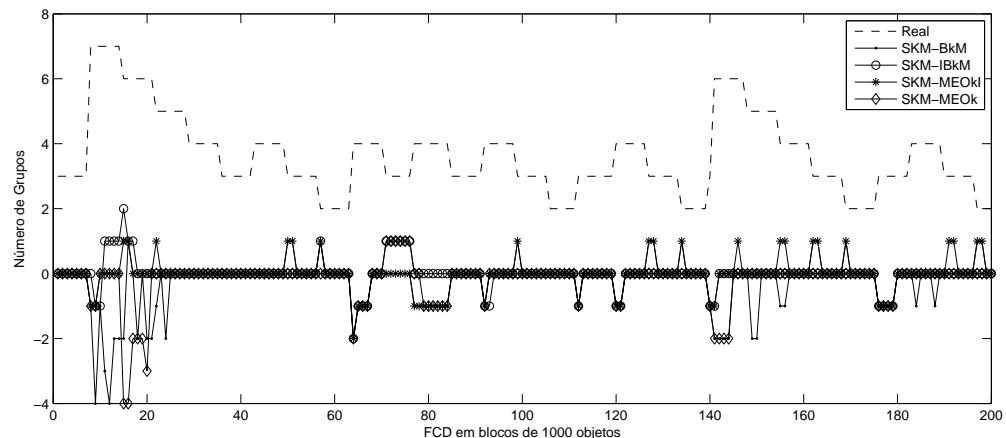
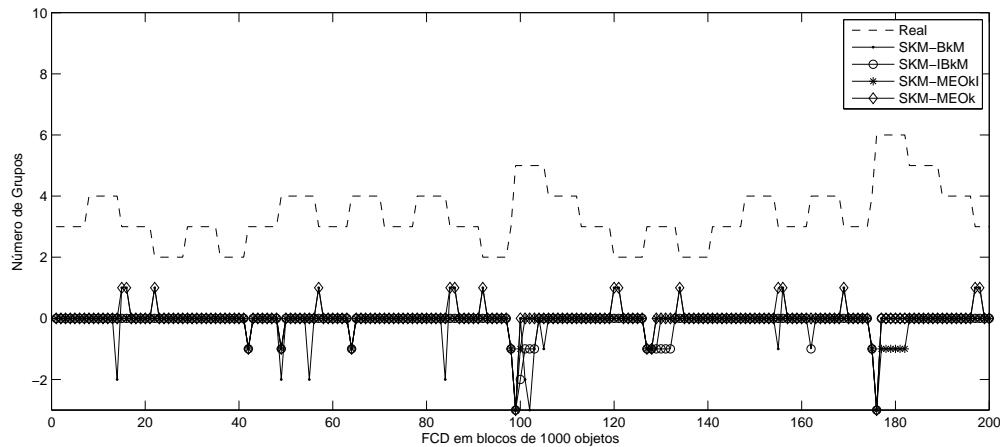
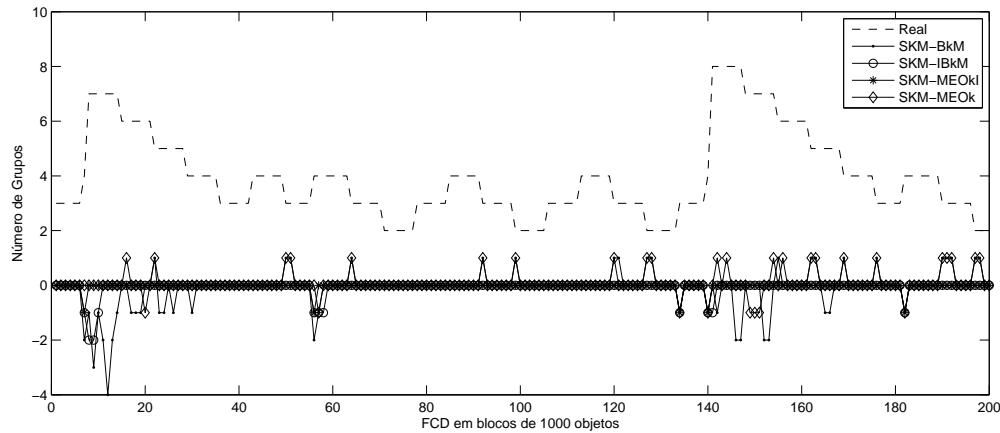


Figura 5.25: Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SKM-BkM, SKM-IBkM, SKM-MEOk e SKM-MEOkl para a base de dados artificial bidimensional. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k .

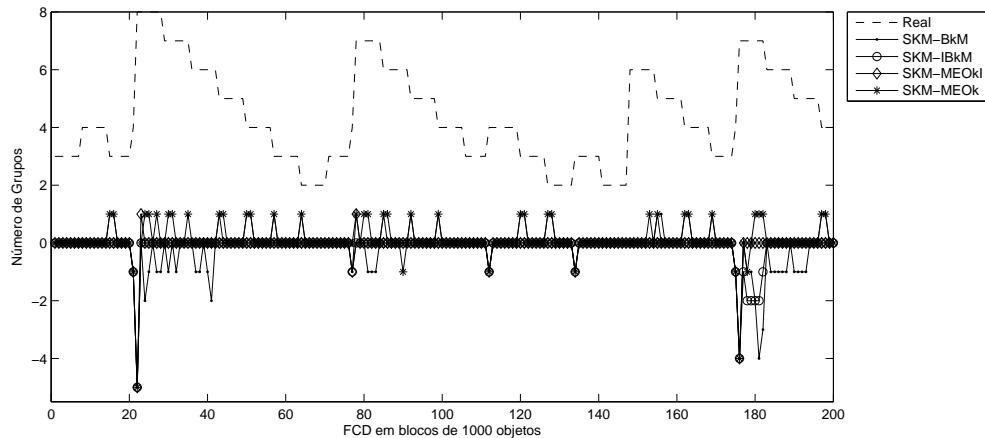


(a) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SKM-B_kM, SKM-IB_kM, SKM-MEO_k e SKM-MEO_kI - Base de dados artificial com 4 dimensões.

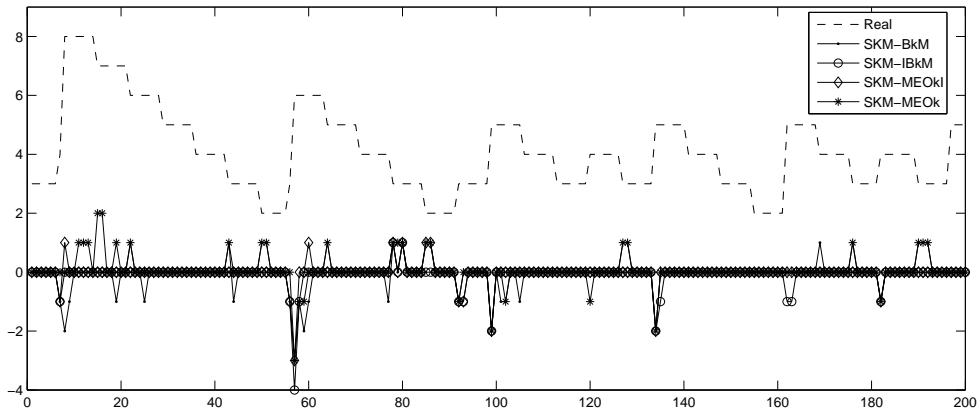


(b) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SKM-B_kM, SKM-IB_kM, SKM-MEO_k e SKM-MEO_kI - Base de dados artificial com 8 dimensões.

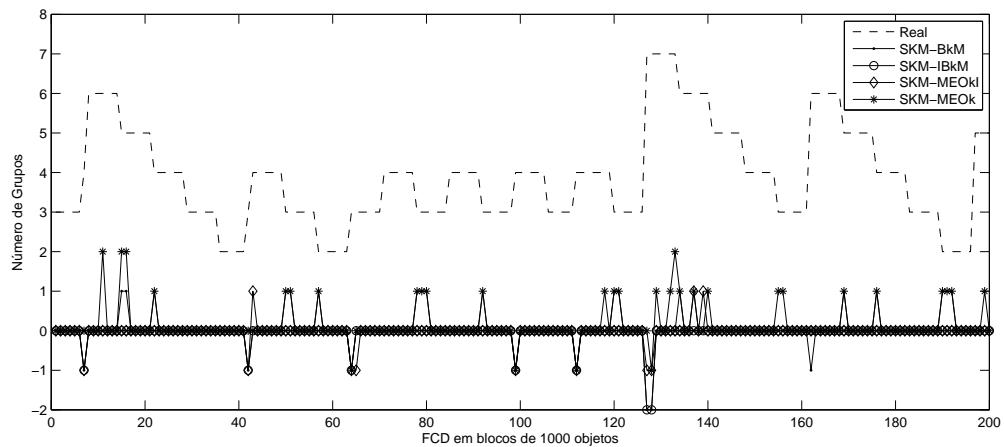
Figura 5.26: Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SKM-B_kM, SKM-IB_kM, SKM-MEO_k e SKM-MEO_kI para as bases de dados artificiais com 4 e 8 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k .



(a) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SKM-BkM, SKM-IBkM, SKM-MEOkI e SKM-MEOk - Base de dados artificial com 16 dimensões.



(b) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SKM-BkM, SKM-IBkM, SKM-MEOkI e SKM-MEOk - Base de dados artificial com 32 dimensões.



(c) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SKM-BkM, SKM-IBkM, SKM-MEOkI e SKM-MEOk - Base de dados artificial com 64 dimensões.

Figura 5.27: Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SKM-BkM, SKM-IBkM, SKM-MEOkI e SKM-MEOk para as bases de dados artificiais com 16, 32 e 64 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k .

Tabela 5.4: Taxas de acerto do número correto de grupos para os algoritmos baseados no StreamKM++ - Bases de Dados Artificiais.

Algoritmo	Base - 2D	Base - 4D	Base - 8D	Base - 16D	Base - 32D	Base - 64D
SKM-BkM	72%	84%	76%	73%	82%	87%
SKM-IBkM	84%	91%	93%	94%	93%	96%
SKM-MEOkl	78%	92%	97%	95%	93%	97%
SKM-MEOk	81%	88%	85%	80%	85%	44%

Com relação ao algoritmo *Stream LSearch*, nas Figuras 5.28, 5.29 e 5.30 são apresentados os resultados para o números de grupos encontrados pelos algoritmos no decorrer dos fluxos. Pode-se observar também que os algoritmos, na maioria dos casos, reagem às mudanças de número de grupos, e frequentemente encontram o valor correto de k . Especificamente, a menor taxa de acerto para o valor de k foi obtida pelo algoritmo SLS-IBkM (70% na base de dados com duas dimensões) e a maior taxa foi obtida pelo algoritmo SLS-IBkM (99% na base de dados com 64 dimensões), conforme apresentado na Tabela 5.5.

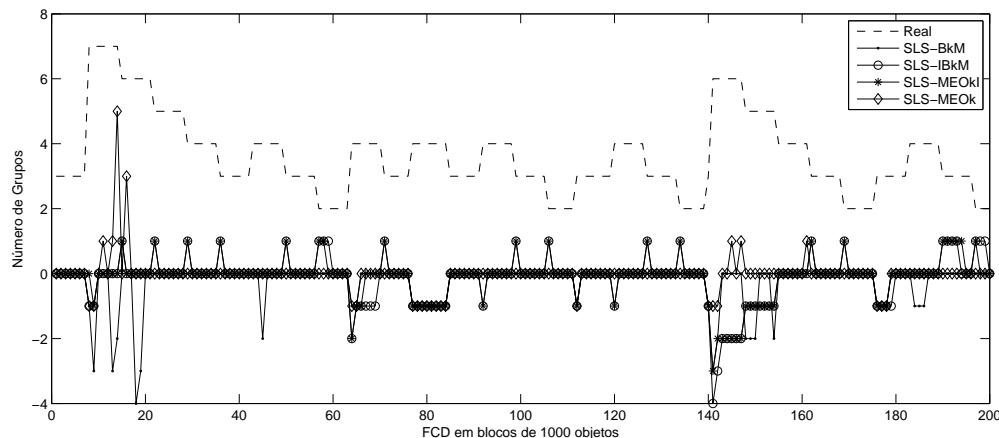
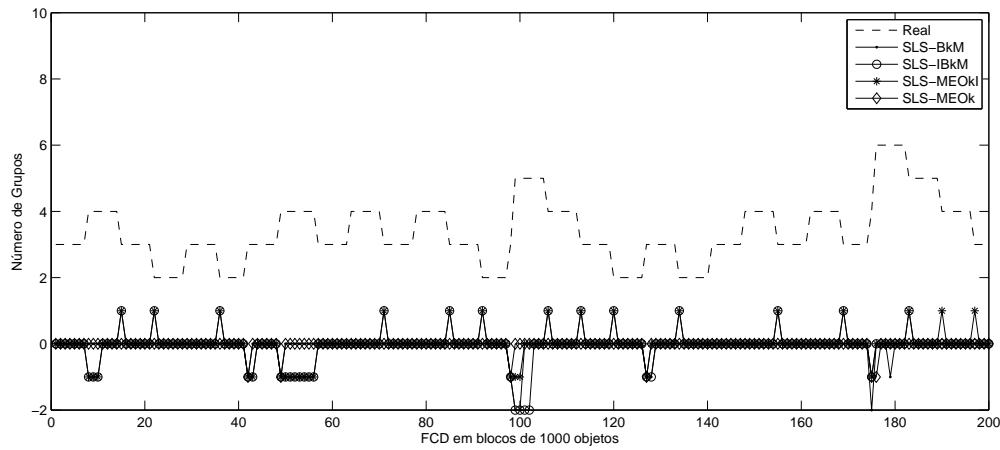
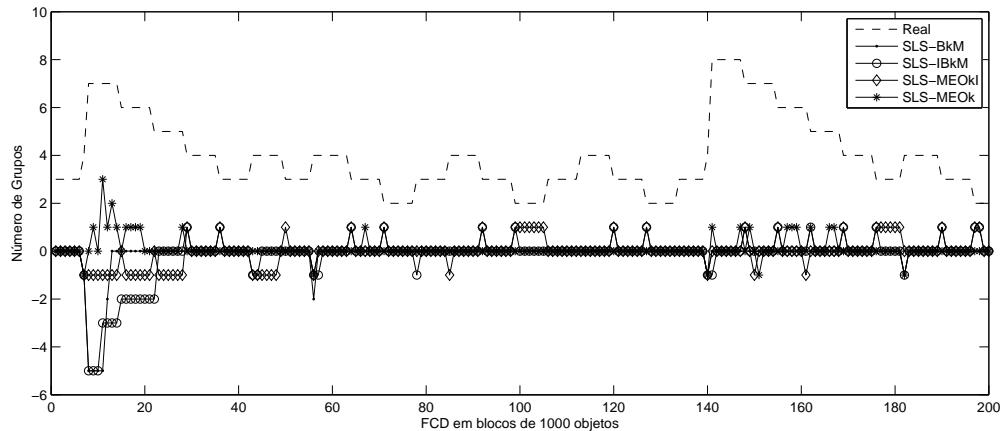


Figura 5.28: Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOk e SLS-MEOkl para a base de dados artificial bidimensional. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k .

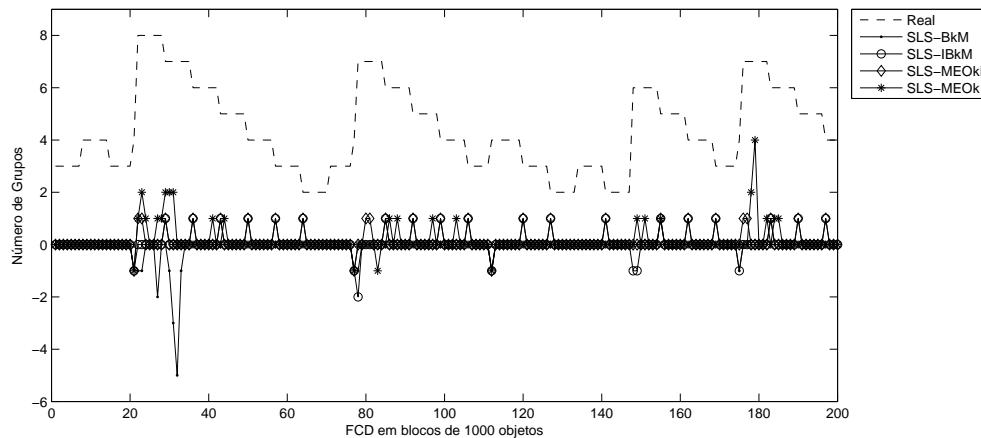


(a) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOkI e SLS-MEOk - Base de dados artificial com 4 dimensões.

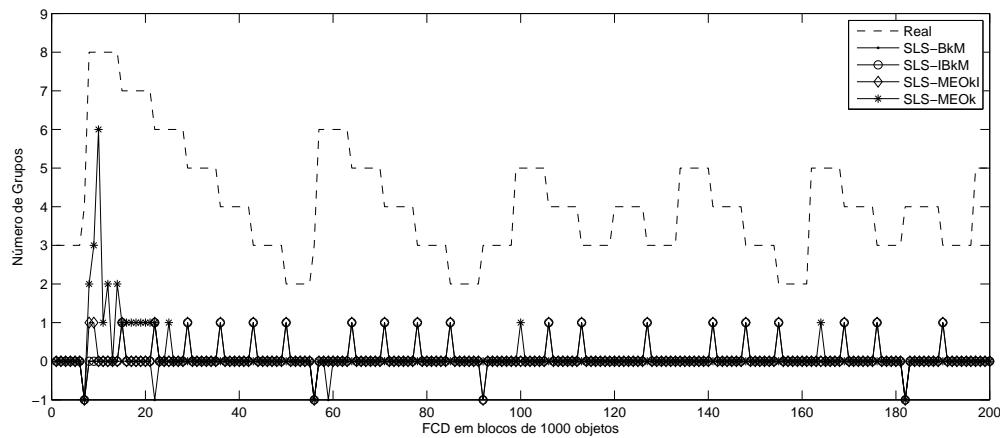


(b) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOkI e SLS-MEOk - Base de dados artificial com 8 dimensões.

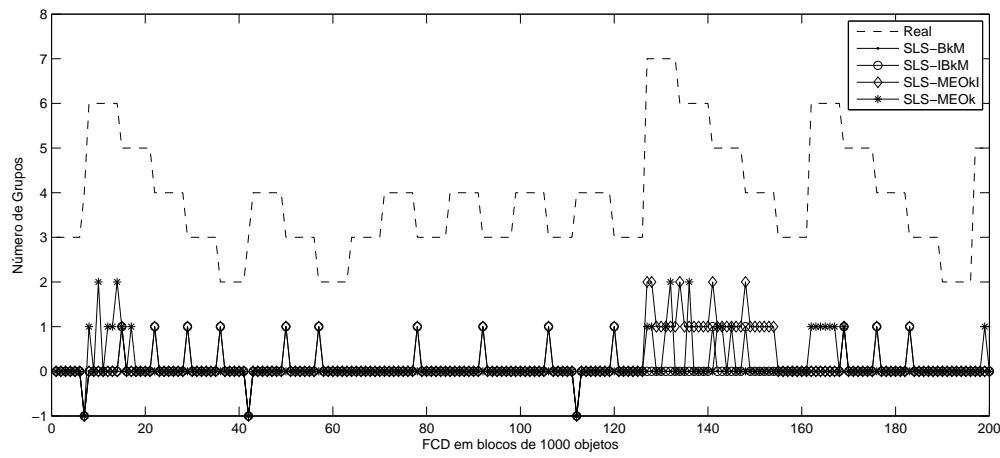
Figura 5.29: Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOkI e SLS-MEOk para as bases de dados artificiais com 4 e 8 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k .



(a) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOk e SLS-MEOkl - Base de dados artificial com 16 dimensões.



(b) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOk e SLS-MEOkl - Base de dados artificial com 32 dimensões.



(c) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOk e SLS-MEOkl - Base de dados artificial com 64 dimensões.

Figura 5.30: Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos SLS-BkM, SLS-IBkM, SLS-MEOk e SLS-MEOkl para as bases de dados artificiais com 16, 32 e 64 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k .

Tabela 5.5: Taxas de acerto do número correto de grupos para os algoritmos baseados no Stream LSearch - Base de Dados Artificiais.

Algoritmo	Base - 2D	Base - 4D	Base - 8D	Base - 16D	Base - 32D	Base - 64D
SLS-BkM	84%	95%	96%	94%	97%	99%
SLS-IBkM	70%	83%	80%	87%	88%	91%
SLS-MEOkI	74%	84%	71%	86%	87%	78%
SLS-MEOk _k	87%	97%	89%	87%	90%	87%

Nas Figuras 5.31, 5.32 e 5.33 são apresentados os resultados para o número de grupos encontrados pelos algoritmos FEACS-SSI e FEACS-PHT. Pode-se observar que os algoritmos, na maioria dos casos, reagem às mudanças de número de grupos, e frequentemente encontram o valor correto de k , sendo que o erro no número de grupos varia em apenas uma unidade. Especificamente, a taxa de acerto para o valor de k obtida pelo algoritmo FEACS-PHT foi de 68% (base de dados com duas dimensões) e para o algoritmo FEACS-PHT foi de 96% (bases de dados com 32 e 64 dimensões), conforme pode-se observar na Tabela 5.6.

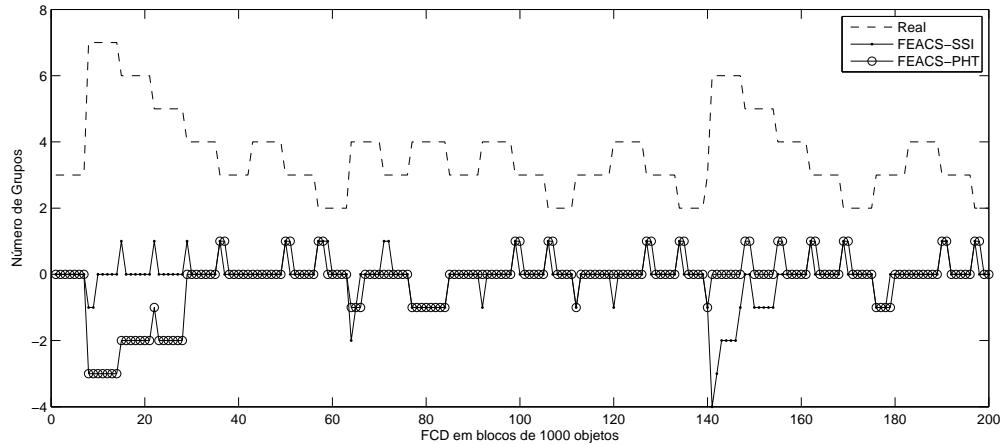
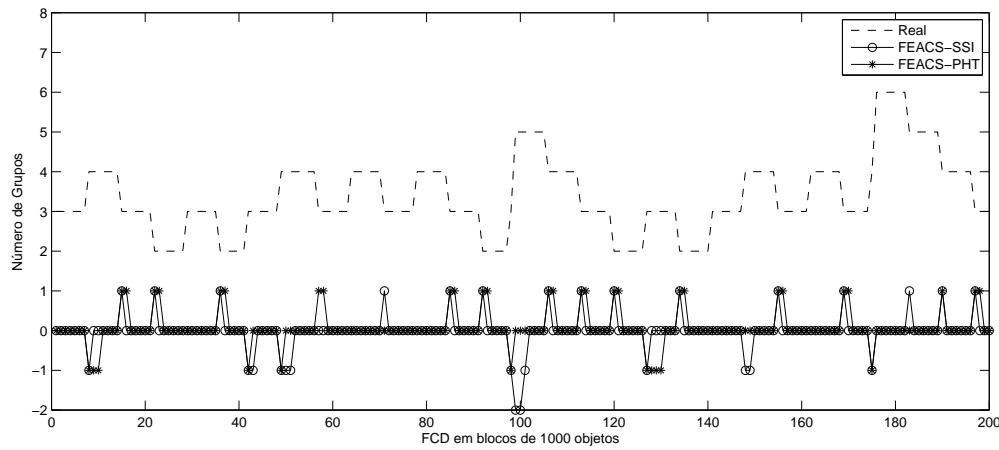
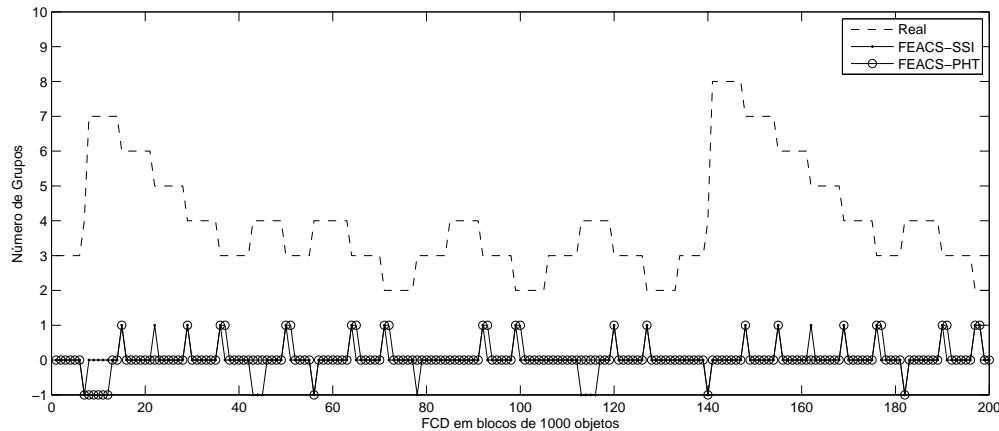


Figura 5.31: Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos FEACS-SSI e FEACS-PHT para a base de dados artificial bidimensional. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k .

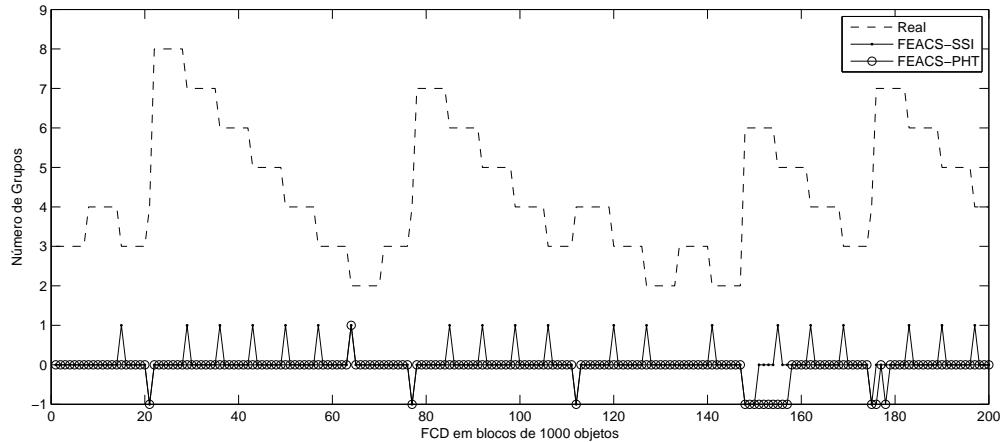


(a) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos FEACS-SSI e FEACS-PHT - Base de dados artificial com 4 dimensões.

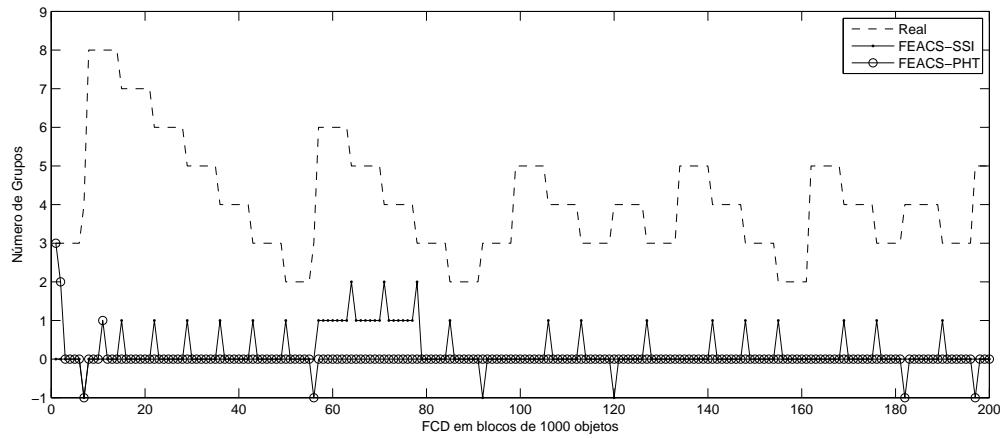


(b) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos FEACS-SSI e FEACS-PHT - Base de dados artificial com 8 dimensões.

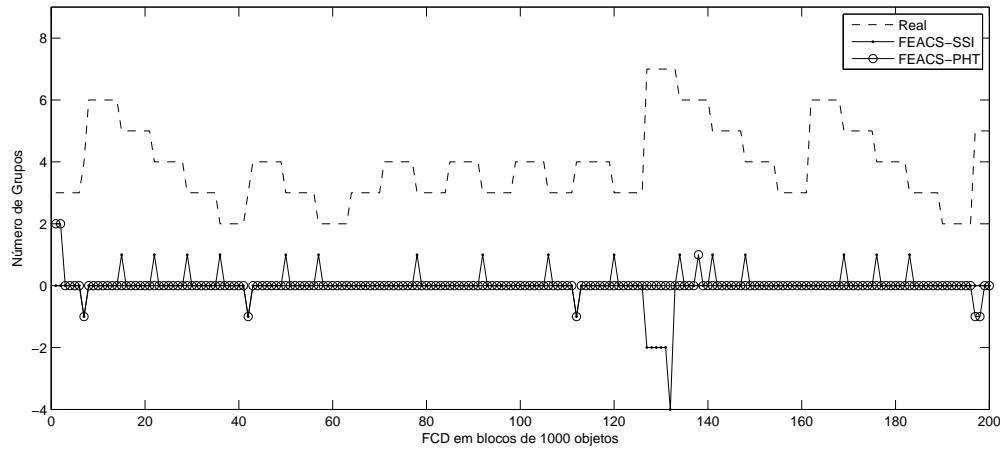
Figura 5.32: Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos FEACS-SSI e FEACS-PHT para as bases de dados artificiais com 4 e 8 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k .



(a) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos FEACS-SSI e FEACS-PHT - Base de dados artificial com 16 dimensões.



(b) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos FEACS-SSI e FEACS-PHT - Base de dados artificial com 32 dimensões.



(c) Evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos FEACS-SSI e FEACS-PHT - Base de dados artificial com 64 dimensões.

Figura 5.33: Exemplo ilustrativo da evolução do número conhecido de grupos (Real) e a sua diferença com relação ao número de grupos encontrado pelos algoritmos FEACS-SSI e FEACS-PHT para as bases de dados artificiais com 16, 32 e 64 dimensões. Valores negativos indicam que os algoritmos subestimaram o valor de k e para valores positivos os algoritmos superestimaram o valor real de k .

Tabela 5.6: Taxas de acerto do número correto de grupos para os algoritmos baseados no FEACS - Bases de Dados Artificiais.

Algoritmo	Base - 2D	Base - 4D	Base - 8D	Base - 16D	Base - 32D	Base - 64D
FEACS-PHT	68%	81%	83%	83%	96%	96%
FEACS-SSI	75%	85%	85%	85%	80%	88%

Resultados Sumarizados

A tabela 5.7 apresenta os resultados médios obtidos para o IRA e o tempo de processamento de 10 diferentes inicializações dos algoritmos considerando todas as bases de dados. Pode-se observar que todos os algoritmos estudados provêm bons resultados de IRA, sugerindo que partições de dados de alta qualidade foram obtidos. Considerando os tempos computacionais, o algoritmo SKM-B_kM se apresentou o mais rápido (102,42 milissegundos). O sucesso em termos de tempo de processamento do algoritmo SKM-B_kM está relacionado com a quantidade de dados a serem agrupados - 100 objetos no *coreset*. Embora o algoritmo *CluStream* também utilize 100 objetos para agrupar (microgrupos), a sua componente de abstração de dados demanda um custo computacional maior do que os algoritmos baseados no StreamKM++. Outra característica é que sua componente de agrupamento é simplesmente o algoritmo *k*-Médias enquanto que o *Stream LSearch* utiliza o algoritmo de agrupamento *LSearch* que realiza um procedimento de busca binária para encontrar uma partição com *k* grupos. Considerando a componente de estimativa de \hat{k} , pode-se observar também que os algoritmos baseados no B_kM têm se mostrado rápidos em relação aos algoritmos baseados no MEO_k, até mesmo para as versões incrementais.

Tabela 5.7: Valores médios de IRA e tempo de processamento - Base de Dados Sintética.

Algoritmo	IRA $\mu(\pm\sigma)$	Tempo de Processamento (milissegundos) $\mu(\pm\sigma)$
CLS-B _k M	0,98 ($\pm 0,03$)	1.271,45 ($\pm 148,52$)
CLS-IB _k M	0,98 ($\pm 0,04$)	1.227,66 ($\pm 148,15$)
CLS-MEO _k	0,99 ($\pm 0,01$)	1.434,43 ($\pm 149,78$)
CLS-MEO _k I	0,98 ($\pm 0,03$)	1.430,28 ($\pm 148,52$)
SKM-B _k M	0,96 ($\pm 0,04$)	102,42 ($\pm 12,98$)
SKM-IB _k M	0,98 ($\pm 0,05$)	153,83 ($\pm 19,70$)
SKM-MEO _k	0,99 ($\pm 0,04$)	239,18 ($\pm 17,45$)
SKM-MEO _k I	0,99 ($\pm 0,04$)	157,14 ($\pm 37,70$)
SLS-B _k M	0,99 ($\pm 0,03$)	1.996,52 ($\pm 673,71$)
SLS-IB _k M	0,98 ($\pm 0,04$)	1.210,12 ($\pm 555,40$)
SLS-MEO _k	0,99 ($\pm 0,01$)	17.885,86 ($\pm 1.869,81$)
SLS-MEO _k I	0,98 ($\pm 0,02$)	2.134,89 ($\pm 3.511,33$)
FEACS-PHT	0,98 ($\pm 0,03$)	120,98 ($\pm 58,43$)
FEACS-ISS	0,98 ($\pm 0,03$)	369,21 ($\pm 198,06$)

Para analisar a sensibilidade dos algoritmos para detectar e reagir a mudanças no FCD artificial, na Figura 5.34 são apresentados os resultados para o números de grupos encontrados pelos algoritmos no decorrer do fluxo dos dados artificiais. Esses resultados são provenientes da média de 10 execuções dos algoritmos, considerando todas as bases de dados artificiais, e foram incluídos para ilustrar o comportamento dos algoritmos. Pode-se observar que os algoritmos, na maioria dos casos, reagem às mudanças de número de grupos, e frequentemente encontram o valor correto de *k*. Especificamente, a menor taxa de acerto para o valor de *k* foi obtida pelo algoritmo FEACS-PHT (74%) e a maior taxa foi obtida pelo algoritmo SLS-MEO_k (90%). Porém, considerando a menor faixa de variação do número de grupos em relação ao valor conhecido de *k*, o menor valor foi obtido pelos algoritmos FEACS-PHT e CLS-MEO_kI, com variação de apenas uma unidade, ou seja,

com o valor do número de grupos uma unidade a mais ou a menos do valor real.

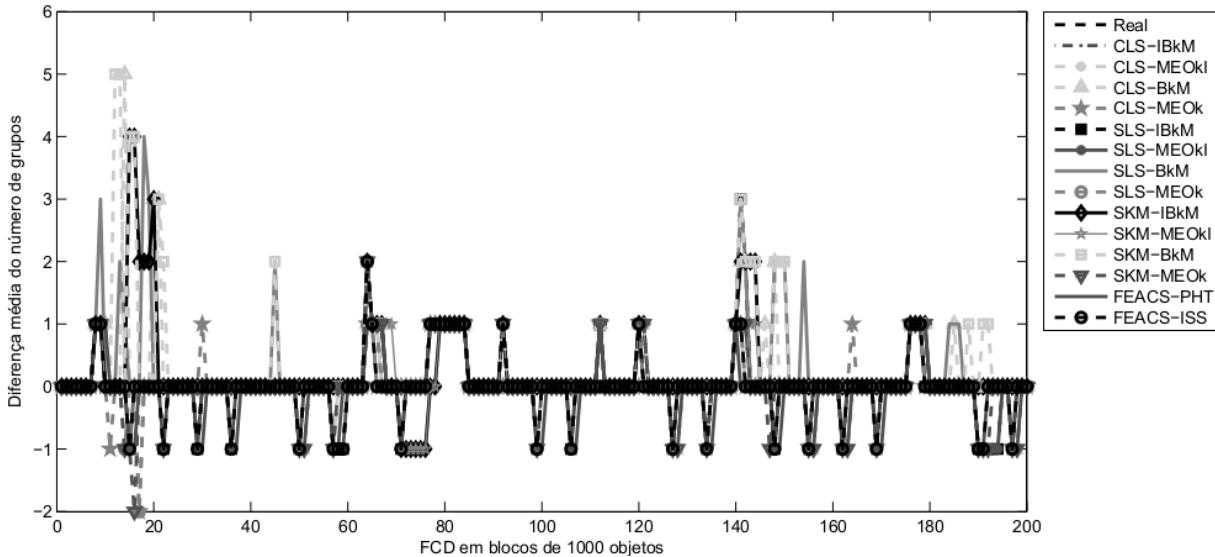
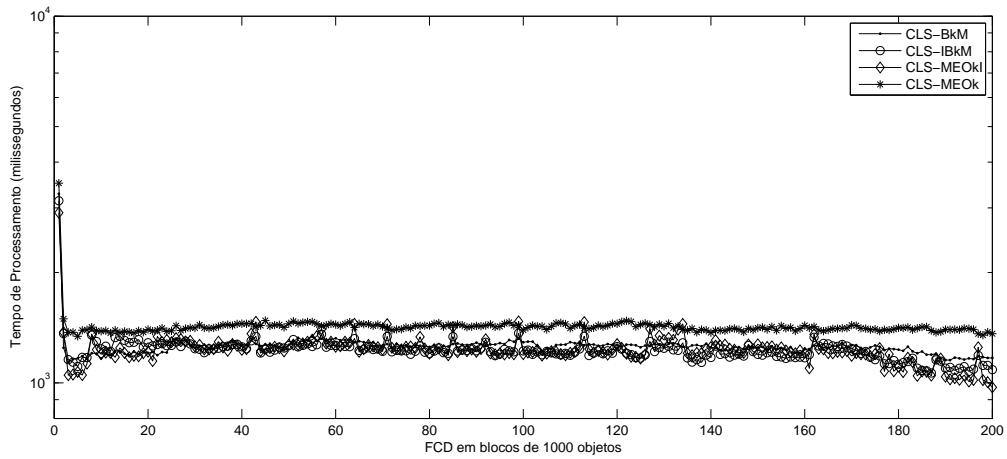


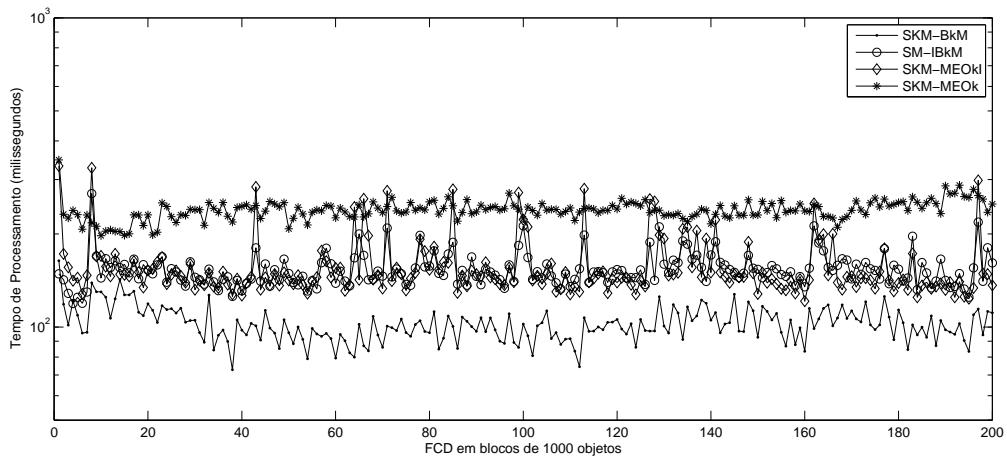
Figura 5.34: Exemplo ilustrativo da evolução do número de grupos e a sua diferença com relação ao número de grupos encontrado pelos algoritmos para a base de dados artificial.

Nas Figuras 5.35 e 5.36 são apresentados os tempos de processamento médios dos algoritmos nas bases artificiais. Os algoritmos baseados no CLS (Figura 5.35a) obtiveram custo de processamento competitivos entre eles. No início do FCD os algoritmos baseados no CLS têm um pico de tempo de processamento devido à necessidade de criar os microgrupos. Uma vez criados os microgrupos, os algoritmos baseados no CLS podem processar os demais blocos de objetos com um custo computacional menor que no início. Isto ocorre porque após essa fase de criação dos microgrupos, o algoritmo realiza apenas operações de atualização dos microgrupos (inserção do objeto no microgrupo mais próximo, remoção de microgrupos antigos e união de dois microgrupos). Além disso, observe que todos os algoritmos baseados no SKM (Figura 5.35b) e os algoritmos FEACS-PHT e FEACS-ISS (Figura 5.36b) obtiveram os menores valores de tempo de processamento (variando entre 102 e 370 milissegundos). O maior tempo de processamento foi obtido com o algoritmo SLS-MEOk (Figura 5.36a), com o valor médio de 17.885 milissegundos. Entretanto, a sua versão incremental (SLS-MEOkI) obteve aproximadamente 2.134 milissegundos, resultando em um ganho de performance de aproximadamente 830%.

Na maioria dos cenários, os algoritmos baseados no BkM mostraram melhores resultados de tempo de processamento comparado aos algoritmos baseados no MEOk por duas principais razões. Primeiro, o procedimento de busca guiada do algoritmo BkM frequentemente encontra partições de dados com o número de grupos menor que \sqrt{N} . Particularmente, uma vez que o BkM percebe que a qualidade da partição de dados decai para um número crescente de k , o algoritmo encerra a indução de grupos. Entretanto, o MEOk sistematicamente executa o algoritmo das k -Médias com o valor de k variando de 2 até \sqrt{N} , ocasionando assim um desperdício do recurso computacional quando o número de

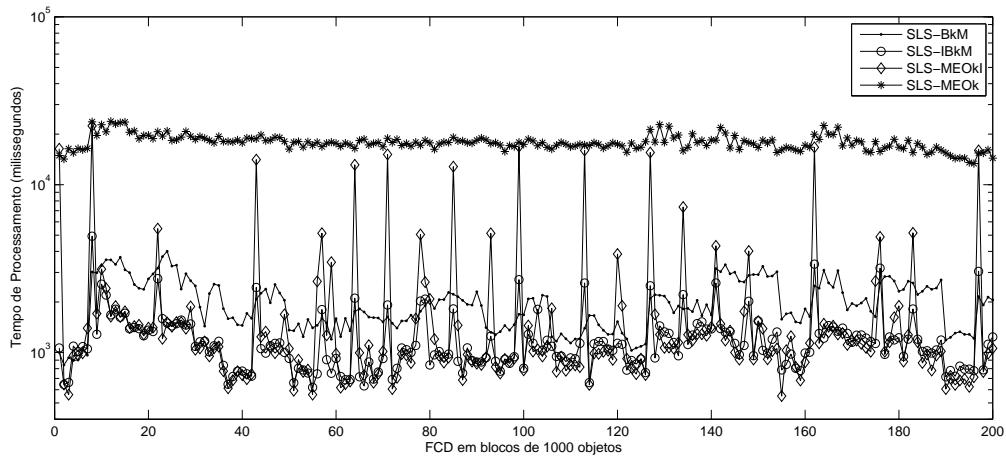


(a) Tempo de processamento dos algoritmos baseados no CluStream - base de dados artificial.

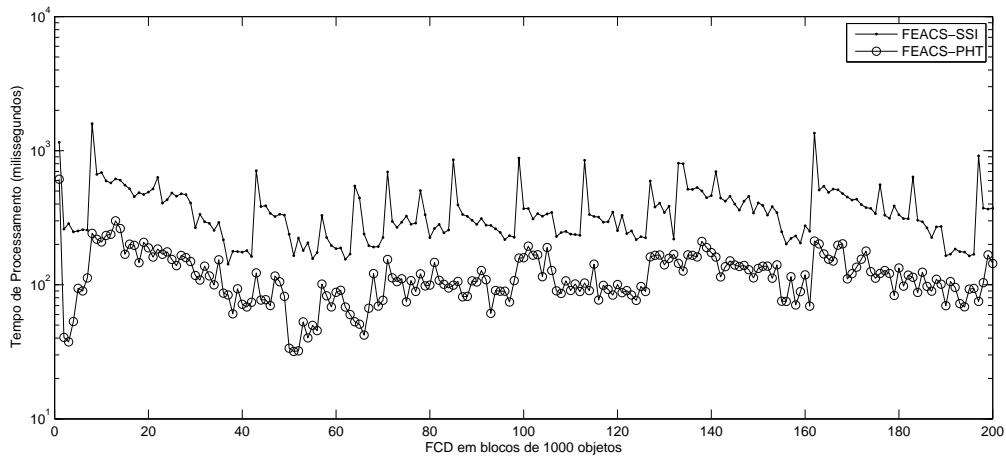


(b) Tempo de processamento dos algoritmos baseado no StreamKM++ - base de dados artificial.

Figura 5.35: Tempo de processamento em milissegundos dos algoritmos baseados no CluStream (Figura 5.35a) e no StreamKM++ (Figura 5.35b) - base de dados artificial.



(a) Tempo de processamento dos algoritmos baseados no *Stream LSearch* - base de dados artificial.



(b) Tempo de processamento dos algoritmos FEACS-SSI e FEACS-PHT - base de dados artificial.

Figura 5.36: Tempo de processamento em milissegundos dos algoritmos baseados no *Stream LSearch* (Figura 5.36a) e dos algoritmos FEACS-SSI e FEACS-PHT (Figura 5.36b) - base de dados artificial.

grupos encontrado é pequeno (*i.e.*, mais próximo de 2 do que de \sqrt{N}). A segunda razão é que para um valor crescente de k , o algoritmo BkM utiliza apenas aqueles objetos que pertencem ao grupo selecionado para executar o k -Médias (com $k = 2$), enquanto que o MEO k executa com toda a base de dados.

Para prover alguma garantia sobre a validade e a não-aleatoriedade dos resultados obtidos, resultados de testes estatísticos são apresentados seguindo o método proposto por Demšar (Demšar, 2006). Basicamente, este método é utilizado para comparações de múltiplos algoritmos em múltiplas bases de dados. Além disso, o método é baseado no uso do teste de Friedman (Friedman, 1940) com um correspondente teste *post-hoc*. Se a hipótese nula, a qual afirma que os algoritmos sob estudo têm desempenhos similares, é rejeitada, então procede-se com o teste *post-hoc* de Nemenyi (Hollander e Wolfe, 1999) para realizar comparações pareadas entre os algoritmos (com $\alpha = 5\%$). Considerando os valores de IRA, não foram encontradas diferenças estatísticas entre os resultados dos algoritmos para as bases sintéticas.

A Tabela 5.8 sumariza os resultados obtidos para as comparações pareadas dos tempos de processamento dos algoritmos. O símbolo \odot indica que não foram encontradas diferenças significativas entre a performance dos algoritmos na linha i e coluna j . Analogamente, o símbolo + indica que os resultados para o algoritmo na linha i são significativamente melhores do que aqueles apresentados pelos algoritmos na coluna j . Finalmente, o símbolo – informa que os resultados para o algoritmo na linha i são significativamente piores do que os resultados obtidos para o algoritmo na coluna j . Pode-se observar na Tabela 5.8 que os algoritmos SKM-BkM e FEACS-ISS apresentam os melhores resultados com relação ao tempo de processamento. O algoritmo SLS-MEO k se mostrou inferior aos demais algoritmos.

Tabela 5.8: Diferença estatisticamente significativas para os tempos de processamentos nas bases de dados artificiais. O símbolo \bullet indica que não foram encontradas diferenças significativas, + indica que os resultados para o algoritmo na linha i são superiores ao das colunas j e oposto é representado pelo símbolo –.

5.3.2 Bases de Dados Reais

KDDCup99

A Tabela 5.9 apresenta os resultados médios obtidos para a Silhueta Simplificada (ss) e o tempo de processamento para 10 diferentes inicializações dos algoritmos. Em razão do bloco de dados ser de tamanho 2.000 e haver 494.020 objetos na base de dados KDD-Cup99, 247 valores foram obtidos para ss e tempo de processamento para cada algoritmo. Na Tabela 5.9 são apresentados os valores de média e desvio padrão desses 247 valores para cada algoritmo. Pode-se observar que todos os algoritmos estudados apresentam valores próximos de 1,0, sugerindo que partições de boa qualidade foram obtidas. Considerando o tempo de processamento, o algoritmo FEACS-PHT se mostrou o mais rápido (tempo de aproximadamente 61 milissegundos). Note, também, que de maneira similar ao que já foi reportado para as bases de dados artificiais, os algoritmos baseados no BkM são mais rápidos que os algoritmos baseados no MEO_k . Além disso, as versões incrementais dos algoritmos apresentam uma grande redução no custo computacional comparado com as versões não incrementais dos algoritmos para a maioria dos casos avaliados.

Tabela 5.9: Valores de Silhueta Simplificada (ss) e tempo de processamento - Base de dados KDDCup99.

Algoritmo	SS - $\mu(\pm\sigma)$	Tempo de Processamento (Milissegundos) - $\mu(\pm\sigma)$
CLS-B_kM	0,80 ($\pm 0,25$)	1.332,19 ($\pm 1.272,58$)
CLS-IB_kM	0,84 ($\pm 0,23$)	1.367,62 ($\pm 1.331,03$)
CLS-MEO_k	0,80 ($\pm 0,24$)	1.717,53 ($\pm 1.343,97$)
CLS-MEO_{kI}	0,83 ($\pm 0,23$)	1.511,46 ($\pm 1.477,83$)
SKM-B_kM	0,80 ($\pm 0,23$)	123,58 ($\pm 40,81$)
SKM-IB_kM	0,90 ($\pm 0,15$)	306,77 ($\pm 92,12$)
SKM-MEO_k	0,82 ($\pm 0,22$)	312,60 ($\pm 87,90$)
SKM-MEO_{kI}	0,82 ($\pm 0,22$)	218,06 ($\pm 124,31$)
SLS-B_kM	0,84 ($\pm 0,18$)	2.960,64 ($\pm 2.021,22$)
SLS-IB_kM	0,84 ($\pm 0,23$)	370,89 ($\pm 1.600,94$)
SLS-MEO_k	0,86 ($\pm 0,15$)	37.702,53 ($\pm 6.504,23$)
SLS-MEO_{kI}	0,80 ($\pm 0,14$)	2.194,41 ($\pm 11.849,68$)
FEACS-PHT	0,82 ($\pm 0,22$)	61,24 ($\pm 254,34$)
FEACS-SSI	0,90 ($\pm 0,12$)	752,51 ($\pm 1.822,75$)

Na Figura 5.37 são apresentados os resultados dos tempos de processamento (escala logarítmica) dos algoritmos baseados no *CluStream* com relação às conexões de ataque. Observe que todos os algoritmos apresentam comportamento similares ao longo do FCD. Inicialmente, o *CluStream* tem um custo elevado de processamento para criar os microgrupos. A diferença notável é para o algoritmo *CLS-MEO_k*, que apresenta o maior tempo de processamento. Porém, essa diferença é observada apenas em alguns momentos do FCD nos quais o tempo de processamento está em torno 1.000 milissegundos. Alguns aspectos específicos dos tempos de processamento dos algoritmos são apresentados nas Figuras 5.38 (algoritmos *CLS-B_kM* e *CLS-IB_kM*) e 5.39 (algoritmos *CLS-MEO_{kI}* e *CLS-MEO_k*). A variação nas curvas de tempo de processamento está relacionada com a componente de abstração de dados, especificamente com o número de operações de união de microgru-

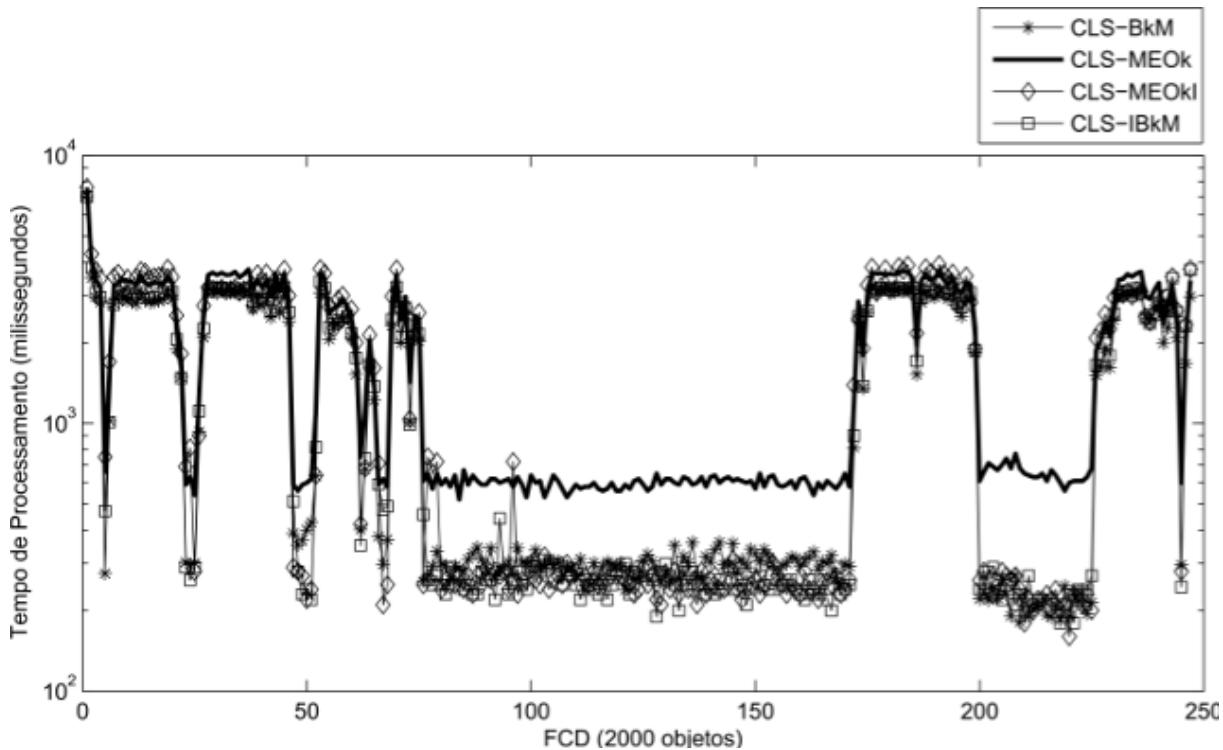


Figura 5.37: Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos baseados no *CluStream* (CLS-BkM, CLS-IBkM, CLS-MEOk e CLS-MEOkl) para a base de dados KDDCup99.

pos. Por exemplo, no intervalo de tempo de 78 à 169 ocorre um tipo de conexão de ataque denominada *Smurf*. Neste tipo de ataque, o *CluStream* apenas insere os objetos no microgrupo mais próximo que está relacionado com ataque *Smurf*, o que é computacionalmente menos custoso do que unir microgrupos vizinhos. Porém, quando há diferentes comportamentos de ataque, o tempo de processamento cresce devido às operações de união de microgrupos executadas pelo *CluStream*, que apresenta complexidade computacional quadrática com número de microgrupos.

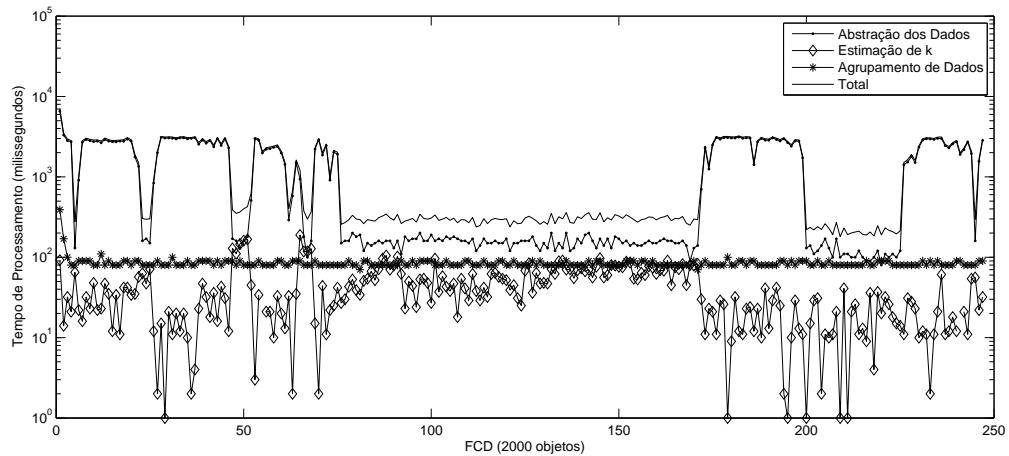
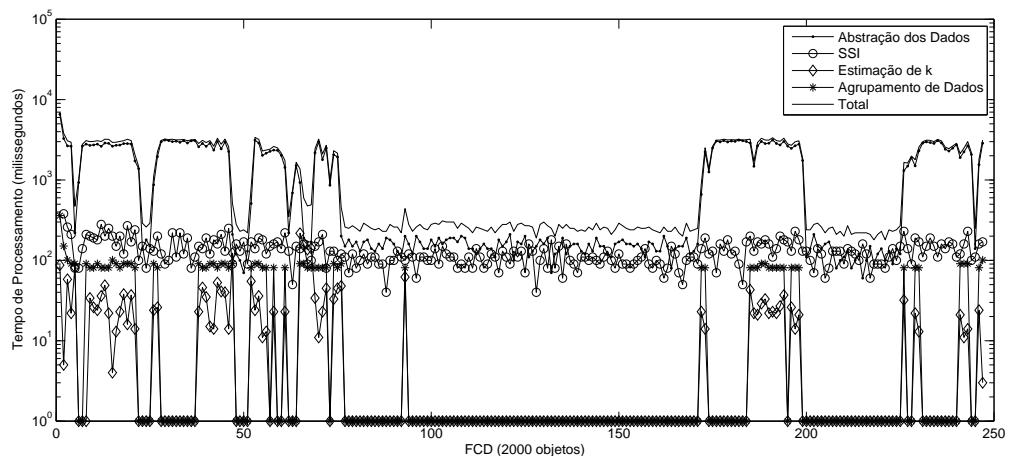
(a) Tempo de processamento do algoritmo CLS-B_kM - base de dados KDDCup99.(b) Tempo de processamento do algoritmo CLS-IB_kM - base de dados KDDCup99.

Figura 5.38: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos CLS-B_kM (Figura 5.38a) e CLS-IB_kM (Figura 5.38b).

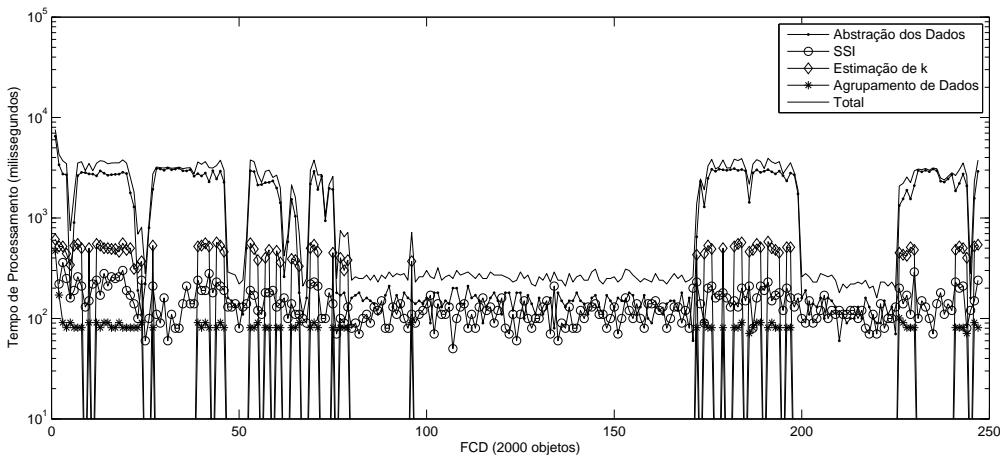
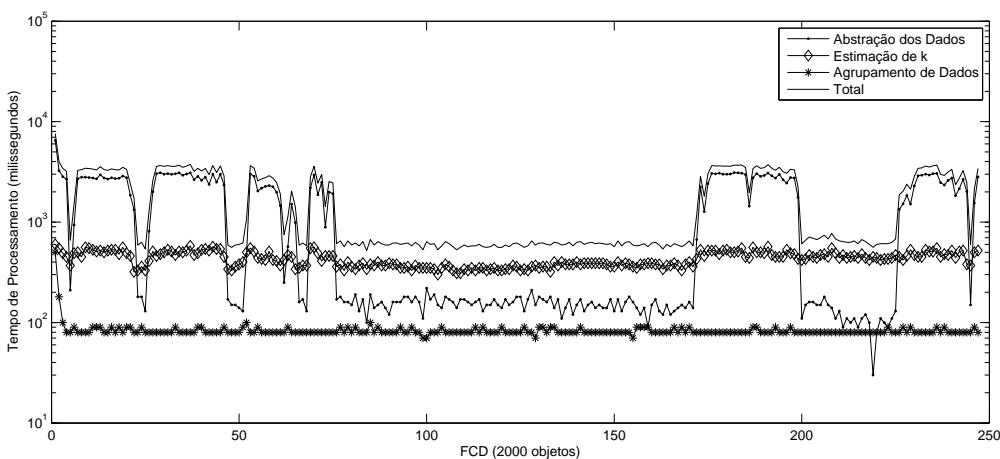
(a) Tempo de processamento do algoritmo CLS-MEO k I - base de dados KDDCup99.(b) Tempo de processamento do algoritmo CLS-MEO k - base de dados KDDCup99.

Figura 5.39: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos CLS-MEO k I (Figura 5.39a) e CLS-MEO k (Figura 5.39b).

Em relação aos algoritmos baseados no StreamKM++, na Figura 5.40 são apresentados os resultados de tempo de processamento médio. Pode-se observar que o algoritmo SKM-B k M obteve frequentemente os menores valores de tempo de processamento. Na Figura 5.41 são apresentados os tempos de processamento médios de cada componente para os algoritmos SKM-B k M e SKM-IB k M e na Figura 5.42 são apresentados os tempos de processamento médios de cada componente para os algoritmos SKM-MEO k e SKM-MEO k I. Pode-se observar que para os algoritmos SKM-IB k M e SKM-MEO k I, as componentes de estimativa de \hat{k} e agrupamento de dados foram utilizadas em momentos nos quais diferentes tipos de ataques ocorriam, similarmente ao ocorrido com o algoritmo CluStream.

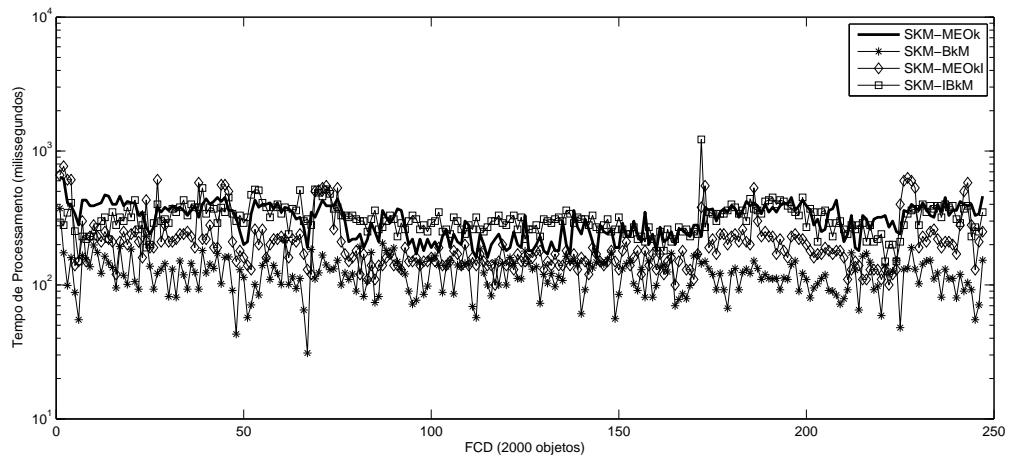
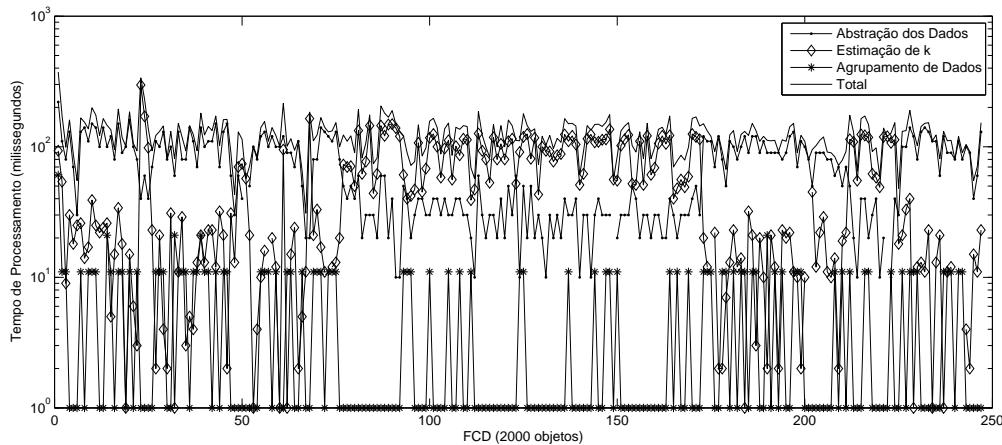
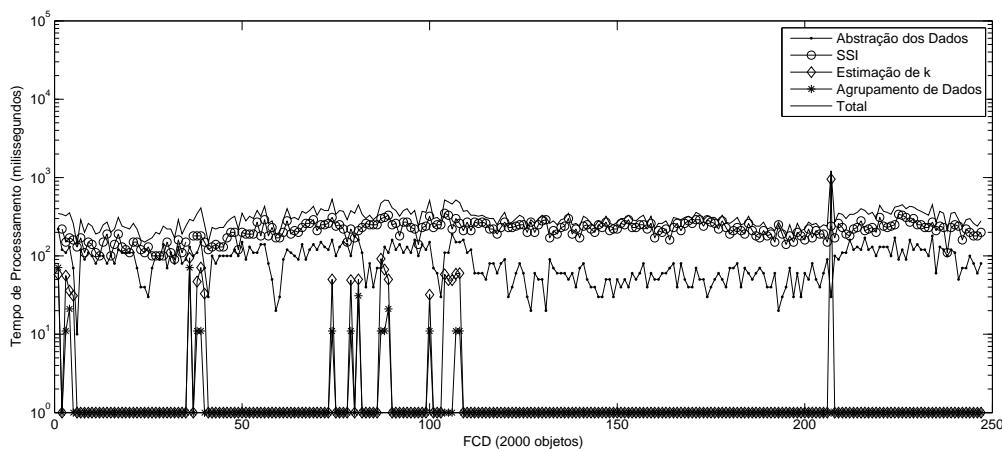


Figura 5.40: Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos baseados no StreamKM++ (SKM-B k M, SKM-IB k M, SKM-MEOk e SKM-MEOkl) para a base de dados KDDCup99.



(a) Tempo de processamento de cada componente do algoritmo SKM-B k M - Base de dados KDDCup99.



(b) Tempo de processamento de cada componente do algoritmo SKM-IB k M - Base de dados KDDCup99.

Figura 5.41: Tempo de processamento em milissegundos de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimação de \hat{k} e agrupamento de dados) e sua soma (total) para os algoritmos SKM-B k M (Figura 5.41a) e SKM-IB k M (Figura 5.41b).

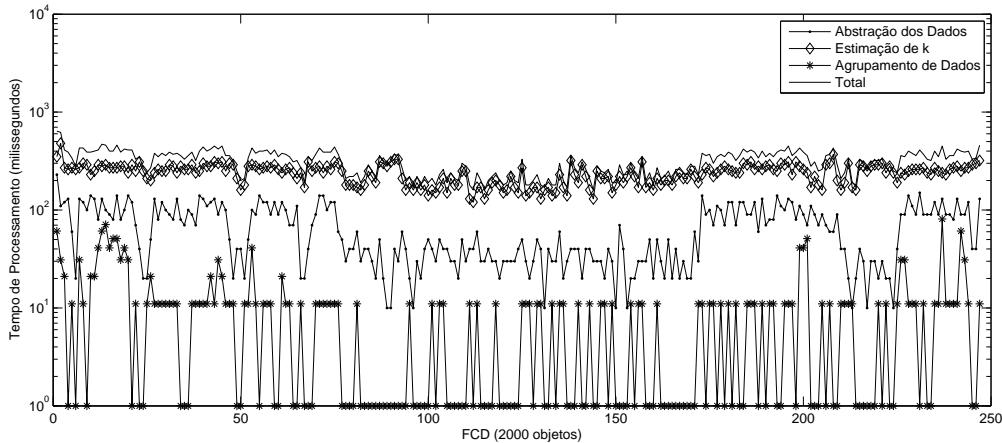
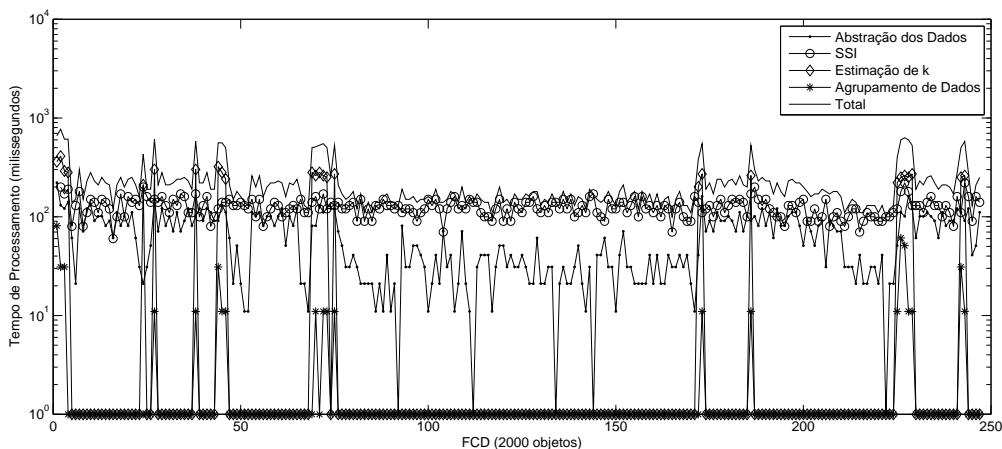
(a) Tempo de processamento de cada componente do algoritmo SKM-MEO k - Base de dados KDDCup99.(b) Tempo de processamento de cada componente do algoritmo SKM-MEO k I - Base de dados KDDCup99.

Figura 5.42: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (ssi), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos SKM-MEO k (Figura 5.42a) e SKM-MEO k I (Figura 5.42b).

Considerando os algoritmos baseados no *Stream LSearch*, na Figura 5.43 são apresentados os tempos de processamento médios dos algoritmos SLS-B k M, SLS-IB k M, SLS-MEO k e SLS-MEO k I. Pode-se observar que o algoritmo SLS-MEO k apresenta o maior resultado de tempo de processamento, enquanto que a sua versão incremental (SLS-MEO k I) e o algoritmo SLS-IB k M apresentam o menor tempo de processamento. Além disso, pode-se observar que as componentes de estimativa de \hat{k} e de agrupamento de dados são utilizadas em momentos do FCD nos quais surgem objetos de diferente tipos de ataques, semelhante ao ocorrido com o algoritmo *CluStream*. Nota-se também que os valores de tempo de processamento dos algoritmos SLS-B k M e SLS-MEO k variam bastante ao longo do FCD. A razão para essa variação está relacionada com o custo de processamento do algoritmo de agrupamento *LSearch*, conforme já observado nas bases de dados artificiais. Por exemplo, considerando o algoritmo SLS-MEO k , na Figura 5.45a são apresentados os

valores de tempo de processamento de cada componente do algoritmo: abstração de dados, estimativa de \hat{k} e agrupamento de dados. O comportamento da curva de tempo de processamento da componente de agrupamento de dados está relacionada com o número de grupos encontrados. Na Figura 5.46 é apresentado o valor médio de número de grupos obtidos ao longo do FCD para o algoritmo SLS-MEO k . Comparando os resultados dessa figura com o tempo de processamento da componente de agrupamento de dados do algoritmo SLS-MEO k pode-se observar que os valores de tempo de processamento estão associados aos valores de número de grupos.

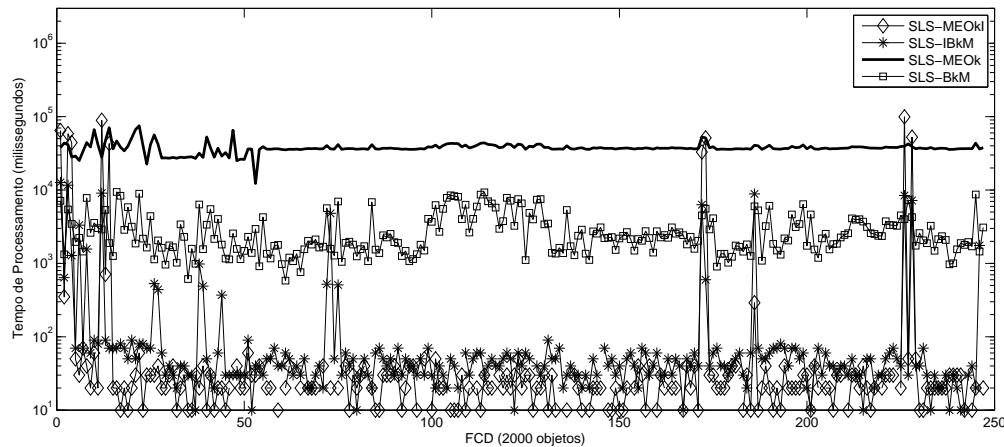
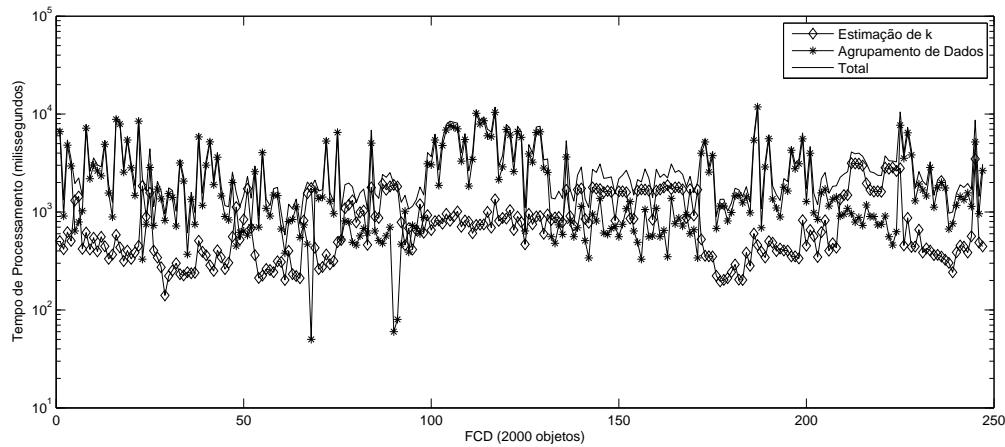
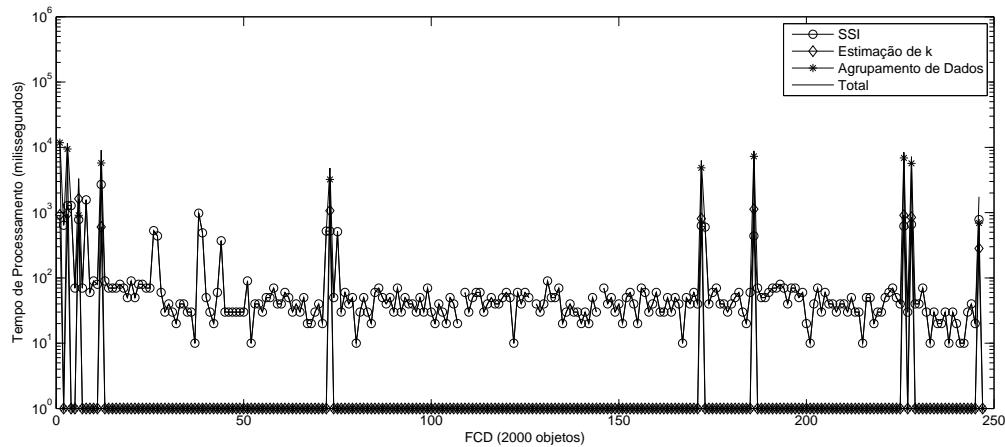


Figura 5.43: Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos baseados no *Stream LSearch* (SLS-BkM, SLS-IBkM, SLS-MEO k e SLS-MEO $k!$) - Base de dados KDDCup99.

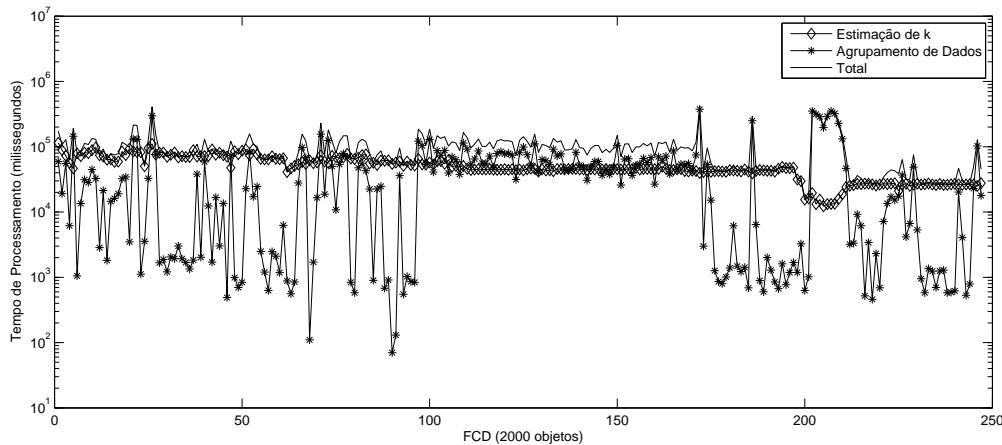


(a) Tempo de processamento (escala logarítmica) de cada componente do algoritmo SLS-B k M - Base de dados KDDCup99.

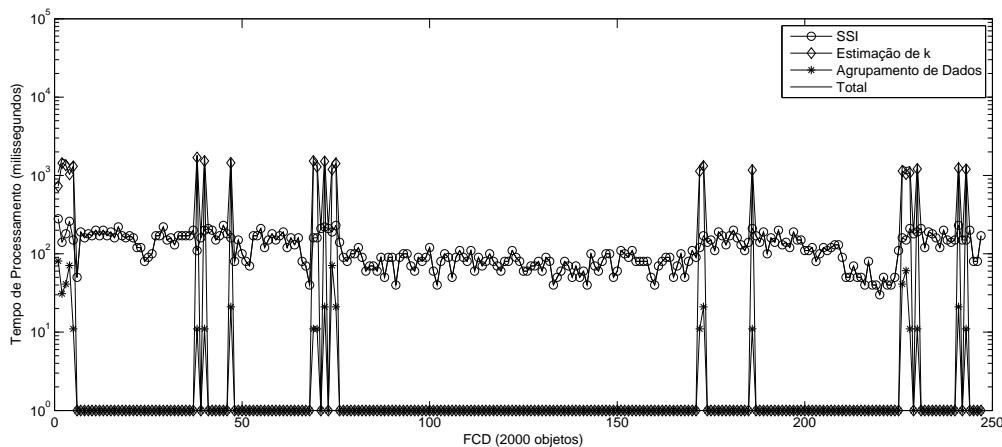


(b) Tempo de processamento (escala logarítmica) de cada componente do algoritmo SLS-IB k M - Base de dados KDDCup99.

Figura 5.44: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimação de \hat{k} e agrupamento de dados) e sua soma (total) para os algoritmos SLS-B k M (Figura 5.44a) e SLS-IB k M (Figura 5.44b).



(a) Tempo de processamento (escala logarítmica) de cada componente do algoritmo SLS-MEO k - Base de dados KDDCup99.



(b) Tempo de processamento (escala logarítmica) de cada componente do algoritmo SLS-MEO k I - Base de dados KDDCup99.

Figura 5.45: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimação de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos SLS-MEO k (Figura 5.45a) e SLS-MEO k I (Figura 5.45b).

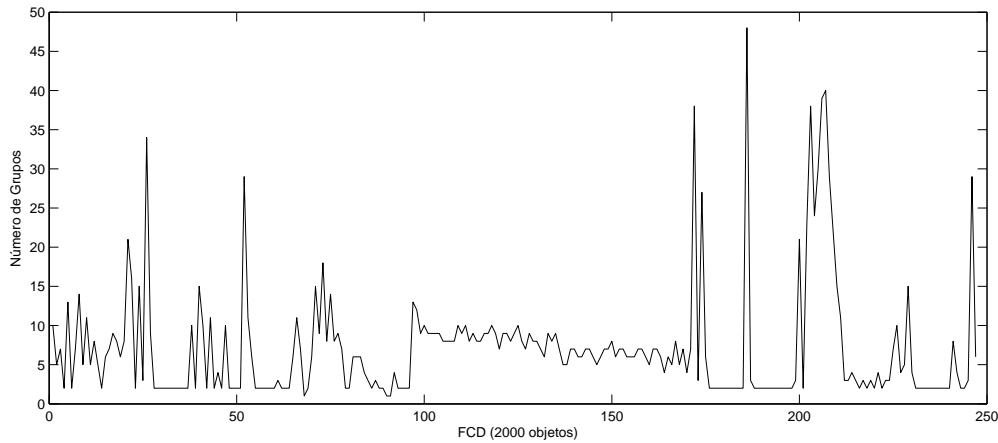


Figura 5.46: Número de grupos obtidos pelo algoritmo SLS-MEO_k - Base de dados KDD-Cup99.

Considerando os algoritmos FEACS-PHT e FEACS-ISS, no gráfico da Figura 5.47 são apresentados os tempos de processamento médio dos algoritmos. Pode-se notar que o algoritmo FEACS-PHT apresenta frequentemente o menor tempo de processamento. Os picos nas curvas de tempo de processamento dos algoritmos estão relacionados com os momentos em que a estimação de \hat{k} foi utilizada *i.e.*, quando diferentes tipos de ataque são observados no FCD. Na Figura 5.48 pode-se observar os momentos em que o algoritmo de estimação de \hat{k} são utilizados, justificando os picos na curva de tempo de processamento.

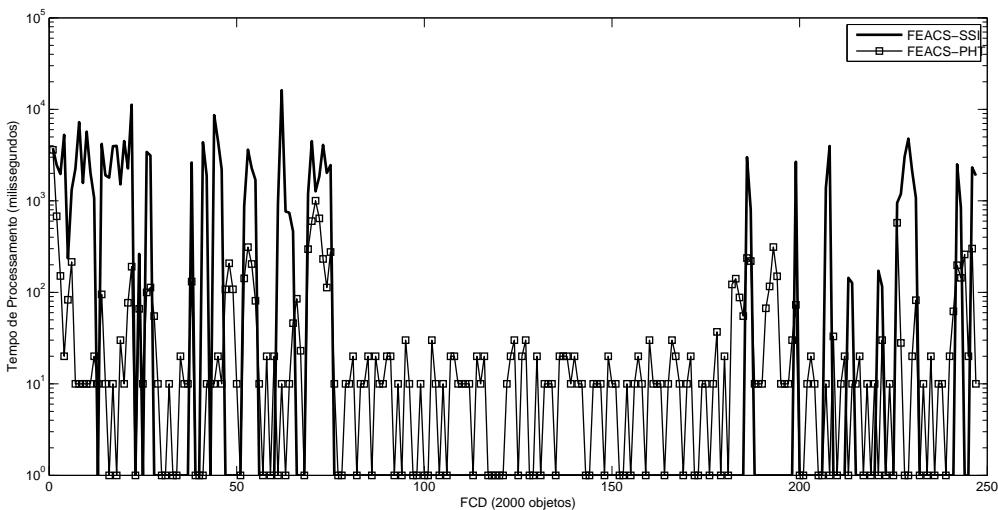
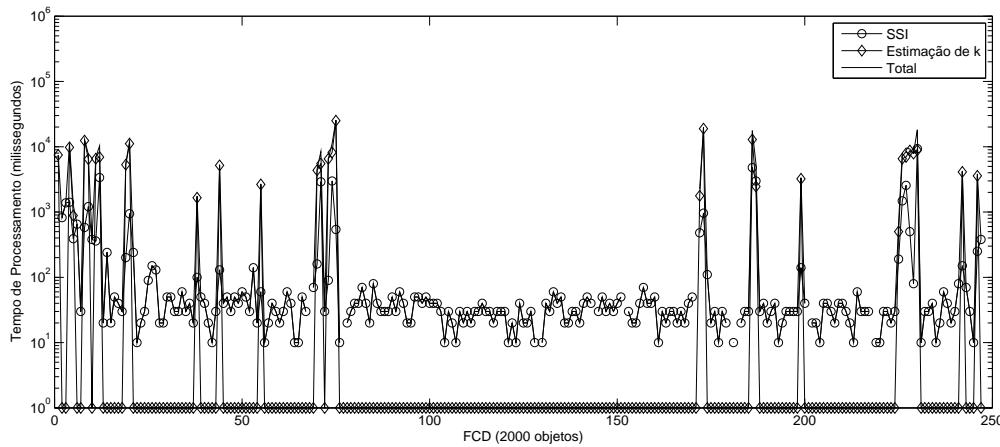
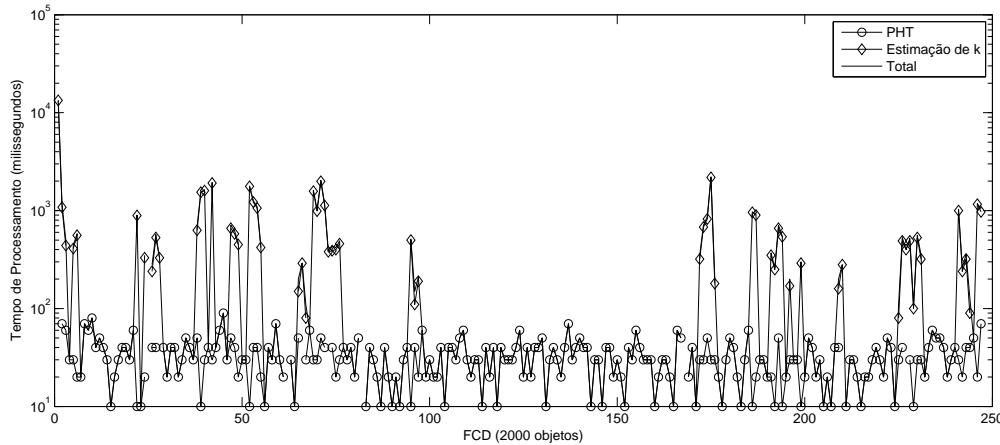


Figura 5.47: Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos FEACS-PHT e FEACS-SSI para a base de dados KDDCup99.



(a) Tempo de processamento (escala logarítmica) de cada componente do algoritmo FEACS-SSI - Base de dados KDDCup99.



(b) Tempo de processamento (escala logarítmica) de cada componente do algoritmo FEACS-PHT - Base de dados KDDCup99.

Figura 5.48: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (silhueta simplificada incremental (SSI), atualização incremental combinada com o PHT e estimativa de \hat{k}) e sua soma (total) dos algoritmos FEACS-SSI 5.48a e FEACS-PHT 5.48b.

A Tabela 5.10 apresenta os resultados do teste de significância estatística para comparações pareadas entre os algoritmos. Seguindo a mesma notação adotada para a Tabela 5.8, pode-se observar que os algoritmos SLS-IB_kM e SLS-MEO_k se mostraram superiores considerando o critério da ss. Considerando os tempos de processamento, a Tabela 5.11 apresenta os resultados do teste de significância estatística. Nesse resultados, os algoritmos FEACS-PHT e SKM-B_kM se mostraram superiores.

Tabela 5.10: Análise de diferenças estatisticamente significativas para os valores de SS para a base de dados KDDCup99. O símbolo \odot indica que não foram encontradas diferenças significativas, + indica que os resultados para o algoritmo na linha i são superiores ao das colunas j e oposto é representado pelo símbolo $-$.

	SKM-IB k M	SLS-MEO k	SKM-MEO k I	FEACS-ISS	CLS-MEO k I	CLS-IB k M	FEACS-PHT	CLS-MEO k	SKM-B k M	CLS-B k M	SLS-B k M	SKM-MEO k I	SLS-IB k M
SKM-IB k M		\odot	+	+	+	+	+	+	+	+	+	+	+
SLS-MEO k	\odot		\odot	\odot	\odot	\odot	\odot	\odot	\odot	\odot	\odot	\odot	\odot
SKM-MEO k I	-	\odot		\odot	\odot	\odot	\odot	\odot	\odot	\odot	\odot	\odot	\odot
FEACS-ISS	-	\odot	\odot		\odot	\odot	\odot	\odot	\odot	\odot	\odot	\odot	\odot
CLS-MEO k I	-	-	\odot	\odot		\odot	\odot	\odot	\odot	\odot	\odot	\odot	\odot
CLS-IB k M	-	-	\odot	\odot	\odot		\odot	\odot	\odot	\odot	\odot	\odot	\odot
FEACS-PHT	-	-	\odot	\odot	\odot	\odot		\odot	\odot	\odot	\odot	\odot	\odot
CLS-MEO k	-	-	-	\odot	\odot	\odot	\odot		\odot	\odot	\odot	\odot	\odot
SKM-B k M	-	-	-	\odot	\odot	\odot	\odot	\odot		\odot	\odot	\odot	\odot
CLS-B k M	-	-	-	-	-		\odot	\odot	\odot		\odot	\odot	\odot
SLS-B k M	-	-	-	-	-	\odot	\odot	\odot	\odot	\odot		\odot	\odot
SLS-MEO k I	-	-	-	-	-	-	-	-	\odot	\odot	\odot		\odot
SKM-MEO k	-	-	-	-	-	-	-	-	-	\odot	\odot		\odot
SLS-IB k M	-	-	-	-	-	-	-	-	-	\odot	\odot	\odot	

Tabela 5.11: Análise de diferenças estatisticamente significativas para os valores de tempo de processamento para a base de dados KDDCup99. O símbolo \odot indica que não foram encontradas diferenças significativas, + indica que os resultados para o algoritmo na linha i são superiores ao das colunas j e oposto é representado pelo símbolo –.

	FEACS-PHT	SKM-B k M	FEACS-ISS	SLS-IB k M	SLS-MEO k I	SKM-MEO k I	SKM-IB k M	CLS-IB k M	CLS-B k M	CLS-MEO k I	SLS-B k M	CLS-MEO k	SLS-MEO k
FEACS-PHT	–	–	+	+	+	+	+	+	+	+	+	+	+
SKM-B k M	–	–	–	–	–	–	–	–	–	–	–	–	–
FEACS-ISS	–	–	–	–	–	–	–	–	–	–	–	–	–
SLS-IB k M	–	–	–	–	–	–	–	–	–	–	–	–	–
SLS-MEO k I	–	–	–	–	–	–	–	–	–	–	–	–	–
SKM-MEO k I	–	–	–	–	–	–	–	–	–	–	–	–	–
SKM-MEO k	–	–	–	–	–	–	–	–	–	–	–	–	–
SKM-IB k M	–	–	–	–	–	–	–	–	–	–	–	–	–
CLS-IB k M	–	–	–	–	–	–	–	–	–	–	–	–	–
CLS-B k M	–	–	–	–	–	–	–	–	–	–	–	–	–
CLS-MEO k I	–	–	–	–	–	–	–	–	–	–	–	–	–
SLS-B k M	–	–	–	–	–	–	–	–	–	–	–	–	–
CLS-MEO k	–	–	–	–	–	–	–	–	–	–	–	–	–
SLS-MEO k	–	–	–	–	–	–	–	–	–	–	–	–	–

Forest Cover Type

A Tabela 5.12 apresenta os resultados médios obtidos para a Silhueta Simplificada (SS) e o tempo de processamento para 10 diferentes inicializações dos algoritmos. Em razão do bloco de dados ser de tamanho 2,000 e haver 581.012 objetos na base de dados *Forest Cover Type*, 290 valores foram obtidos para SS e tempo de processamento para cada algoritmo. Na Tabela 5.12 são apresentados os valores de média e desvio padrão desses 290 valores para cada algoritmo. Pode-se observar que todos os algoritmos estudados apresentam valores próximos de 1,0, sugerindo que partições de boa qualidade foram obtidas. Considerando o tempo de processamento, o algoritmo FEACS-PHT se mostrou o mais rápido (tempo de aproximadamente 80 milissegundos). Note, também, que de maneira similar ao que já foi reportado para a base de dados KDDCup99, os algoritmos baseados no B k M são mais rápidos que os algoritmos baseados no MEO k . Além disso, as versões incrementais dos algoritmos apresentam uma grande redução no custo computacional comparado com as versões não incrementais dos algoritmos — para a maioria dos casos avaliados.

Tabela 5.12: Valores de Silhueta Simplificada (ss) e tempo de processamento - Base de dados *Forest Cover Type*.

Algoritmo	SS - $\mu(\pm\sigma)$	Tempo de Processamento (Milissegundos) - $\mu(\pm\sigma)$
CLS-B_kM	0,83 (\pm 0,05)	2.269,55 (\pm 549,40)
CLS-IB_kM	0,82 (\pm 0,06)	1.886,48 (\pm 417,17)
CLS-MEO_k	0,84 (\pm 0,06)	2.930,45 (\pm 527,62)
CLS-MEO_{kI}	0,83 (\pm 0,06)	4.217,93 (\pm 544,02)
SKM-B_kM	0,81 (\pm 0,06)	126,48 (\pm 37,24)
SKM-IB_kM	0,80 (\pm 0,06)	246,24 (\pm 50,56)
SKM-MEO_k	0,83 (\pm 0,06)	449,30 (\pm 65,93)
SKM-MEO_{kI}	0,82 (\pm 0,07)	366,17 (\pm 216,54)
SLS-B_kM	0,84 (\pm 0,05)	4.238,14 (\pm 2.581,72)
SLS-IB_kM	0,82 (\pm 0,07)	645,28 (\pm 1.319,80)
SLS-MEO_k	0,86 (\pm 0,15)	39.267,78 (\pm 18.368,40)
SLS-MEO_{kI}	0,80 (\pm 0,14)	11.898,81 (\pm 11.849,68)
FEACS-PHT	0,78 (\pm 0,07)	80,63 (\pm 215,18)
FEACS-SSI	0,85 (\pm 0,04)	1.893,94 (\pm 5.027,93)

Na Figura 5.49 são apresentados os resultados dos tempos de processamento dos algoritmos baseados no *CluStream* para a base de dados *Forest Cover Type*. Observe que todos os algoritmos apresentam comportamento similares ao longo do FCD.

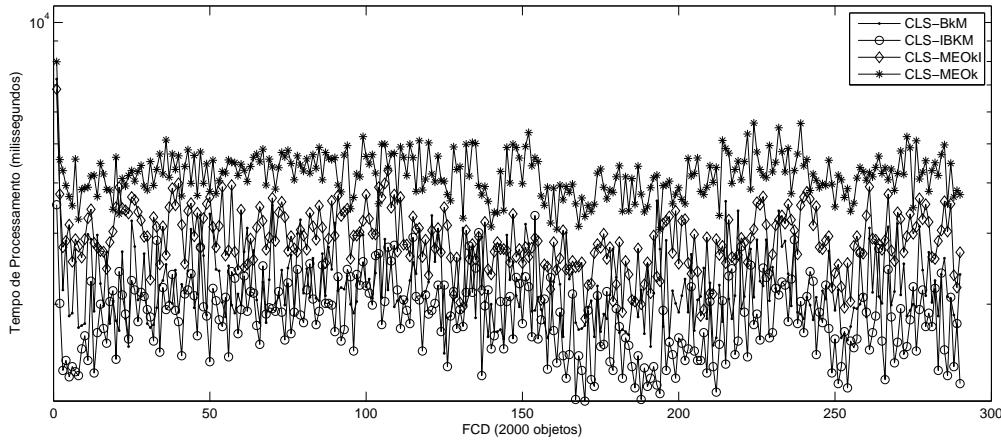
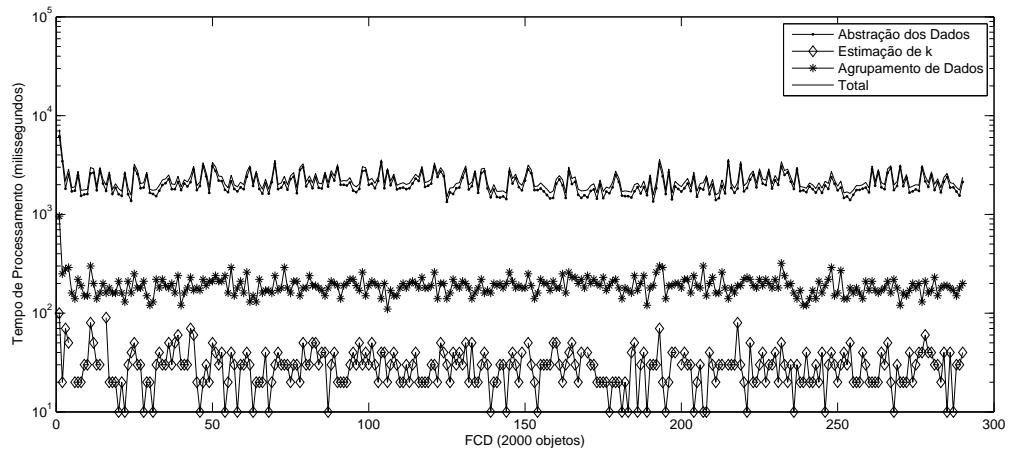
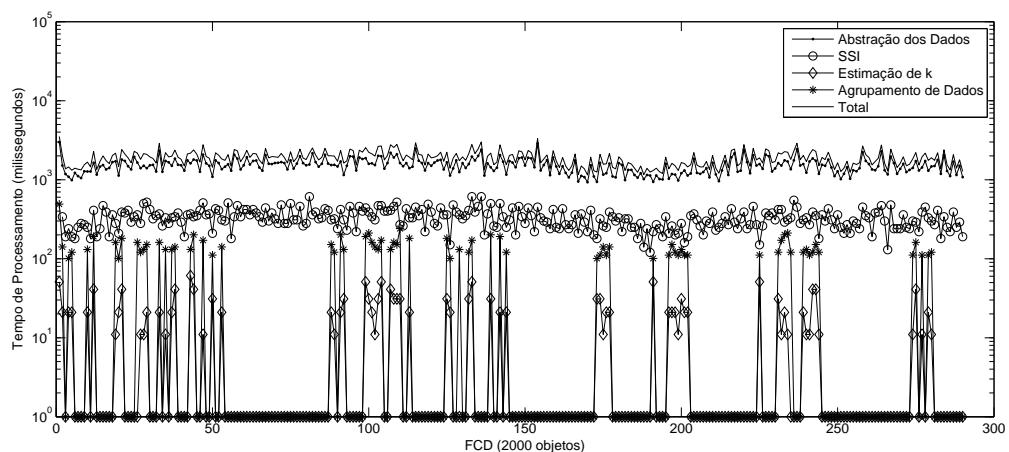


Figura 5.49: Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos baseados no *CluStream* (CLS-BkM, CLS-IBkM, CLS-MEOk e CLS-MEOkl) para a base de dados *Forest CoverType*.

Inicialmente, o *CluStream* tem um custo elevado de processamento para criar os microgrupos. Em seguida, são executadas as etapas de abstração dos dados para atualização dos microgrupos e estimativa de \hat{k} a partir dos microgrupos. Similarmente ao ocorrido nos experimentos com as demais bases de dados estudadas nessa tese, a razão desses comportamento está relacionada com a componente de abstração de dados, conforme pode-se observar na Figura 5.50 para visualizar os tempos de processamento médios de cada componente dos algoritmos CLS-BkM e CLS-IBkM e na Figura 5.51, que permite visualizar os tempos de processamentos médios de cada componente para os algoritmos CLS-MEOk e CLS-MEOkl. Pode-se observar que a componente de abstração de dados dos algoritmos apresenta um custo computacional superior (aproximadamente 2.500 milissegundos) comparado às demais componentes, sendo significativa no tempo total (soma dos tempos de processamento de cada componente).

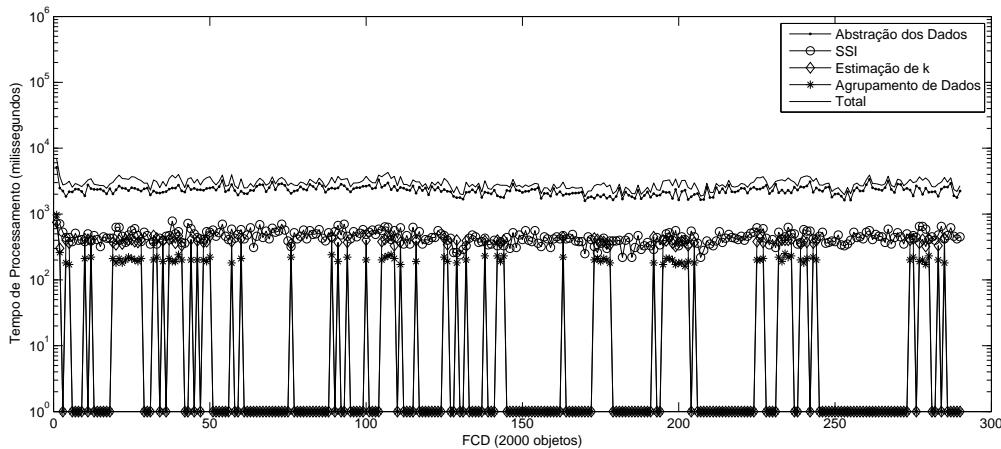


(a) Tempo de processamento do algoritmo CLS-B k M - base de dados *Forest CoverType*.

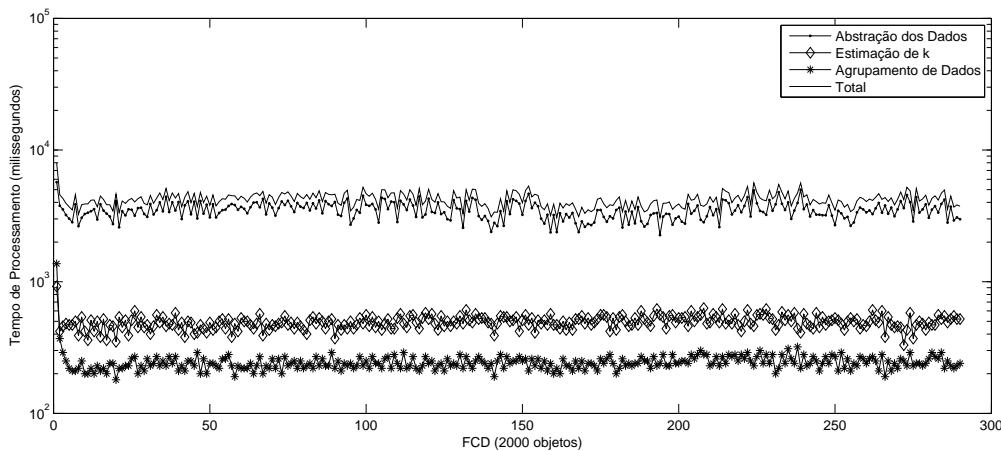


(b) Tempo de processamento do algoritmo CLS-IB k M - base de dados *Forest CoverType*.

Figura 5.50: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de k e agrupamento de dados) e sua soma (total) dos algoritmos CLS-B k M (Figura 5.50a) e CLS-IB k M (Figura 5.50b) - Base de dados *Forest CoverType*.



(a) Tempo de processamento do algoritmo CLS-MEO k_l - base de dados *Forest CoverType*.



(b) Tempo de processamento do algoritmo CLS-MEO k - base de dados *Forest CoverType*.

Figura 5.51: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (ssi), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos CLS-MEO k_l (Figura 5.51a) e CLS-MEO k (Figura 5.51b) - Base de dados *Forest CoverType*.

Em relação aos algoritmos baseados no StreamKM++, na Figura 5.52 são apresentado os resultados de tempo de processamento médio. Pode-se observar que o algoritmo SKM-B k M obteve frequentemente os menores valores de tempo de processamento (aproximadamente 126 milissegundos) em contrapartida com o algoritmo SKM-MEO k (aproximadamente 450 milissegundos). Na Figura 5.53 são apresentados os tempos de processamento médios de cada componente para os algoritmos SKM-B k M e SKM-IB k M e na Figura 5.54 são apresentados os tempos de processamentos médios de cada componente para os algoritmos SKM-MEO k e SKM-MEO k_l . Pode-se observar que para os algoritmos SKM-IB k M e SKM-MEO k_l , as componentes de estimativa de \hat{k} e agrupamento de dados foram utilizadas em poucos momentos do FCD, indicando que uma mudança na qualidade da partição foi observada e portanto sendo necessário aplicar o método de estimativa de \hat{k} para encontrar uma partição que melhore a qualidade do agrupamento.

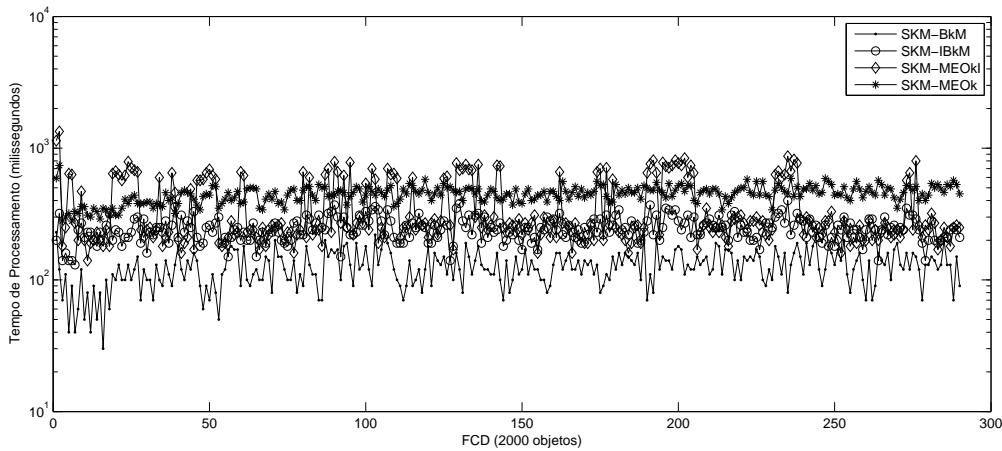
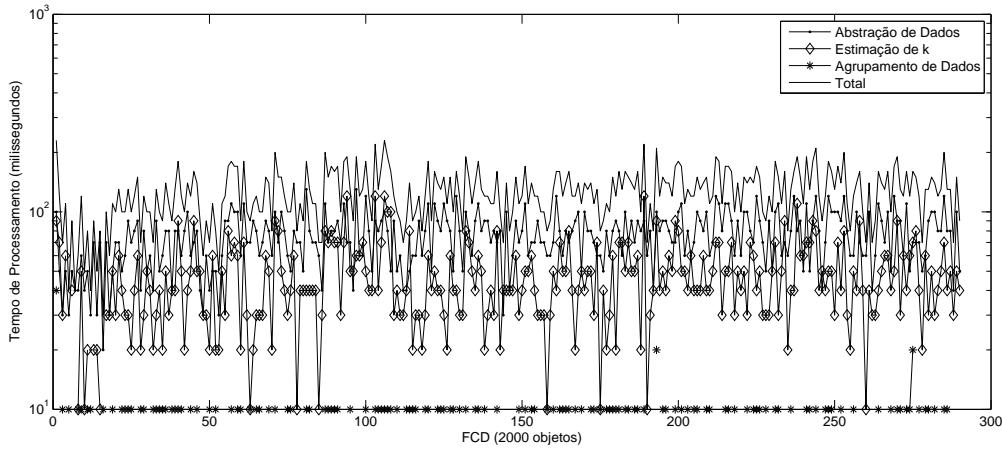
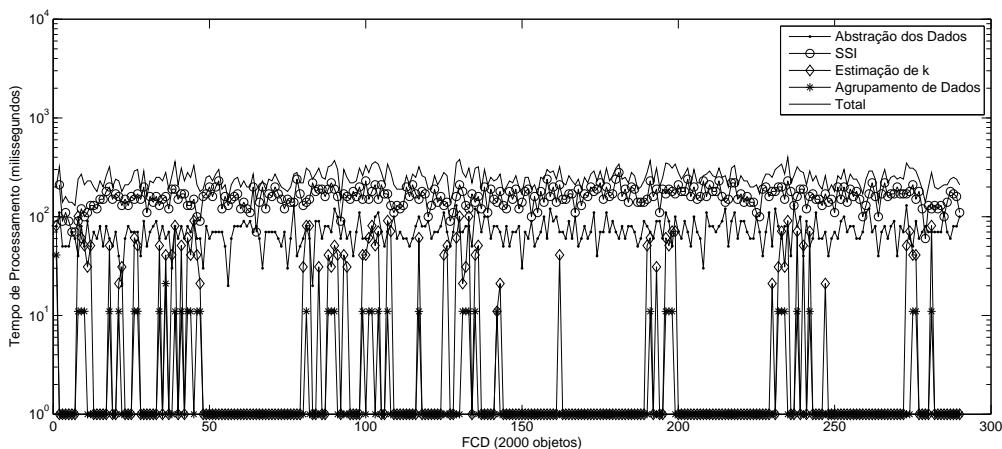


Figura 5.52: Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos baseados no StreamKM++ (SKM-BkM, SKM-IBkM, SKM-MEOkl e SKM-MEOkl) para a base de dados *Forest CoverType*.

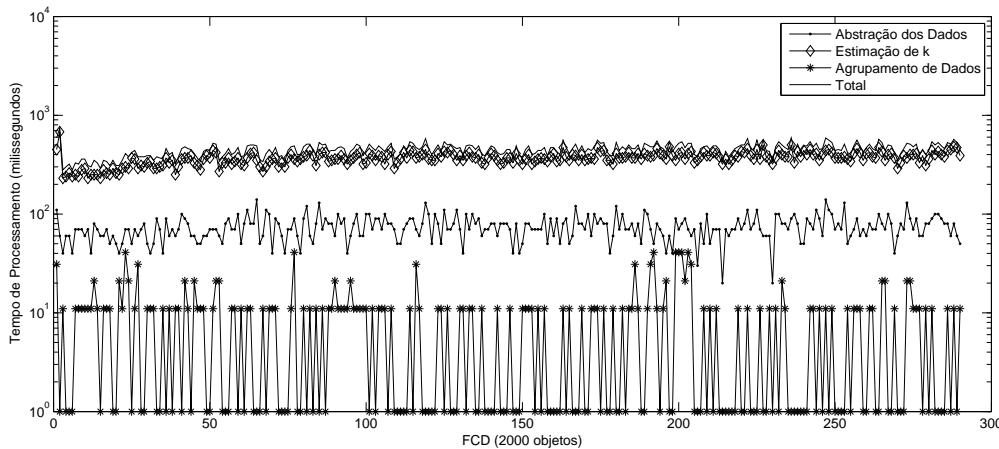


(a) Tempo de processamento de cada componente do algoritmo SKM-BkM - Base de dados *Forest CoverType*.

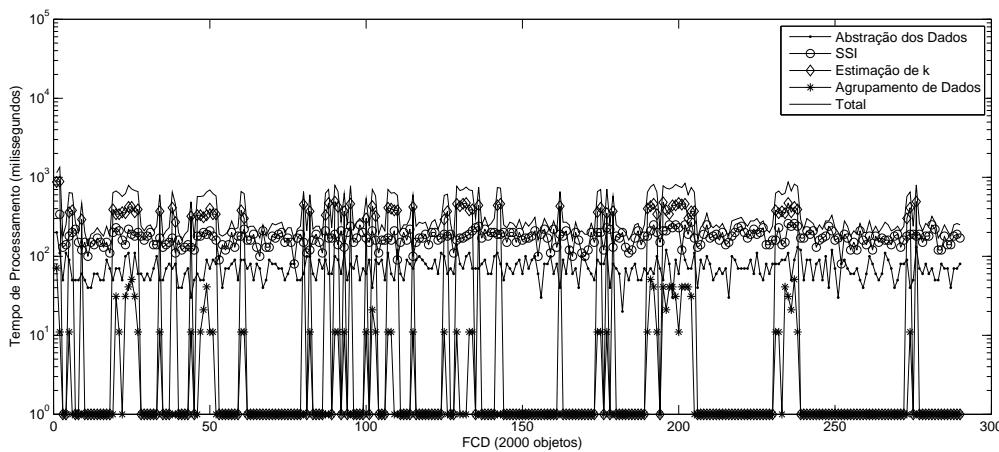


(b) Tempo de processamento de cada componente do algoritmo SKM-IBkM - Base de dados *Forest CoverType*.

Figura 5.53: Tempo de processamento em milissegundos de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) para os algoritmos SKM-BkM (Figura 5.53a) e SKM-IBkM (Figura 5.53b) - Base de dados *Forest CoverType*.



(a) Tempo de processamento de cada componente do algoritmo SKM-MEOk - Base de dados *Forest CoverType*.



(b) Tempo de processamento de cada componente do algoritmo SKM-MEOkl - Base de dados *Forest CoverType*.

Figura 5.54: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos SKM-MEO k (Figura 5.54a) e SKM-MEO k I (Figura 5.54b) - Base de dados *Forest CoverType*.

Considerando os algoritmos baseados no *Stream LSearch*, na Figura 5.55 são apresentados os tempos de processamento médios dos algoritmos SLS-B k M, SLS-IB k M, SLS-MEO k e SLS-MEO k I. Pode-se observar que o algoritmo SLS-MEO k apresenta o maior tempo de processamento, enquanto que a sua versão incremental (SLS-MEO k I) e o algoritmo SLS-IB k M apresentam frequentemente os menores tempos de processamento. Além disso, pode-se observar nas Figuras 5.56a e 5.56b que o tempo de processamento da componente de agrupamento de dados é superior ao da componente de estimativa de \hat{k} para os algoritmos SLS-B k M e SLS-IB k M. Porém, para os algoritmos SLS-MEO k e SLS-MEO k I (ver Figuras 5.57a e 5.57b) o custo de processamento da componente de estimativa de \hat{k} apresenta os maiores valores. Esse comportamento ocorre devido ao custo

de executar o algoritmo BkM para agrupar 2.000 objetos ser inferior ao aplicar o algoritmo MEO_k para agrupar a mesma quantidade de objetos. A cada iteração do algoritmo BkM a quantidade de objetos a ser agrupada é reduzida pela metade devido as divisões realizadas nos grupos de dados. Já para o algoritmo MEO_k a quantidade de objetos é sempre a mesma — *i.e.*, 2.000 objetos.

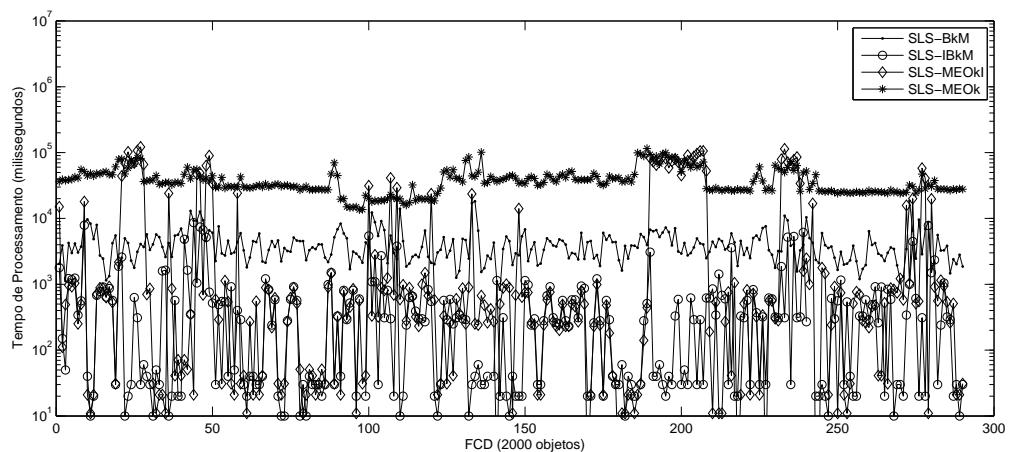
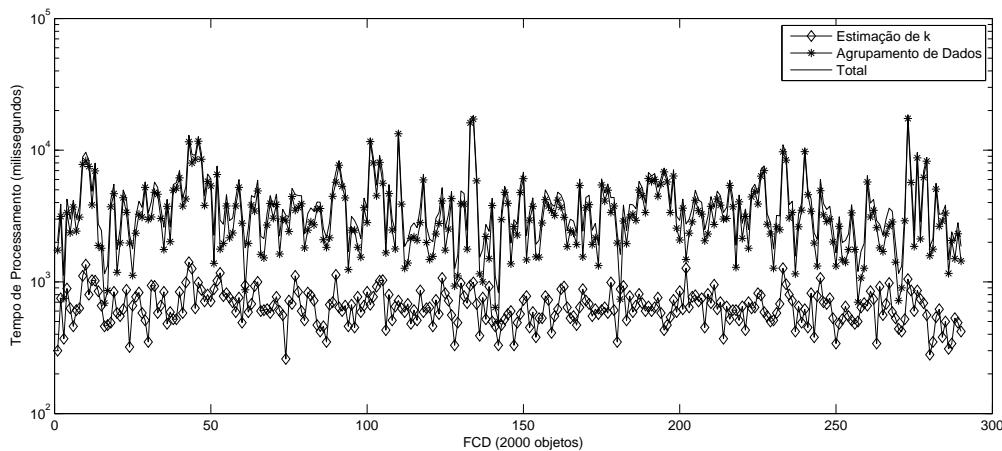
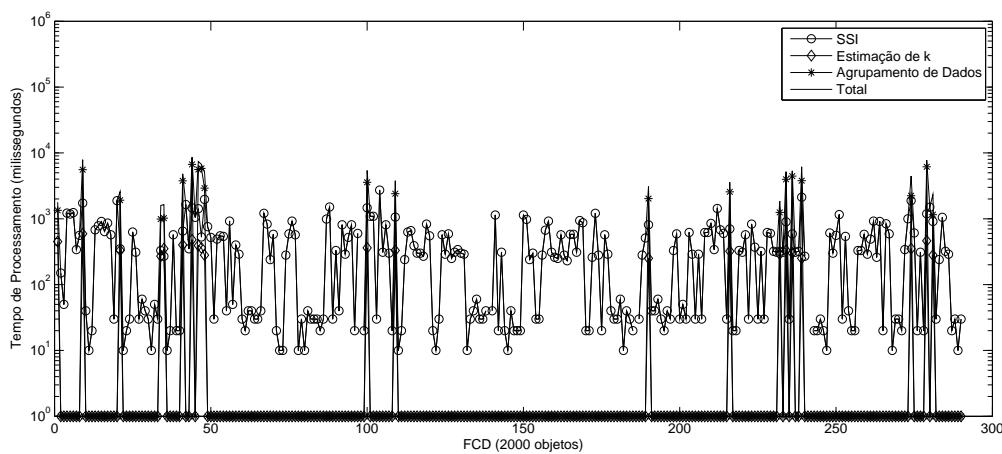


Figura 5.55: Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos baseados no *Stream LSearch* (SLS-BkM, SLS-IBkM, SLS-MEOkl e SLS-MEOk) - Base de dados *Forest CoverType*.

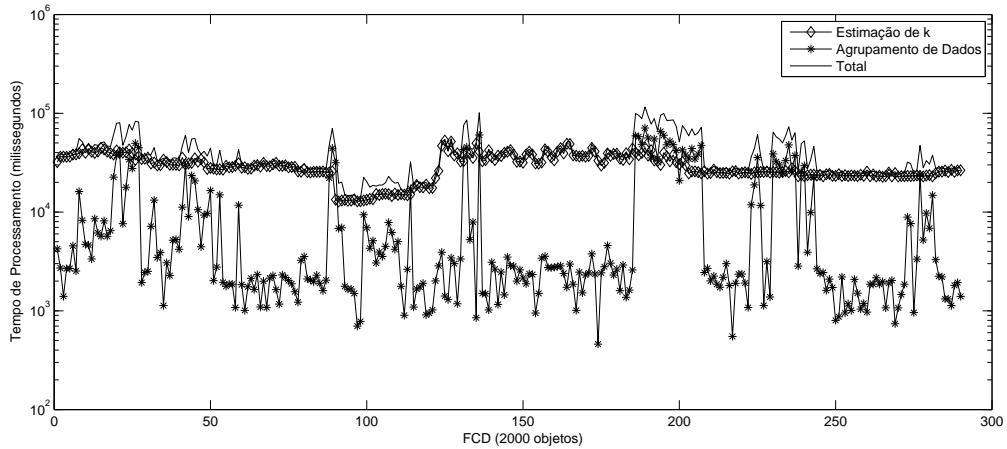


(a) Tempo de processamento (escala logarítmica) de cada componente do algoritmo SLS-BkM - Base de dados *Forest CoverType*.

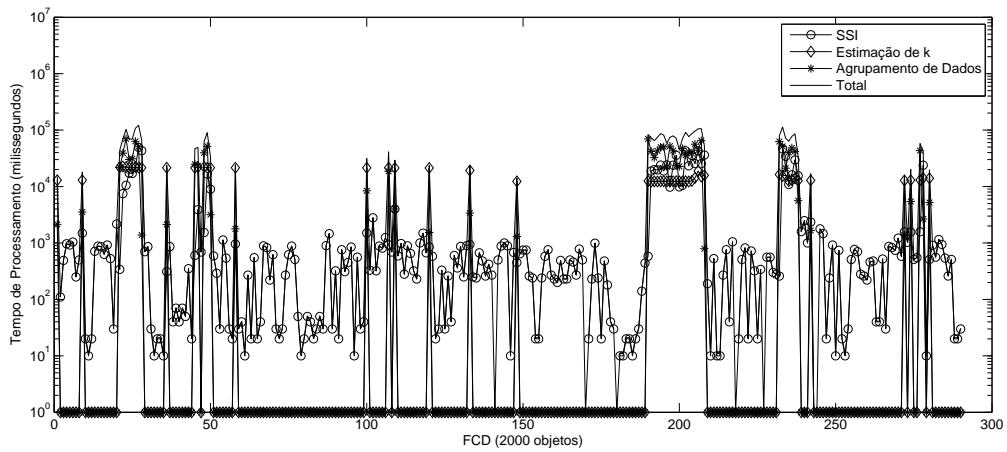


(b) Tempo de processamento (escala logarítmica) de cada componente do algoritmo SLS-IBkM - Base de dados *Forest CoverType*.

Figura 5.56: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimação de \hat{k} e agrupamento de dados) e sua soma (total) para os algoritmos SLS-BkM (Figura 5.56a) e SLS-IBkM (Figura 5.56b).



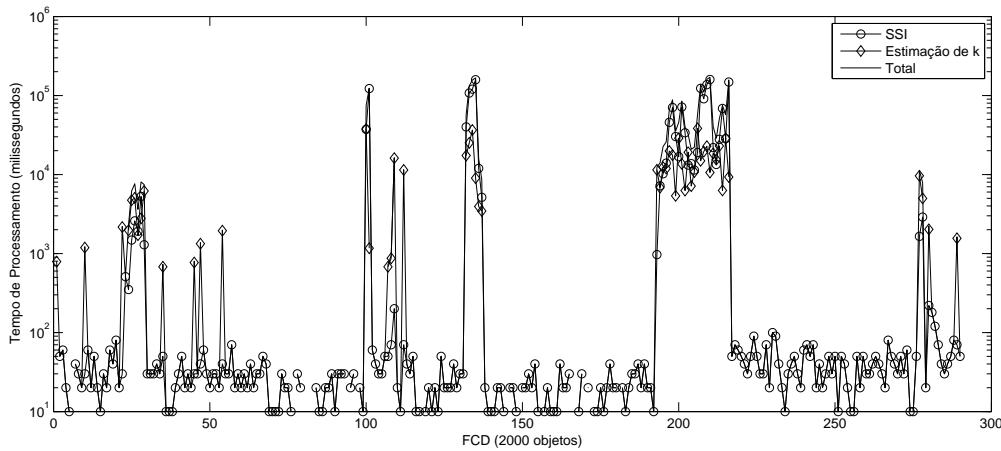
(a) Tempo de processamento (escala logarítmica) de cada componente do algoritmo SLS-MEO k - Base de dados *Forest CoverType*.



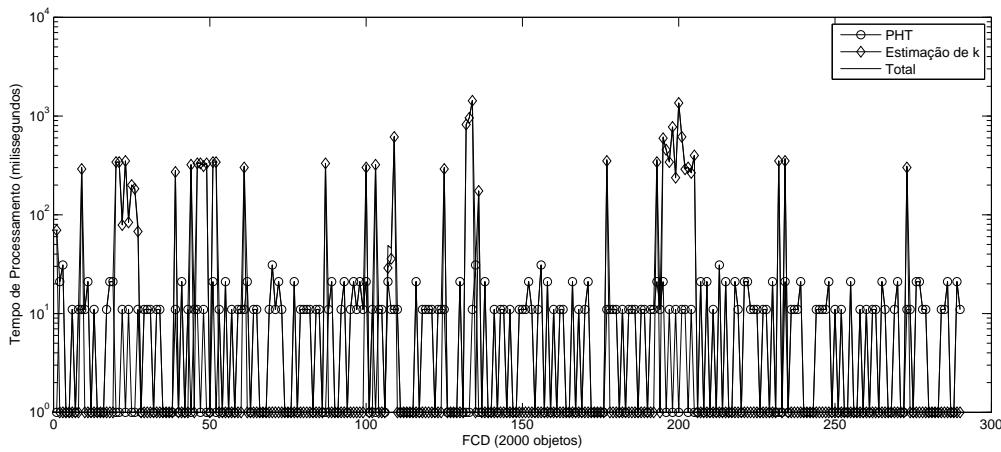
(b) Tempo de processamento (escala logarítmica) de cada componente do algoritmo SLS-MEO k I - Base de dados *Forest CoverType*.

Figura 5.57: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (abstração de dados, silhueta simplificada incremental (SSI), estimativa de \hat{k} e agrupamento de dados) e sua soma (total) dos algoritmos SLS-MEO k (Figura 5.57a) e SLS-MEO k I (Figura 5.57b).

Considerando os algoritmos FEACS-PHT e FEACS-ISS, no gráfico da Figura 5.58 são apresentados os tempos de processamento médios dos algoritmos. Pode-se notar que o algoritmo FEACS-PHT apresenta frequentemente o menor tempo de processamento. Os picos nas curvas de tempo de processamento dos algoritmos estão relacionados com os momentos em que a estimativa de \hat{k} foi utilizada. Na Figura 5.59 pode-se observar os momentos em que o algoritmo de estimativa de \hat{k} é utilizado, justificando os picos na curva de tempo de processamento.



(a) Tempo de processamento (escala logarítmica) de cada componente do algoritmo FEACS-SSI - Base de dados *Forest CoverType*.



(b) Tempo de processamento (escala logarítmica) de cada componente do algoritmo FEACS-PHT - Base de dados *Forest CoverType*.

Figura 5.59: Tempo de processamento em milissegundos (escala logarítmica) de cada componente (silhueta simplificada incremental (SSI), atualização incremental combinada com o PHT e estimativa de \hat{k}) e sua soma (total) dos algoritmos FEACS-SSI 5.59a e FEACS-PHT 5.59b - Base de dados *Forest CoverType*.

A Tabela 5.13 apresenta os resultados do teste de significância estatística para comparações pareadas entre os algoritmos. Seguindo a mesma notação adotada para a Tabela 5.8, pode-se observar que os algoritmos SLS-IB k M e SLS-MEO k obtiveram os melhores resultados considerando o critério da ss. Considerando os tempos de processamento reportados na Tabela 5.14, pode-se observar que os algoritmos FEACS-PHT e SKM-B k M obtiveram os melhores resultados. Similarmente ao ocorrido para a base de dados KDD-Cup99.

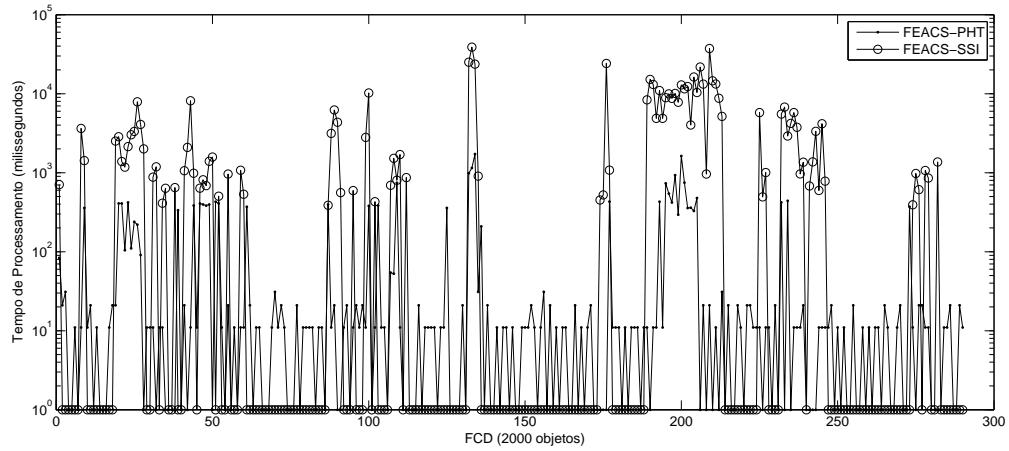


Figura 5.58: Tempo de processamento em milissegundos (escala logarítmica) dos algoritmos FEACS-PHT e FEACS-SSI para a base de dados *Forest CoverType*.

Tabela 5.13: Análise de diferenças estatisticamente significativas para os valores de ss para a base de dados *Forest CoverType*. O símbolo \odot indica que não foram encontradas diferenças significativas, + indica que os resultados para o algoritmo na linha i são superiores ao das colunas j e oposto é representado pelo símbolo $-$.

	SKM-IB k M	SLS-MEO k	SKM-MEO k I	FEACS-ISS	CLS-MEO k I	CLS-IB k M	FEACS-PHT	CLS-MEO k	SKM-B k M	CLS-B k M	SLS-B k M	SLS-MEO k I	SKM-MEO k	SLS-IB k M
SKM-IB k M	■	○	+	+	+	+	+	+	+	+	+	+	+	+
SLS-MEO k	○	■	○	○	+	+	+	+	+	+	+	+	+	+
SKM-MEO k I	-	○	○	○	○	○	○	+	+	+	+	+	+	+
FEACS-ISS	-	○	○	■	○	○	○	○	+	+	+	+	+	+
CLS-MEO k I	-	-	○	○	○	○	○	○	○	+	+	+	+	+
CLS-IB k M	-	-	○	○	○	○	■	○	○	○	+	+	+	+
FEACS-PHT	-	-	○	○	○	○	■	○	○	○	○	+	+	+
CLS-MEO k	-	-	-	○	○	○	○	■	○	○	○	+	+	+
SKM-B k M	-	-	-	-	○	○	○	○	■	○	○	+	+	+
CLS-B k M	-	-	-	-	-	○	○	○	○	■	○	○	+	+
SLS-B k M	-	-	-	-	-	○	○	○	○	○	■	○	+	+
SLS-MEO k I	-	-	-	-	-	-	-	-	-	○	○	■	○	○
SKM-MEO k	-	-	-	-	-	-	-	-	-	-	-	○	■	○
SLS-IB k M	-	-	-	-	-	-	-	-	-	-	-	○	○	■

Tabela 5.14: Resultados dos testes de significância estatística para os valores de tempo de processamento para a base de dados *Forest CoverType*. O símbolo \odot indica que não foram encontradas diferenças significativas, + indica que os resultados para o algoritmo na linha i são superiores ao das colunas j e oposto é representado pelo símbolo $-$.

5.4 Considerações Finais

Neste capítulo foram apresentados os resultados dos experimentos para avaliar os algoritmos propostos de agrupamento em FCD. Foram utilizadas oito bases de dados (seis bases de dados artificiais e duas base de dados reais).

Os experimentos mostram o potencial dos algoritmos propostos para processar FCD bem como suas limitações. Para as bases de dados artificiais foram analisados a qualidade dos agrupamentos obtidos, o tempo de processamento dos algoritmos bem como o custo de processamento de cada componente dos algoritmos e a sensibilidade dos algoritmos em reagir às mudanças do número de grupos. Todos os algoritmos obtiveram partições de boa qualidade. No que diz respeito ao custo computacional, destacam-se os algoritmos baseados no StreamKm++ e os algoritmos evolutivos FEACS-SSI e FEACS-PHT.

Para as duas bases de dados reais foi observado se os algoritmos reagiam às mudanças de classes, as quais sugerem que novos padrões nos dados estão sendo observados. Tais mudanças se refletiram nos grupos de dados obtidos. Essas observações ficaram mais evidentes para a base de dados KDDCup99, na qual diferentes tipos de ataques fizaram com que o algoritmo reagisse a essa mudança nos padrões dos dados para capturar um novo agrupamento que represente novos conceitos sendo observados no FCD. Assim como nos experimentos com as bases sintéticas, foi possível observar que os algoritmos SKM-BkM, SKM-IBkM, FEACS-SSI e FEACS se destacaram realizando o processamento das bases de dados reais de maneira eficiente com nível de qualidade muito bom (aproximadamente 0,8 de SS). Conforme sugerido por [Kaufman e Rousseeuw \(1990\)](#), valores de silhueta acima de 0,71 implicam em partições de excelente qualidade. Ainda que não se possa afirmar que tal limiar se aplique estritamente pra a silhueta simplificada utilizada nesse trabalho, resultados mais recentes ([Vendramin et al., 2010](#)) sugerem que os valores obtidos correspondem a partições de boa qualidade, dessa forma permitindo avaliar apropriadamente os tempos computacionais envolvidos no processamento dos algoritmos estudados.

No próximo capítulo são apresentadas as principais contribuições dessa tese, bem como sugestões de trabalhos futuros.

Conclusões

6.1 Contribuições

Neste trabalho foram propostos, estudados e avaliados 14 algoritmos de agrupamento para FCD que estimam o número de grupos automaticamente a partir dos dados. Tais algoritmos foram concebidos a partir de uma extensa pesquisa bibliográfica (Silva et al., 2013) que permitiu identificar as lacunas existentes nessa área do conhecimento. Em particular observadas características comuns dos algoritmos previamente reportados na literatura e, a partir do estudo desses algoritmos, foram determinadas suas componentes que prioritariamente mereciam maior atenção. Dessa maneira, o estudo e o desenvolvimento de duas novas componentes (componente de estimação do valor de k e sua versão incremental) possibilitaram sua inclusão em três algoritmos estado-da-arte da literatura: *Stream LSearch* (O'Callaghan et al., 2002), *CluStream* (Aggarwal et al., 2003) e *StreamKM++* (Ackermann et al., 2010). Sob uma perspectiva mais ampla, pode-se dizer que esse estudo teve como principal resultado o desenvolvimento de um *framework* que permite estender os algoritmos da literatura que não apresentam a capacidade de estimar o valor de k de maneira automática a partir dos dados, conforme descrito no Capítulo 3. Aproveitando estudos anteriores sobre o desenvolvimento de algoritmos evolutivos computacionalmente eficientes para agrupamento de dados em aplicações que não envolviam FCD, foram desenvolvidos também dois algoritmos evolutivos para agrupamento em FCD, conforme descrito no Capítulo 4. Como uma contribuição derivada e complementar do estudo realizado, foi desenvolvido um protótipo de sistema que permite acompanhar visualmente a evolução dos grupos, conforme apresentado no Apêndice A.

Todos os algoritmos desenvolvidos são adaptações do popular algoritmo das k -Médias pra lidar com FCD, assumindo-se que o número de grupos é desconhecido *a priori* e que pode ser automaticamente estimado a partir dos dados. Neste contexto, é conveniente relembrar que a seguinte hipótese de pesquisa norteou o presente trabalho:

"É possível desenvolver algoritmos eficazes para agrupamento de dados que estimem o número de grupos (k) automaticamente sem comprometer significativamente a eficiência computacional do processo de agrupamento em ambientes dinâmicos com restrições de tempo e armazenamento encontrados em FCD."

Para comprovar essa hipótese de pesquisa, foram propostos 14 algoritmos de agrupamento de dados em FCD que estimam o valor de k automaticamente a partir dos dados. Doze desses algoritmos são combinações de três algoritmos que representam o estado da arte mas que não são capazes de estimar o número de grupos a partir dos dados, e outros dois são algoritmos evolutivos para agrupamento em FCD. A motivação para a investigação de algoritmos evolutivos para agrupamento em FCD é derivada do sucesso desses algoritmos em estimar o número de grupos em bases de dados tradicionais, em particular quando o número de grupos é desconhecido *a priori* (Naldi, 2011). Para comprovar experimentalmente a hipótese levantada, experimentos foram realizados em seis bases de dados artificiais e duas bases de dados reais. Experimentos controlados com bases de dados artificiais foram realizados para simular a dinâmica dos grupos em FCD. A partir de partições de referência é possível obter conclusões detalhadas sobre os diferentes comportamentos dos algoritmos. Nas bases de dados reais não é possível conhecer a estrutura de agrupamento dos dados, porém é possível observar como as partições de dados mudam quando objetos de diferentes classes são observados no FCD.

Na Tabela 6.1 são sumarizadas as componentes de cada algoritmo estudado. Pode-se observar que os algoritmos StreamKM++ e CluStream contêm uma componente de abstração de dados, o que permite sumarizar o FCD por meio de coresets ou microgrupos para posteriormente serem agrupados pela componente de agrupamento de dados. Comparando esses algoritmos com os algoritmos baseados no Stream LSearch e no FEACS, que não contêm a componente de abstração, a quantidade de objetos utilizada pela componente de agrupamento de dados é maior. Entretanto, os algoritmos baseados no Stream LSearch e no FEACS não necessitam de um custo de processamento adicional para a manutenção dos dados na componente de abstração de dados, que no caso dos algoritmos baseados no CluStream é de ordem quadrática com o número de microgrupos no pior caso. Com relação à componente de estimativa de \hat{k} , observa-se que esta é uma propriedade dos algoritmos BkM, MEO k e dos algoritmos evolutivos. Nos experimentos foi observado que os algoritmos que utilizam o algoritmo BkM e o algoritmo evolutivo foram os mais rápidos. Entretanto, os algoritmos que utilizam o BkM como componente de estimativa de \hat{k} são mais susceptíveis a mínimos locais do que os algoritmos que utilizam o MEO k . Para a componente de detecção de mudança, tem-se a SSI e o PHT que podem reduzir a frequê-

Tabela 6.1: Componentes dos algoritmos estudados.

Algoritmo	Abstração dos Dados	Estimação de \hat{k}	Detecção de Mudança	Agrupamento de Dados	Algoritmos Resultantes
StreamKM++	coreset	BkM MEO k	SSI (IBkM MEO k I)	k -Means++	SKM-BkM/IBkM SKM-MEO k /MEO k I
CluStream	microgrupos			k -Means	CLS-BkM/IBkM CLS-MEO k /MEO k I
Stream LSearch	—			LSearch	SLS-BkM/IBkM SLS-MEO k /MEO k I
FEACS	—	Algoritmos Evolutivos	SSI / PHT	—	FEACS-SSI/PHT

cia de uso da componente de estimação de \hat{k} , sendo utilizados em momentos em que uma mudança for observada na qualidade da partição (SSI) ou nas distâncias entre os objetos e os centroides dos grupos (PHT). Os algoritmos que não utilizam uma componente de detecção de mudança (CLS-BkM, CLS-MEO k , SLS-BkM, SLS-MEO k , SKM-BkM e SKM-MEO k) utilizam a componente de estimação de \hat{k} em todo o FCD. Em termos de custo de processamento, conforme observado nos experimentos a SSI possui um custo maior do que o PHT. Na SSI, quando uma mudança é detectada o algoritmo das k -médias é utilizado para encontrar uma partição de dados com k e $k + 1$ grupos e então é mantida a partição com maior valor de silhueta. Ao processar um bloco de objetos e ao observar que nesse bloco algum objeto ocasionou uma mudança na partição de k para $k + 1$ grupos, o algoritmo de estimação de \hat{k} é aplicado. Já no PHT o algoritmo calcula distâncias entre os objetos e os centroides dos grupos e monitora a distância média entre o objeto e centroide do grupo mais próximo. Quando uma mudança é detectada o algoritmo de estimação de \hat{k} é aplicado. Nesse sentido, ao comparar os métodos SSI e PHT pode-se observar que o PHT apresenta um custo de processamento menor pelo fato de calcular apenas distâncias e monitorar a distância média. Entretanto, em termos de parâmetros, o PHT contém três parâmetros críticos (λ_a , λ_w e σ) que estão relacionados com os limiares de detecção. Considerando a componente de agrupamento de dados, o algoritmo LSearch apresentou o maior valor de tempo de processamento comparado às demais componentes. A razão para o seu custo de processamento elevado está relacionada com o procedimento de busca iterativo utilizado para encontrar uma partição com k grupos. Inicialmente assume-se que a partição contém um grupo de dados e o valor do seu custo é o maior possível. A cada iteração, o valor do custo da partição de dados é ajustado para permitir incluir grupos de dados. Pode-se observar que à medida que o número de grupos aumenta o seu custo de processamento também aumenta. Esse processo iterativo apresentou um custo elevado comparado aos algoritmos k -Means e k -Means++.

Com base nas características analisadas dos algoritmos e nos resultados dos experimentos realizados, observou-se que os algoritmos FEACS-SSI, FEACS-PHT e os algoritmos baseados no StreamKM++ se destacaram na maioria dos cenários avaliados e dessa forma fazem parte de um grupo de algoritmos eficientes para agrupar FCD. Portanto, pode-

se sugerir o algoritmo FEACS-SSI por não necessitar de um custo adicional de memória e processamento para manter uma componente de abstração dos dados. Além disso, este algoritmo não contém uma componente adicional de agrupamento de dados já que a componente de estimativa de \hat{k} já fornece uma partição de dados para os dados processados. Além disso, o FEACS-SSI se apresentou eficiente em relação aos algoritmos que utilizam o MEO k para estimativa de \hat{k} .

6.1.1 Publicações Geradas e Artigos Submetidos

- Silva, Jonathan A. and Hruschka, E.R. **Extending k-Means-Based Algorithms for Evolving Data Streams with Variable Number of Clusters.** *The tenth International Conference on Machine Learning and Applications (ICMLA)*, pp. 14-19, IEEE Press, 2011.
- Silva, Jonathan A., Faria, Elaine R., Barros, Rodrigo C., Hruschka, Eduardo R., Carvalho, André C. P. L. F. de, Gama, João. **Data Stream Clustering: A Survey.** pp. 1-31, *ACM Computing Surveys*, 2013.
- Silva, Jonathan A., Hruschka, Eduardo R. **A Support System for Clustering Data Streams with a Variable Number of Clusters.** pp. 1-26, *ACM Transactions on Autonomous and Adaptive Systems*, 2014.
- Silva, Jonathan A., Hruschka, Eduardo R., Gama, João **An Evolutionary Algorithm for Clustering Data Streams with a Variable Number of Clusters.** pp. 1-23, *Evolutionary Computation, under revision*.

6.2 Limitações e Trabalhos Futuros

Os algoritmos desenvolvidos nessa tese possuem diversas limitações, dentre as quais se destacam aquelas inerentes ao algoritmo das k -Médias. Há, portanto, muito espaço para novos desenvolvimentos, desde que as questões de eficiência computacional sejam mantidas em mente, pois cenários de FCD são naturalmente hostis a algoritmos de alta complexidade computacional. Nesse contexto, trabalhos futuros promissores incluem:

- **Tratamento de ruídos e outliers:** realizar experimentos em bases de dados com diferentes taxas de ruído e *outliers* no intuito de verificar como as partições serão influenciadas. Em seguida, propor mecanismos para lidar com ruídos e *outliers* de modo a reduzir suas influências na qualidade das partições. O desafio em detectar *outliers* e ruídos em FCD está em diferenciar se o dados que estão surgindo representam um novo grupo de dados ou se são dados ruidosos. A grande maioria dos algoritmos para minerar FCD trabalham com uma janela de dados. Nesse sentido,

alguns algoritmos da literatura utilizam uma quantidade de memória adicional para armazenar dados que aparentam ser ruidosos para ter certo grau de confiança na decisão a ser tomada: determinar se os dados são realmente ruidosos ou pertencem a um novo grupo. Em Cao et al. (2006), por exemplo, uma componente adicional que contém microgrupos que estão relacionados aos *outliers* é adotada. Os microgrupos relacionados aos *outliers* podem se tornar microgrupos denominados potenciais *i.e.*, microgrupos que não são classificados como ruidosos. Este processo de determinação de *outliers* está relacionado com o raio dos microgrupos. Caso os dados não estejam dentro desse limite de raio esses são inseridos nos microgrupos denominados de *outliers* mais próximo. Quando o peso dos microgrupos *outliers* se tornam significativos (maior que um limiar) estes são considerados potenciais.

- **Desenvolvimento de algoritmos distribuídos:** adaptar os algoritmos propostos para realizar o processamento distribuído pode aumentar ainda mais o ganho em termos de tempo de processamento dos algoritmos. Especificamente, é promissora a aplicação de processamento distribuído da componente de estimativa de \hat{k} dos algoritmos. Nos experimentos realizados pode-se observar que, embora a componente de estimativa de \hat{k} das versões incrementais dos algoritmos são utilizadas com pouco frequência, em momentos em que uma mudança na qualidade da partição é detectada, ainda o tempo de processamento dessa componente é maior em relação às demais componentes. A realização de um processamento distribuído durante a avaliação de diferentes partições de dados pode aumentar a velocidade do algoritmo. Em (Naldi, 2011) são apresentadas maneiras de realizar o processamento distribuído de algoritmos para estimativa do número de grupos.
- **Análise de sensibilidade dos parâmetros:** realizar experimentos para avaliar a sensibilidade dos algoritmos em relação aos seus parâmetros. Um dos parâmetros que é comum a todos os algoritmos é o tamanho da janela de dados. Outro parâmetro importante é o número de dados armazenados na componente de abstração de dados. A determinação dos valores desses parâmetros em geral é baseada no recurso computacional disponível. Porém, avaliar valores diferentes para esses parâmetros podem fornecer indícios de quais valores são mais promissores para se obter partições de alta qualidade e minimizar o tempo de processamento.

Referências

- ACKERMANN, M. R.; LAMMERSEN, C.; MÄRTENS, M.; RAUPACH, C.; SOHLER, C.; SWIERKOT, K. StreamKM++: A Clustering Algorithms for Data Streams. In: *Proc. of the ALENEX*, SIAM, pp. 173–187, 2010. (Not cited.)
- ACKERMANN, M. R.; MÄRTENS, M.; RAUPACH, C.; SWIERKOT, K.; LAMMERSEN, C.; SOHLER, C. Streamkm++: A clustering algorithm for data streams. *ACM Journal of Experimental Algorithms*, vol. 17, no. 1, 2012. (Not cited.)
- AGARWAL, P. K.; HAR-PELED, S.; VARADARAJAN, K. R. Approximating extent measures of points. *Journal of the ACM*, vol. 51, no. 4, pp. 606–635, 2004. (Not cited.)
- AGGARWAL, C. C. *Data streams – models and algorithms*. Springer, 2007. (Not cited.)
- AGGARWAL, C. C.; HAN, J.; WANG, J.; YU, P. S. A framework for clustering evolving data streams. In: *Proceedings of the 29th Conference on Very Large Data Bases*, pp. 81–92, 2003. (Not cited.)
- AGGARWAL, C. C.; HAN, J.; WANG, J.; YU, P. S. A framework for projected clustering of high dimensional data streams. In: *Proc. of the VLDB*, pp. 852–863, 2004. (Not cited.)
- ALVES, V.; CAMPELLO, R.; HRUSCHKA, E. Towards a fast evolutionary algorithm for clustering. In: *IEEE Congress on Evolutionary Computation (CEC'06)*, pp. 1776–1783, 2006. (Not cited.)
- ARABIE, P.; HUBERT, L. J. An overview of combinatorial data analysis. In: ARABIE, P.; HUBERT, L. J.; SOETE, G. D., eds. *Clustering and Classification*, World Scientific, pp. 5–63, 1996. (Not cited.)
- ARTHUR, D.; VASSILVITSKII, S. k-means++: the advantages of careful seeding. In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035, 2007. (Not cited.)

- BABCOCK, B.; BABU, S.; DATAR, M.; MOTWANI, R.; WIDOM, J. Models and issues in data stream systems. In: *PODS '02: Proceedings of the 21th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, ACM Press, pp. 1–16, 2002. (Not cited.)
- BABCOCK, B.; DATAR, M.; MOTWANI, R.; O'CALLAGHAN, L. Maintaining variance and k-medians over data stream windows. In: *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, ACM, pp. 234–243, 2003. (Not cited.)
- BÄCK; FOGEL, D.; MICHALEWICZ, Z., eds. *Evolutionary computation 1: Basic algorithms and operators*. Institute of Physics Publishing, Bristol, 2000. (Not cited.)
- BĀDOIU, M.; HAR-PELED, S.; INDYK, P. Approximate clustering via core-sets. In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, ACM Press, pp. 250–257, 2002 (STOC '02,). (Not cited.)
- BARBARÁ, D. Requirements for clustering data streams. *SIGKDD Explorations (Special Issue on Online, Interactive, and Anytime Data Mining)*, vol. 3, pp. 23–27, 2002. (Not cited.)
- BENNETT, C.; HAUSER, K. Artificial intelligence framework for simulating clinical decision-making: A markov decision process approach. *Artificial Intelligence in Medicine*, vol. 57, no. 1, pp. 9–19, 2013. (Not cited.)
- BIFET, A.; HOLMES, G.; KIRKBY, R.; PFAHRINGER, B. Moa: Massive online analysis. *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010. (Not cited.)
- BRADLEY, P. S.; FAYYAD, U. M.; REINA, C. Scaling clustering algorithms to large databases. In: *Proceedings of Knowledge Discovery and Data Mining*, AAAI Press, pp. 9–15, 1998. (Not cited.)
- CAO, F.; ESTER, M.; QIAN, W.; ZHOU, A. Density-based clustering over an evolving data stream with noise. In: *Proceedings of the Sixth SIAM International Conference on Data Mining*, SIAM, pp. 328–339, 2006. (Not cited.)
- CHARIKAR, M.; GUHA, S. Improved combinatorial algorithms for the facility location and k-median problems. In: *40th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, pp. 378–388, 1999. (Not cited.)
- CHEN, Y.; TU, L. Density-based clustering for real-time stream data. In: *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM Press, pp. 133–142, 2007. (Not cited.)

- DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006. (Not cited.)
- DOMINGOS, P.; HULTEN, G. A general method for scaling up machine learning algorithms and its application to clustering. In: *Proceedings of the 8th International Conference on Machine Learning*, Morgan Kaufmann, pp. 106–113, 2001. (Not cited.)
- EIBEN, A. E.; SMITH, J. E. *Introduction to evolutionary computing (natural computing series)*. Springer, 2003. (Not cited.)
- ESTER, M.; KRIEGEL, H.-P.; SANDER, J.; XU, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In: *2nd International Conference on Knowledge Discovery and Data Mining*, pp. 226–231, 1996. (Not cited.)
- EVERITT, B. S.; LANDAU, S.; LEESE, M. *Cluster analysis*. Arnold Publishers, 2001. (Not cited.)
- FALKENAUER, E. *Genetic algorithms and grouping problems*. John Wiley & Sons, Inc., 1998. (Not cited.)
- FARNSTROM, F.; LEWIS, J.; ELKAN, C. Scalability for clustering algorithms revisited. *SIGKDD Exploration*, pp. 51–57, 2000. (Not cited.)
- FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From data mining to knowledge discovery: an overview. In: *Advances in knowledge discovery and data mining*, Menlo Park, CA, USA: American Association for Artificial Intelligence, 1996. (Not cited.)
- FOGEL, D. B. *Evolutionary computation: Towards a new philosophy of machine intelligence*. IEEE Press, 1999. (Not cited.)
- FOLINO, G.; PIZZUTI, C.; SPEZZANO, G. Mining distributed evolving data streams using fractal gp ensembles. In: *Proceedings of the 10th European conference on Genetic programming*, Springer-Verlag, pp. 160–169, 2007. (Not cited.)
- FREITAS, A. *Evolutionary algorithms for data mining*, vol. The Data Mining and Knowledge Discovery Handbook Springer, pp. 435–467, 2005. (Not cited.)
- FREITAS, A. A. *Data mining and knowledge discovery with evolutionary algorithms*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002. (Not cited.)
- FRIEDMAN, M. A Comparison of Alternative Tests of Significance for the Problem of m Rankings. *The Annals of Mathematical Statistics*, vol. 11, no. 1, pp. 86–92, 1940. (Not cited.)

- GAMA, J. *Knowledge discovery from data streams*. Chapman Hall/CRC, 2010. (Not cited.)
- GAMA, J.; GABER, M. M. *Learning from data streams: Processing techniques in sensor networks*. Springer, 2007. (Not cited.)
- GAMA, J.; MEDAS, P.; CASTILLO, G.; RODRIGUES, P. P. Learning with Drift Detection. In: *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence (SBIA 2004)*, pp. 286–295, 2004. (Not cited.)
- GAMA, J.; PEREIRA, P. R.; LOPES, L. Clustering distributed sensor data streams using local processing and reduced communication. *Intelligent Data Analysis*, vol. 15, no. 1, pp. 3–28, 2011. (Not cited.)
- GONZALEZ, T. F. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985. (Not cited.)
- GUHA, S.; MISHRA, N.; MOTWANI, R.; O'CALLAGHAN, L. Clustering data streams. In: *IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society, pp. 359–366, 2000. (Not cited.)
- HAN, J.; KAMBER, M. *Data mining: Concepts and techniques*. Morgan Kaufmann, 2000. (Not cited.)
- HAR-PELED, S.; MAZUMDAR, S. On coresets for k-means and k-median clustering. In: *36th Annual ACM symposium on Theory of computing*, pp. 291–300, 2004. (Not cited.)
- HASSAN, Y.; CRANDALL, J. Genetic algorithms in repeated matrix games: The effects of algorithmic modifications and human input with various associates. pp. 28–35, 2013. (Not cited.)
- HOLLANDER, M.; WOLFE, D. A. *Nonparametric statistical methods*. 2nd edn.. Wiley, 1999. (Not cited.)
- HORTA, D.; CAMPELLO, R. J. G. B. Fast evolutionary algorithms for relational clustering. In: *9th International Conference on Intelligent Systems Design and Applications (ISDA'09)*, IEEE Computer Society, pp. 1456–1462, 2009. (Not cited.)
- HRUSCHKA, E.; DE CASTRO, L.; CAMPELLO, R. J. G. B. Evolutionary algorithms for clustering gene-expression data. In: *Data Mining, 2004. ICDM '04. Fourth IEEE International Conference on*, pp. 403–406, 2004. (Not cited.)
- HRUSCHKA, E. R.; CAMPELLO, R. J. G. B.; DE CASTRO, L. N. Evolving clusters in gene-expression data. *Information Sciences*, vol. 176, pp. 1898–1927, 2006. (Not cited.)

- HRUSCHKA, E. R.; CAMPELLO, R. J. G. B.; FREITAS, A. A.; DE CARVALHO, A. C. P. L. F. A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, vol. 39, no. 2, pp. 133–155, 2009. (Not cited.)
- ISAKSSON, C.; DUNHAM, M. H.; HAHSLER, M. Sostream: Self organizing density-based clustering over data stream. Springer, pp. 264–278, 2012 (*Lecture Notes in Computer Science*, vol.7376). (Not cited.)
- JAIN, A. K. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, vol. 31, pp. 651–666, 2009. (Not cited.)
- JAIN, A. K.; DUBES, R. C. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988. (Not cited.)
- JAIN, A. K.; MURTY, M. N.; FLYNN, P. J. Data clustering: a review. *ACM Computing Surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999. (Not cited.)
- JIANG, N.; GRUENWALD, L. Research issues in data stream association rule mining. *SIGMOD Record*, vol. 35, no. 1, pp. 14–19, 2006. (Not cited.)
- KAUFMAN, L.; ROUSSEEUW, P. J. *Finding groups in data an introduction to cluster analysis*. Wiley Interscience, 1990. (Not cited.)
- KRANEN, P.; ASSENT, I.; BALDAUF, C.; SEIDL, T. The clustree: indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems*, vol. 29, no. 2, pp. 249–272, 2011. (Not cited.)
- LEE, Y.-J.; YEH, Y.-R.; WANG, Y.-C. Anomaly detection via online oversampling principal component analysis. *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 7, pp. 1460–1470, 2013. (Not cited.)
- LIU, C. L. *Introduction to combinatorial mathematics*. McGraw-Hill, 1968. (Not cited.)
- MACQUEEN, J. B. Some Methods for Classification and Analysis of MultiVariate Observations. In: CAM, L. M. L.; NEYMAN, J., eds. *5th Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, 1967. (Not cited.)
- MASUD, M. M.; GAO, J.; KHAN, L.; HAN, J.; THURAISINGHAM, B. M. Classification and novel class detection in concept-drifting data streams under time constraints. *Knowledge and Data Engineering, IEEE Transactions on*, vol. 23, pp. 859 –874, 2011. (Not cited.)
- METWALLY, A.; AGRAWAL, D.; ABBADI, A. E. Duplicate detection in click streams. In: *Proceedings of the 14th international conference on World Wide Web*, ACM, pp. 12–21, 2005. (Not cited.)

- MEYERSON, A. Online facility location. *Foundations of Computer Science, Annual IEEE Symposium on*, pp. 426–431, 2001. (Not cited.)
- MITCHELL, T. M. *Machine learning*. New York: McGraw-Hill, 1997. (Not cited.)
- MOUSS, H.; MOUSS, D.; MOUSS, N.; SEFOUHI, L. Test of page-hinckley, an approach for fault detection in an agro-alimentary production system. In: *Control Conference, 2004. 5th Asian*, pp. 815–818, 2004. (Not cited.)
- NALDI, M. C. *Técnicas de combinação para agrupamento centralizado e distribuído de dados*. Ph.D. thesis, 2011. (Not cited.)
- NALDI, M. C.; CAMPELLO, R. J.; HRUSCHKA, E. R.; CARVALHO, A. C. Efficiency issues of evolutionary k-means. *Applied Soft Computing*, vol. 11, pp. 1938–1952, 2011. (Not cited.)
- NALDI, M. C.; FONTANA, A.; CAMPELLO, R. J. G. B. Comparison among methods for k estimation in k-means. In: *Proc. of the ISDA'09*, pp. 1006–1013, 2009. (Not cited.)
- O'CALLAGHAN, L.; MISHRA, N.; MEYERSON, A.; GUHA, S.; MOTWANI, R. Streaming-data algorithms for high-quality clustering. In: *18th International Conference on Data Engineering*, pp. 685–694, 2002. (Not cited.)
- PAGE, E. S. Continuous inspection schemes. *Biometrika*, vol. 41, 1954. (Not cited.)
- PAL, N. R.; BEZDEK, J. C. On cluster validity for the fuzzy c-means model. *IEEE Transactions on Fuzzy Systems*, vol. 3, pp. 370–379, 1995. (Not cited.)
- PARK, N. H.; LEE, W. S. Cell trees: An adaptive synopsis structure for clustering multi-dimensional on-line data streams. *Data and Knowledge Engineering*, vol. 63, no. 2, pp. 528–549, 2007. (Not cited.)
- REN, J.; MA, R. Density-based data streams clustering over sliding windows. In: *Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, Piscataway, NJ, USA: IEEE Press, pp. 248 –252, 2009. (Not cited.)
- RODRIGUES, P. P.; AO GAMA, J.; LOPES, L. Clustering distributed sensor data streams. In: *Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases - Part II*, ECML PKDD '08, Springer-Verlag, pp. 282–297, 2008 (ECML PKDD '08,). (Not cited.)
- SHAFI, K.; ABBASS, H. A. An adaptive genetic-based signature learning system for intrusion detection. *Expert Syst. Appl.*, pp. 12036–12043, 2009. (Not cited.)

- SHAH, R.; KRISHNASWAMY, S.; GABER, M. M. Resource-aware very fast k-means for ubiquitous data stream mining. In: *2nd International Workshop on Knowledge Discovery in Data Streams, 16th European Conference on Machine Learning (ECML'05)*, 2005. (Not cited.)
- SILVA, A. J.; HRUSCHKA, E. Extending k-means-based algorithms for evolving data streams with variable number of clusters. In: *Machine Learning and Applications and Workshops (ICMLA), 2011 10th International Conference on*, pp. 14–19, 2011. (Not cited.)
- SILVA, A. J.; HRUSCHKA, E. A support system for clustering data streams with a variable number of clusters. In: *ACM Transactions on Autonomous and Adaptive Systems*, p. to be appeared, 2014. (Not cited.)
- SILVA, J. A.; FARIA, E. R.; BARROS, R. C.; HRUSCHKA, E. R.; CARVALHO, A. C. P. L. F. D.; GAMA, J. A. Data stream clustering: A survey. *ACM Comput. Surv.*, vol. 46, no. 1, pp. 13:1–13:31, 2013. (Not cited.)
- SILVA, J. A.; HRUSCHKA, E. R.; GAMA, J. An evolutionary algorithm for clustering data streams with a variable number of clusters. In: *Evolutionary Computation*, p. to be appeared, 2014. (Not cited.)
- STEINBACH, M.; KARYPIS, G.; KUMAR, V. A comparison of document clustering techniques. In: *Proc. of the KDD Workshop on Text Mining*, pp. 109–111, 2000. (Not cited.)
- VENDRAMIN, L.; CAMPELLO, R. J. G. B.; HRUSCHKA, E. R. Relative clustering validity criteria: A comparative overview. *Statistical Analysis and Data Mining*, vol. 3, pp. 209–235, 2010. (Not cited.)
- VIVEKANANDAN, P.; NEDUNCHEZHIAN, R. Mining data streams with concept drifts using genetic algorithm. *Artificial Intelligence Review*, pp. 1–16, 2011. (Not cited.)
- WU, X.; KUMAR, V.; QUINLAN, J. R.; GHOSH, J.; YANG, Q.; MOTODA, H.; McLACHLAN, G. J.; NG, A.; LIU, B.; YU, P. S.; ZHOU, Z.-H.; STEINBACH, M.; HAND, D. J.; STEINBERG, D. Top 10 algorithms in data mining. *Knowledge and Information Systems*, vol. 14, pp. 1–37, 2007. (Not cited.)
- XU, R.; WUNSCH, D. *Clustering*. IEEE Press Series on Computational Intelligence. Wiley-IEEE Press, 2009. (Not cited.)
- ZHANG, T.; RAMAKRISHNAN, R.; LIVNY, M. Birch: An efficient data clustering method for very large databases. *SIGMOD Record*, vol. 25, no. 2, pp. 103–114, 1996. (Not cited.)

ZHANG, T.; RAMAKRISHNAN, R.; LIVNY, M. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 141–182, 1997. (Not cited.)

ZHOU, A.; CAO, F.; QIAN, W.; JIN, C. Tracking clusters in evolving data streams over sliding windows. *Knowledge and Information Systems*, pp. 181–214, 2008. (Not cited.)

ZHU, Y.; SHASHA, D. StatStream: statistical monitoring of thousands of data streams in real time. In: *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB Endowment, pp. 358–369, 2002. (Not cited.)

Apêndice

Neste Capítulo encontra-se o artigo sobre o protótipo de um sistema para auxiliar no monitoramento dos grupos de dados que mudam com o decorrer do FCD. Este artigo foi aceito em 2014 pela *ACM Transactions on Autonomous and Adaptive Systems*

A Support System for Clustering Data Streams with a Variable Number of Clusters

Jonathan de Andrade Silva, University of São Paulo (USP)
 Eduardo Raul Hruschka, University of São Paulo (USP)

Many algorithms for clustering data streams that are based on the widely used k -Means have been proposed in the literature. Most of these algorithms assume that the number of clusters, k , is known and fixed *a priori* by the user. Aimed at relaxing this assumption, which is often unrealistic in practical applications, we propose a support system that allows not only estimating the number of clusters automatically from data but also monitoring the process of the data stream clustering. We illustrate the potential of the proposed system by means of a prototype that implements eight algorithms for clustering data streams, namely, Stream LSearch-OMR k , Stream LSearch-BkM, Stream LSearch-IOMR k , Stream LSearch-IBkM, CluStream-OMR k , CluStream-BkM, StreamKM++-OMR k , and StreamKM++-BkM. These algorithms are combinations of three state-of-the-art algorithms for clustering data streams with fixed k , namely, Stream LSearch, CluStream, and StreamKM++, with two algorithms for estimating the number of clusters, which are Ordered Multiple Runs of k -Means (OMR k) and Bisecting k -Means (BkM). We experimentally compare the performance of these algorithms using both synthetic and real-world data streams. Analyses of statistical significance suggest that the algorithms that are based on OMR k yield the best data partitions, while the algorithms that are based on BkM are more computationally efficient. Additionally, StreamKM++-OMR k and Stream LSearch-IBkM provide the best trade-off relationship between accuracy and efficiency.

Categories and Subject Descriptors: I.5.3 [Pattern Recognition]: Clustering

General Terms: Design, Algorithms, Experimentation

Additional Key Words and Phrases: Clustering, Data Stream, Online Clustering

ACM Reference Format:

Jonathan A. Silva and Eduardo R. Hruschka, 2013. A Support System for Clustering Data Streams with a Variable Number of Clusters. *ACM Trans. Autonom. Adapt. Syst.* x, x, Article x (March 2013), 26 pages.
 DOI: <http://dx.doi.org/10.1145/0000000.0000000>

1. INTRODUCTION

Advances in both hardware and software have enabled large-scale data acquisition. Currently, very large amounts of data are being collected in dynamic environments, at fast speeds. Such data are usually referred to as *data streams*. A data stream is an unbounded, ordered sequence of objects that must be accessed in order and that can be read only once or a small number of times [Guha et al. 2003]. The data stream problem has been extensively studied in recent years because of relevant applications, e.g., see [Yu et al. 2008; Gama 2010; Mouchaweh 2010; Lughofer 2011; Sayed Mouchaweh and Lughofer 2012; Fisch et al. 2012].

The authors would like to thank the Research Agencies CAPES, CNPq, and FAPESP grant #2010/15049-7 for their financial support of this work.

Author's addresses: J. A. Silva and E. R. Hruschka, Computer Science Department, University of São Paulo (USP) at São Carlos, SP, Brazil.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2013 ACM 1556-4665/2013/03-ARTx \$15.00
 DOI: <http://dx.doi.org/10.1145/0000000.0000000>

Data streams must be intelligently transformed into meaningful and actionable information, which can then be used to enable more effective decision-making. To accomplish such a goal, machine learning algorithms that are capable of continuous learning over time play a pivotal role. Specifically, data streams require learning algorithms that can evolve models, eventually being able to forget the (portion of) the data that becomes obsolete. In this context, incremental algorithms are of great relevance to avoid the computationally intensive task of re-training the whole model while accounting for dynamic patterns in the data that change over time. Additionally, the data stream must be processed in a single-pass-like manner, i.e., the data stream cannot be revisited over the course of the computation. Usually, the data objects are discarded after being processed.

A useful form of analyzing data streams involves clustering [Aggarwal et al. 2004; Gama 2010; Bouchachia 2011; Quiroz et al. 2012; Silva et al. 2013]. The literature on clustering is very large. Among the many algorithms that are available is k -Means, which is very popular for data mining due to its simplicity, scalability, and empirical success in many real-world applications [Wu et al. 2007; Jain 2009]. Not surprisingly, a number of k -Means variants have been developed for addressing data streams, e.g., see [Guha et al. 2003; O'Callaghan et al. 2002; Aggarwal et al. 2003; 2004; Ackermann et al. 2010]. Despite the successful application of these algorithms to many real-world problems, they have a major limitation: the number of clusters, k , must be specified in advance by the user. A few algorithms are able to estimate the number of clusters [Beringer and Hüllermeier 2006; de Andrade Silva and Hruschka 2011; Lughofer 2012]. In [Beringer and Hüllermeier 2006; Lughofer 2012] split-and-merge strategies are proposed to estimate the number of clusters by increasing or decreasing the value of k according to threshold values. An algorithmic framework that also allows estimating k automatically from the data has been introduced in [de Andrade Silva and Hruschka 2011]. This framework was designed to adapt some traditional data stream clustering algorithms to estimate the k value. We use this framework as a starting point in designing a support system for clustering data streams with a variable number of clusters. We also propose an incremental variant of this framework that reduces the processing time. From a more practical viewpoint, we note that MOA (Massive Online Analysis) [Bifet et al. 2010] is the framework that is most closely related to the prototype that we have implemented. More specifically, MOA is an open-source framework for addressing massive evolving data streams, and it is related to the popular WEKA (Waikato Environment for Knowledge Analysis) system. In this context, our work is aimed at extending this framework toward allowing an automatic estimation of the number of clusters, which is a desirable property that is not supported by MOA.

We illustrate the potential of the proposed system by means of a prototype that implements eight algorithms for clustering data streams [de Andrade Silva and Hruschka 2011], namely, Stream LSearch-OMR k , Stream LSearch-BkM, Stream LSearch-IOMR k , Stream LSearch-IBkM, CluStream-OMR k , CluStream-BkM, StreamKM++-OMR k and StreamKM++-BkM. In short, these algorithms are combinations of three state-of-the-art algorithms for clustering data streams with fixed k , namely, Stream LSearch [O'Callaghan et al. 2002], CluStream [Aggarwal et al. 2003], and StreamKM++ [Ackermann et al. 2010], with two well-known algorithms for estimating the number of clusters, which are Ordered Multiple Runs of k -Means (OMR k) and Bisecting k -Means (BkM). Such combinations have been designed to retain the desirable properties of the data stream algorithms while additionally allowing them to automatically estimate the number of clusters from data, as briefly introduced in [de Andrade Silva and Hruschka 2011]. Because these algorithms are based on the widely used k -Means [Wu et al. 2007], they can be helpful for a large audience, e.g., k -

Means users and data mining practitioners who are potentially interested in this type of algorithm. From this perspective, the main contributions of our work are as follows:

- We propose a support system for clustering data streams that have a variable number of clusters. The implemented prototype shows that such a system allows not only estimating the number of clusters automatically from data but also monitoring the process of the data stream clustering;
- We propose an incremental framework that speeds up the estimation of the number of clusters. Such a prototype is used to extend the Stream LSearch to perform the k estimation step incrementally and additionally can detect the eventual deterioration of the data partition quality. The proposed incremental framework can also be used to extend the Clustream and StreamKM++ algorithms;
- We illustrate how the proposed system can support a variety of data inputs, data visualization and projection techniques, data stream algorithms, and clustering validity measures;
- We experimentally evaluate the performance of the proposed system on both synthetic and real-world data streams.

The remainder of this paper is organized as follows. In Section 2, three state-of-the-art algorithms for clustering data streams that are used as components of the proposed support system are briefly described. Section 3 presents a framework adopted to estimate the number of clusters. In Section 5, the proposed prototype system is presented in its full form. Illustrative experimental results are reported in Section 6. Finally, Section 7 concludes the paper.

2. K-MEANS-BASED ALGORITHMS FOR DATA STREAMS

2.1. Stream LSearch

The problem of clustering data streams is formulated in [O'Callaghan et al. 2002] as the problem of finding a solution for the k -Median optimization problem. In brief, the objective function essentially involves minimizing the sum of squared distances of the data points to their assigned medians. Such an optimization problem is related to the k -Medoids problem, except for the fact that one does not have to know the exact value of k a priori [O'Callaghan et al. 2002] and that a cost is imposed for each medoid. However, this arrangement does not mean that the user no longer needs to provide an initial estimate of k before running the algorithm. Actually, the number of clusters, k , is a (user-defined) parameter of the algorithm, which searches for the best data partition by considering a number of clusters in $[k, 2k]$. Ackermann et al. [2010] observe that Stream LSearch does not always find the pre-specified k and that usually the difference lies within a 20% margin from the value of k chosen in advance. The authors in [O'Callaghan et al. 2002] observe that the optimization problem tackled by Stream LSearch is NP-hard. Therefore, approximation algorithms are usually employed. Aggarwal et al. [2003] note that Stream LSearch is implemented as a continuous version of k -Means and assumes that the clusters are to be induced over the entire data stream. More precisely, the streaming algorithm assumes that the data arrive in chunks X_1, \dots, X_T , where each chunk $X_i, i \in [1, T]$ can be clustered in main memory. Stream LSearch is performed for each X_i . At the i th chunk of the stream the algorithm retains $O(i \cdot k)$ data points. For long streams ($T \rightarrow \infty$), retaining $O(i \cdot k)$ points could become prohibitively large. Aimed at avoiding this problem, Stream LSearch clusters the $i \cdot k$ centers and retains only the resulting $2k$ centers, as illustrated in Fig. 1. In particular, note that each chunk, X_i , which has m objects, will be clustered into $2k$ centers. As a final comment, Stream LSearch is limited in that it does not account for

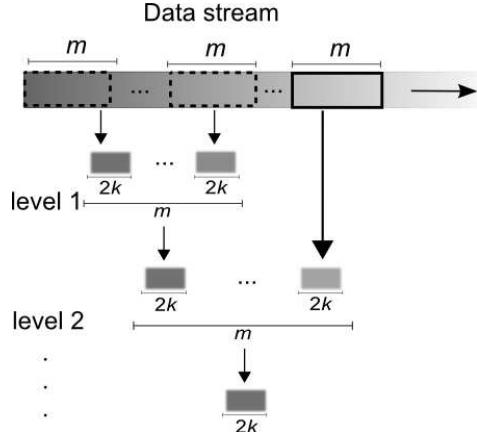


Fig. 1. Overview of the streaming algorithm used by Stream LSearch [O'Callaghan et al. 2002].

the data evolution, in the sense that the obtained clusters from each chunk are given the same importance over the stream.

2.2. CluStream

This algorithm considers that the data evolves over time and allows exploring clusters over different portions of the stream. The clustering process is divided into two components: (i) an online micro-clustering component that periodically stores summary statistics and (ii) an offline macro-clustering component that uses the summary statistics, in conjunction with users' inputs, to provide the clusters. CluStream makes use of a temporal variant of BIRCH's Cluster Feature [Zhang et al. 1996] to maintain the summary statistics, which are essentially the sums of the data values (and the time stamps) and their squares. Initially, CluStream runs k -Means on a predefined number ($Init$) of objects, thus finding q micro-clusters. As noted in [Aggarwal et al. 2003], the value of q is determined by the amount of main memory that is available for storing the micro-clusters. Therefore, typical values of q are larger than the natural number of clusters in the data and are smaller than the number of objects that arrive over a long period of time. Then, for each new object that arrives, the algorithm updates the micro-clusters in order to reflect the changes.

Consider that each object is described by means of an l -dimensional feature vector $\mathbf{x}_v = [x_j^v]_{j=1}^l$, $1 \leq v < \infty$. Thus, once CluStream runs k -Means on a predefined number ($Init$) of objects, an object \mathbf{x}_{Init+v} is inserted either into an existent micro-cluster or into a new micro-cluster, i.e., it \mathbf{x}_{Init+v} corresponds to either an outlier or to the seed of a new cluster (in this case because of the evolution of the data stream). To illustrate how CluStream works, Fig. 2(a) presents objects as circles, micro-cluster centers as squares, and a new object \mathbf{x}_{Init+v} . This new object is inserted into an existent micro-cluster iff its distance to the closest micro-cluster falls within a maximum boundary of the micro-cluster. Otherwise, a new micro-cluster, which initially contains only \mathbf{x}_{Init+v} , is created — see Fig. 2(a), where the distance from the object to the closest micro-cluster, which is 0.025, is greater than the radius of this micro-cluster, which is 0.021. Thus, the number of (other) clusters is reduced by one when creating the memory space — see Fig. 2(b), where the two closest micro-clusters, whose distance from each other is 0.014, are merged to save space. In particular, either the oldest micro-cluster is removed or two micro-clusters are merged. The oldest micro-cluster is removed if its timestamp exceeds a given threshold δ (input parameter). Otherwise, the two closest micro-clusters

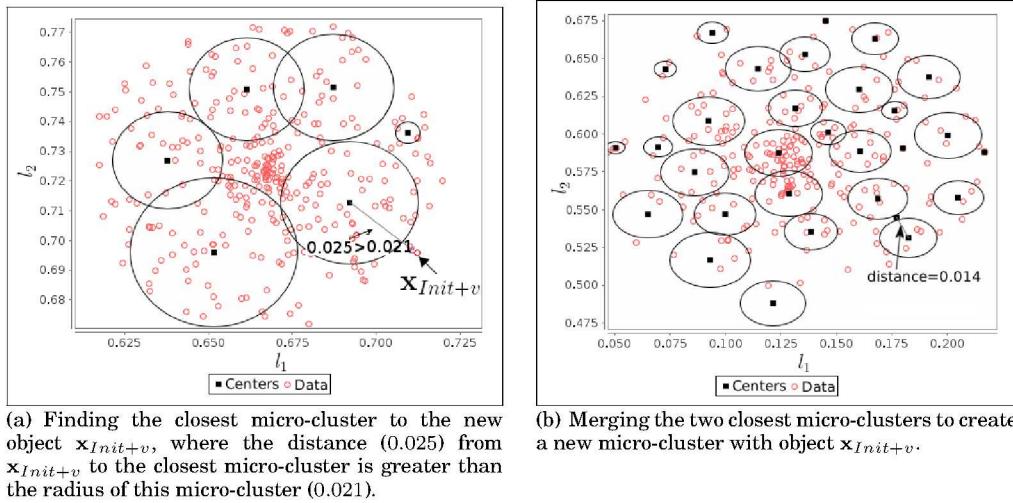


Fig. 2. Illustrative example of the CluStream algorithm.

are merged. The q micro-clusters are stored in a secondary storage device from time to time, the so-called *snapshots*. These snapshots are stored at different levels of granularity, which allows for a search for clusters at different time horizons, through a *pyramidal time window* concept [Aggarwal et al. 2003]. The macro-clustering component employs the compactly stored summary statistics of the micro-clusters in order to find k clusters (by using a variant of k -Means) for a given time horizon, h . The user supplies (among other parameters) both k and h as inputs.

2.3. StreamKM++

This algorithm is based on the k -Means++ algorithm, which can be viewed as a seeding procedure for the traditional k -Means (Lloyd's) algorithm and which guarantees a data partition with a certain quality [Arthur and Vassilvitskii 2007]. The streaming algorithm maintains a small summary of the input objects of the stream by using a merge-and-reduce technique [Ackermann et al. 2010]. To do so, the algorithm maintains a number of buckets, which can store m objects, where m is a parameter of the algorithm. The main idea is to organize a small number of samples in such a way that a bucket B_i contains m objects that represent $2^i m$ input objects. Whenever two samples that represent the same number of inputs exist, they are merged into $2m$ objects, and a new sample is created by means of the *reduce step*.

Fig. 3 illustrates this procedure by considering three buckets. The objects are always inserted into the first bucket (B_1), which is represented as a dashed border. Initially, some objects (which are represented as large circles at time $T = 1$) are inserted into bucket B_1 only. Once this bucket is full, their objects are moved to the neighboring bucket (B_2) when it is empty, as illustrated at time $T = 1$. However, if the buckets B_1 and B_2 are full, then the objects of these two buckets are merged, and the result is kept in the third bucket (B_3), as illustrated in Fig. 3 at $T = 2$, thus, making room for the new objects in the buckets B_1 and B_2 (unless the third bucket is also full, in which case the process is repeated as illustrated in Fig. 3 when inserting the objects that are represented by “•” symbols for $T = 4$).

For the *reduce step*, the authors proposed a data structure called the coresset tree. A coresset for a set of points is a small (weighted) point set that approximates the points

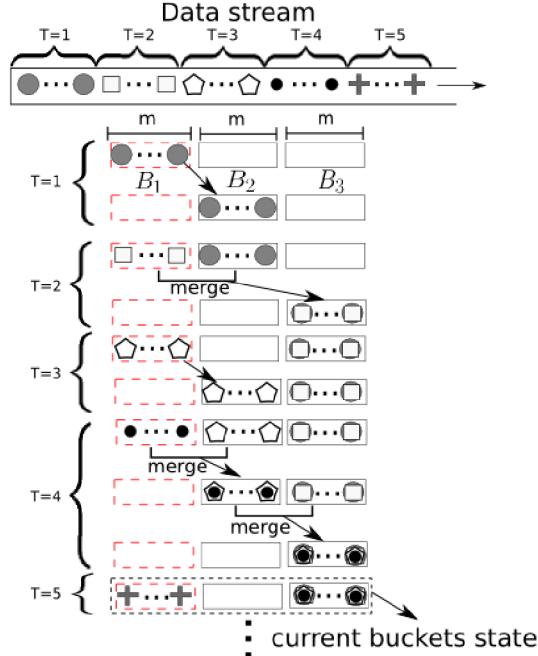


Fig. 3. Overview of the streaming algorithm used by StreamKM++ [Ackermann et al. 2010].

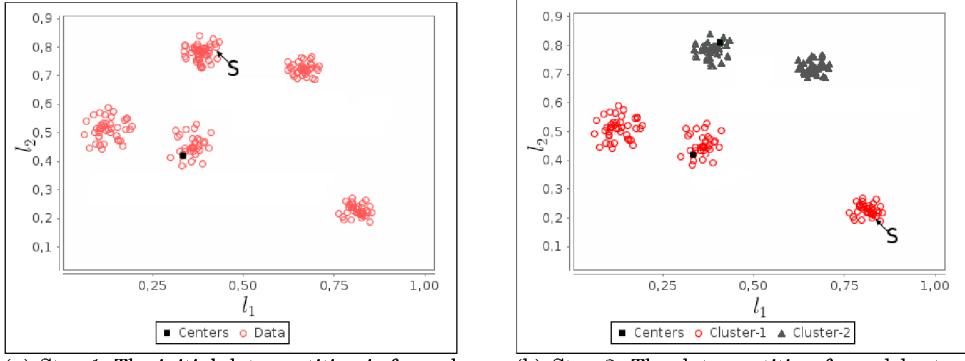
from the stream with respect to the k -Means clustering problem. More precisely, given a set of points, \mathcal{P} , for any set of k cluster centers, the (weighted) clustering cost of the coresset is an approximation for the clustering cost of \mathcal{P} with a small relative error. Fig. 4 illustrates an example of the reduce step, where the algorithm recursively partitions the data ($2m$ objects) into two clusters until m clusters have been found. Afterward, the clusters can be obtained at any time by running k -Means++ on the coresset, by applying the reduce step for all of the buckets. Note that k is still defined by the user.

3. ESTIMATING THE NUMBER OF CLUSTERS

One of the most difficult problems in data clustering is to determine the appropriate number of clusters [Jain 2009]. Determining an appropriate number of clusters is usually a procedure that is based on two main steps. First, the data are partitioned with different values for k . Second, quality measures, the so-called relative validity indexes [Jain and Dubes 1988], are used to assess the appropriateness of the obtained partitions. The relative clustering validity criteria that are non-monotonic with the number of clusters can be potentially used as an objective function for an algorithm that is designed to optimize k . Among the alternatives that are available in the literature, we have chosen the simplified version of the Silhouette Width Criterion [Kaufman and Rousseeuw 1990] as the objective function. Such an index has scored among the best in a comprehensive comparative study [Vendramin et al. 2010].

3.1. Simplified Silhouette (SS)

To explain this relative validity index, let us consider a data set X that is composed of N objects that are described by means of l -dimensional feature vectors $\mathbf{x}_i = [x_j^i]_{j=1}^l$, $1 \leq i \leq N$ and a hard partition $C = \{C_1, \dots, C_k\}$ of data into k non-overlapping clusters, such that $\bigcup_{f=1}^k C_f = X$, $C_f \neq \emptyset$ and $C_h \cap C_f, h \neq f$ ($h, f \in \{1, \dots, k\}$) and $C_h \neq \emptyset$.



(a) Step 1: The initial data partition is formed by a single cluster. Then, an object (S) is probabilistically selected to be the next (initial) cluster center.

(b) Step 2: The data partition formed by two clusters (triangles and circles) whose individual SSQ costs are 2.831 and 12.541, respectively. The cluster with a higher cost (circles) is chosen to be split. Again, a new object, which is represented by S, is chosen to be the initial center of the cluster.

Fig. 4. Illustrative example of the reduce step used by StreamKM++ [Ackermann et al. 2010].

First, let us introduce the silhouette that is proposed in [Kaufman and Rousseeuw 1990]. Consider an object x_i that belongs to cluster C_a . The average dissimilarity¹ of x_i to all of the other objects in C_a is denoted by $a(x_i)$. Next, let us consider cluster C_b . The average dissimilarity of x_i to all of the objects in C_b will be called $d(x_i, C_b)$. For all clusters $C_b \neq C_a$, the closest cluster is selected, i.e., $b(x_i) = \min d(x_i, C_b)$, $C_b \neq C_a$. This value represents the dissimilarity of x_i to its closest neighboring cluster, and the silhouette $s(x_i)$ is given by:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max\{a(x_i), b(x_i)\}} \quad (1)$$

Note that $0 \leq s(x_i) \leq 1$ if the objects are assigned to the closest cluster. Thus, the higher the value of $s(x_i)$ is, the better the assignment of object x_i to a given cluster. In addition, if $s(x_i) = 0$, then it is not clear whether the object should have been assigned to its current cluster or to a neighboring cluster [Everitt et al. 2001]. Finally, if cluster C_a is a singleton, then $s(x_i)$ is not defined, and the most neutral choice is to set $s(x_i) = 0$ [Kaufman and Rousseeuw 1990].

The silhouette proposed in [Kaufman and Rousseeuw 1990] depends on the computation of all of the distances between pairs of objects, which leads to a computational cost of $O(N^2)$; this cost is often considered to be computationally inefficient for streaming applications. To circumvent this limitation, a simplified silhouette [Hruschka et al. 2006] can be used. The SS is based on the computation of the distances between the objects and cluster centroids, which are the mean vectors of the clusters. More specifically, the term $a(x_i)$ of Equation 1 becomes the dissimilarity of object x_i to its corresponding cluster (C_a) centroid. Similarly, instead of computing $d(x_i, C_b)$ as the average dissimilarity of x_i to all of the objects of C_b , $C_b \neq C_a$, the distances between x_i and the centroid of C_b are computed. Thus, for k -Means based algorithms the silhouette can be

¹Dissimilarities are usually computed from distance measures, such as the well-known Euclidean distance.

computed as:

$$s(\mathbf{x}_i) = 1 - \frac{a(\mathbf{x}_i)}{b(\mathbf{x}_i)} \quad (2)$$

While these modifications reduce the estimated computational cost from $O(N^2)$ to $O(N)$, empirical results ([Hruschka et al. 2004; Hruschka et al. 2006]) suggest that the quality of the achieved partitions might be not significantly affected. The average of $s(\mathbf{x}_i)$ over $i = 1, \dots, N$, i.e.,

$$\text{SS} = \frac{1}{N} \sum_{i=1}^N s(\mathbf{x}_i) \quad (3)$$

can be used as a criterion to assess the quality of a given partition. By using this criterion, the best clustering is achieved when the SS value is maximized.

3.2. Algorithms to Generate Data Partitions

3.2.1. Ordered Multiple Runs of k -Means (OMR k). In practice, a common approach for estimating k constitutes obtaining different data partitions (possibly with different k values) and then assessing each obtained partition with a relative validity index [Jain and Dubes 1988] in order to estimate the best value for k [Jain 2009]. This approach is essentially the procedure that is performed by OMR k [Naldi et al. 2011], which executes k -Means repeatedly for an increasing number of clusters. For each value of k , a number of partitions achieved by k -Means (from different initializations) is assessed by using a relative validity index (e.g., the Simplified Silhouette [Hruschka et al. 2006] used here) to estimate the best value of k (we denote this estimated value as \hat{k}) and its corresponding data partition. Then, after running k -Means for every k in $\{k_{\min}, \dots, k_{\max}\}$, the best obtained partition (w.r.t. the validity index) is selected. In our experiments, following a commonly adopted rule of thumb [Pal and Bezdek 1995], the values of k_{\min} and k_{\max} have been set to two and \sqrt{N} , respectively, where N is the number of objects to be clustered.

3.2.2. Bisecting k -Means (BkM). The BkM algorithm [Steinbach et al. 2000] is a (divisive) hierarchical variant of k -Means that recursively partitions the data into two clusters at each step, until the desired number of clusters has been found. It considers that initially there is a single cluster that is formed by all of the dataset objects. Such a cluster is split into two sub-clusters. From now on, BkM recursively chooses a cluster to be split. There are several ways to perform this division. We have chosen the largest cluster criterion, where the size of the cluster is measured as two times the distance of the centroid to the farthest object that belongs to that cluster. To automatically estimate k from the data, we use the approach that is described in [Naldi et al. 2009], where BkM estimates \hat{k} from the set $\{2, \dots, k_{\max}\}$ by using the Simplified Silhouette criterion [Hruschka et al. 2006]. With respect to OMR k , we have adopted $k_{\max} = \sqrt{N}$.

3.3. A Framework for k Estimation

The algorithms addressed in Section 2 can be summarized by means of two main steps, namely, a *streaming* step and a *clustering* step. Basically, the *streaming* step summarizes the data stream with the help of particular data structures for dealing with space and memory constraints that are typical in stream applications. The *clustering* step obtains a data partition from these summaries to provide a quick understanding of the clusters.

To estimate the number of clusters from the data, our framework uses an intermediate \hat{k} -step, as illustrated in Fig. 5. More precisely, the *streaming* algorithm creates

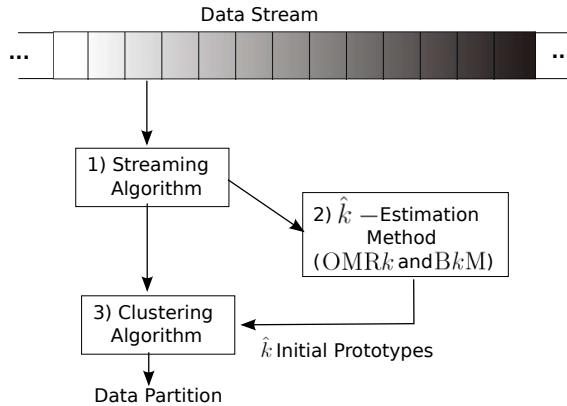


Fig. 5. Overview of the proposed framework.

data synopses from the data stream. Such synopses can be kept in data structures that are smaller than the whole data stream [Han and Kamber 2000]. After building a data synopsis, the number of clusters is estimated via either OMR k or B k M. Then, not only the estimated number of clusters but also the obtained cluster centers serve as inputs to the *clustering step*. Further details on the *instantiation* of this framework are provided below.

Fig. 7 illustrates a running example of the adopted framework. Let us again consider that the data stream arrives in chunks (X_1, \dots, X_T). In our framework, we process one chunk of data X_i at a time. Note in our running example that, as previously explained in Section 2, the algorithms for clustering data streams create data synopses from the data chunk. From such synopses, the \hat{k} -Estimation step automatically estimates the number of clusters and then sends both \hat{k} and the obtained prototypes for the clustering step, which depends on the specific clustering algorithm (recall that we can use three different algorithms, as shown in Fig. 7). For example, by using CluStream, k -Means now finds \hat{k} clusters from the summary, i.e., from the micro-clusters obtained in the streaming step. Note also that, by estimating the number of clusters, we can reduce the interaction with the user, which is not required to constantly provide the number of clusters anymore. The adopted framework allows the user to focus on the changes in the number of clusters, \hat{k} , if so desired. Additionally, (s)he could still monitor the clusters themselves, although such monitoring is usually a tedious and error-prone task.

3.3.1. Stream LSearch. The data synopsis is built from a *divide and conquer* strategy that splits the data stream into chunks. More precisely, let us consider that the data arrives in chunks X_1, \dots, X_T . For a specific chunk, the number of clusters is estimated via either OMR k or B k M. The clustering step is then performed by the LSearch algorithm, which refines the cluster centers that are obtained from the \hat{k} -step.

3.3.2. CluStream. The data synopsis is built from the on-line component, which keeps a set of micro-clusters. The off-line component, in its turn, runs k -Means to refine the data centers of the micro-clusters found in the \hat{k} -step.

3.3.3. StreamKM++. A set of coresets (data synopses) is used by the streaming algorithm to keep a compact representation of the processed objects from the data stream. The k -Means++ algorithm is run on the coressets, with the number of clusters estimated

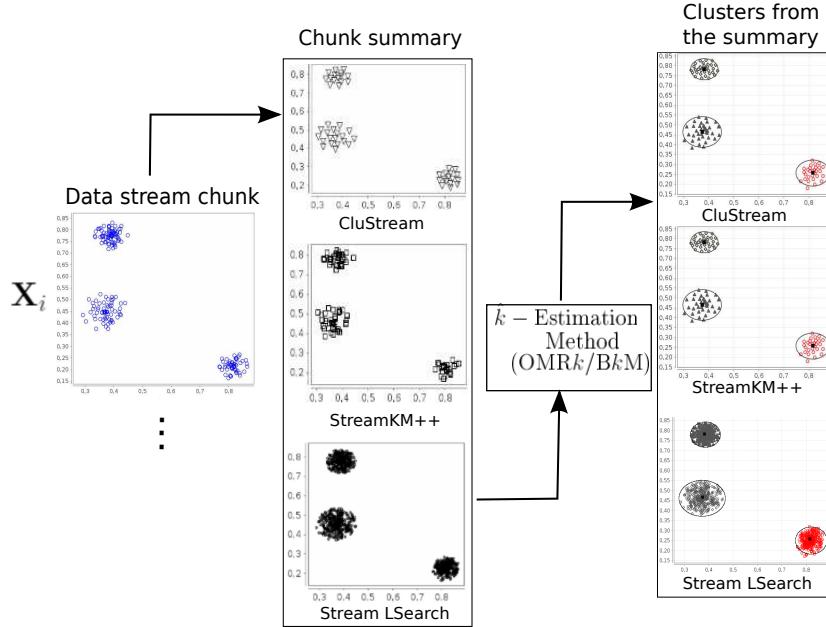


Fig. 6. Running example of the framework that is adopted to estimate the number of clusters from the data.

from the \hat{k} -step. The prototypes obtained from this step are not used, however, because k -Means++ has its own procedure to initialize the \hat{k} initial seeds.

4. INCREMENTAL FRAMEWORK FOR K ESTIMATION

4.1. Incremental Simplified Silhouette

In data stream scenarios, ideally we should be able to update the data partition in an online fashion. This alternative can save computational resources when the clusters do not change significantly over a certain time window. Note that to update the data partition in an online fashion, we also must incrementally evaluate the decisions that are taken during the update process. In this case, we want to estimate the impact of inserting an object that arrives from the data stream into the closest cluster. Specifically, we want to detect whether the (online) assignment of an object to the closest cluster makes the overall silhouette worse.

Let us assume that initially a data partition for a predefined $Init$ amount of data is available. Additionally, for each cluster C_c , $1 \leq c \leq k$, we store the minimum silhouette value (ss_c) by considering all of the objects that belong to that specific cluster. According to Equation 2, the minimum silhouette value corresponds to the maximum values for the terms $a(x_i)$ and $b(x_i)$, which usually correspond to objects in cluster boundaries. The ss_c value can be used as a threshold to determine whether the arriving objects should be inserted into the closest cluster C_c or whether a new cluster is emerging. In this determination, for each new object x_{Init+v} , we compute $s(x_{Init+v})$, and then, we compare this value with the ss_c of the corresponding closest cluster C_c . Specifically, if $s(x_{Init+v}) > ss_c$, then x_{Init+v} is inserted into cluster C_c . Otherwise, we run the k -Means algorithm with $k = |C|$ and $k = |C| + 1$ and choose the best data partition according to Equation 3. In Section 4.2, we provide more details on how to run k -Means for the Stream LSearch algorithm.

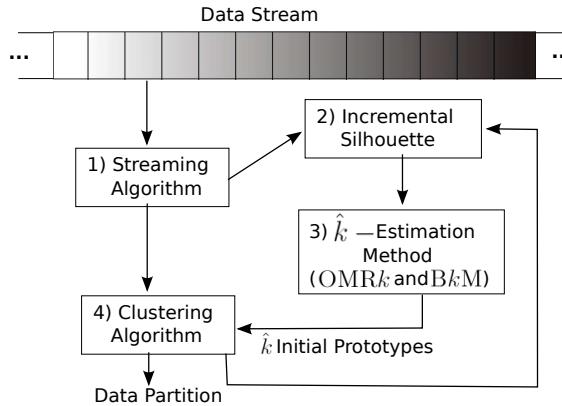


Fig. 7. Overview of the proposed incremental framework.

4.2. Incremental Adaptation Framework

In this section, we describe an incremental variant of the framework proposed in Section 3.3. Specifically, we describe the extension of the Stream LSearch for incremental \hat{k} -Estimation. We start by inserting this framework into the Stream LSearch algorithm because this algorithm has shown the highest processing costs in [de Andrade Silva and Hruschka 2011], and these can be used as upper bounds for the processing times. However, the incremental adaptation for CluStream and StreamKM++ can be done similarly to Stream LSearch by using the summaries instead of the objects in the data chunk. In particular, CluStream uses micro-clusters, whereas StreamKM++ uses core-set.

According to step 2 of Fig. 5, the number of clusters is estimated via either OMR k or B k M. The processing cost of this step can be reduced by introducing a method to incrementally update the clusters, such as the method described in Section 4.1. The resulting incremental framework is illustrated in Figure 7, where IOMR k and IB k M are the resulting algorithms from the combinations of steps 2 and 3, respectively. In the Stream LSearch algorithm, the streaming component receives data in the chunks X_1, \dots, X_T . For a specific chunk, the number of clusters is estimated via either IOMR k or IB k M. To estimate \hat{k} , step 2 stores the objects, which are arriving one by one, into a sliding window, which is essentially a *fifo*(first-in first-out) data structure. In this step, for each object v , we find the closest cluster and compute the silhouette $s(x_v)$ according to Equation 2. Then, we compare the silhouette value $s(x_v)$ with the minimum silhouette value of the closest cluster (ss_c). When $s(x_{Init+v}) < ss_c$, we run the k -Means algorithm on the objects of the sliding window, whose size is equal to the chunk size, with $k = |C|$ and $k = |C| + 1$, and we choose the best clustering according to Equation 3. Otherwise, we insert the object into the closest cluster. At the end of each chunk, we run step 3 if the number of clusters was changed (from k to $k + 1$) during step 2. This is an optional step that is used to reduce the impact of possible wrong decisions from the (on-line) creation of new clusters.

5. PROTOTYPE OF THE PROPOSED SUPPORT SYSTEM

The proposed system allows the data analyst to visually monitor the clustering process over the data stream. To illustrate the main features of the idealized system, we have implemented a prototype, whose characteristics are detailed here. The implemented prototype provides a set of algorithms for clustering data streams, as discussed in the

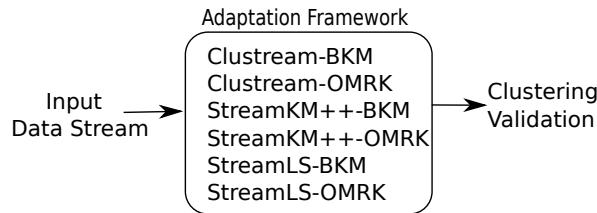


Fig. 8. Overview of the prototype architecture.

previous sections. One important feature of these algorithms is that they are capable of automatically estimating the number of clusters from the data, which is of great value as a decision-making support system for the data analyst.

Fig. 8 shows an overview of the prototype architecture. The input is a data stream source such as GPS data, sensor data, marketing data, sales data, and so on. The second component involves the algorithms described in Section 3.3. Finally, there is a component that provides mechanisms for the clustering validity. While the data stream is being processed, the support system provides visual information about the clusters that are found as well as information about the clustering validity. Altogether, these features allow the data analyst to suitably perform exploratory data analysis from the obtained data partitions, identifying changes over time.

5.1. Data Visualization

The proposed support system allows data visualization for user-defined subsets of attributes, as illustrated in this section by means of some screen shots. For example, Fig. 9 presents windows that show clusters for different pairs of attributes, given a specific data chunk that is formed by 1,000 objects synthetically generated from a four-dimensional Gaussian distribution with $k=5$.

At every time instant, 1,000 objects are clustered, e.g., as illustrated in Fig. 9 for different subsets of attributes. By inspecting those windows, the data analyst can infer the subsets of attributes that are the most discriminative, such as those in the second window (see Fig. 9(b)). Furthermore, one can analyze the clustering structure at any time. Aimed at further improving this desirable feature, more sophisticated forms of data visualization can be used, such as those based on automatic attribute selection [Covões and Hruschka 2011] and projection [Pearson 1901] (to name a few), in such a way that the obtained clustering becomes more interpretable to the data analyst.

5.2. Results Interpretation

In addition to data visualization, another useful form of interpreting the obtained results involves using clustering validity indexes [Jain 2009]. There are many different clustering validity indexes that are helpful in practice as quantitative criteria for evaluating the quality of data partitions (for a recent review, see [Vendramin et al. 2010]). In particular, because in practice the number of clusters is generally unknown, especially for l -dimensional data, where visual inspection is prohibitive for large l , a widely known and simple approach to get around this drawback constitutes obtaining a set of data partitions that have different numbers of clusters and then selecting the specific partition that provides the best result according to a specific quality criterion. Some well-known validity measures (also referred to as relative validity (or quality) criteria) are the Davies-Bouldin Index [Davies and Bouldin 1979], the Calinski-Harabasz Index [Calinski and Harabasz 1974], the Dunn Index [Dunn 1974], and the Silhouette Width Criterion of Kaufman and Rousseeuw [Kaufman and Rousseeuw 1990], to mention a few. In our prototype, we have implemented the Simplified Silhouette (SS)

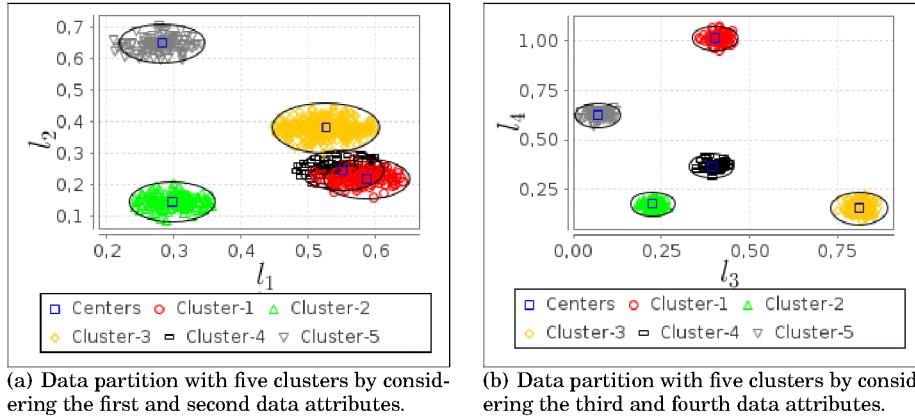


Fig. 9. Overview of a two-dimensional data partition with five clusters for a given chunk by considering: (a) the first and second data attributes and (b) the third and fourth data attributes.

[Hruschka et al. 2006] as a relative validity criterion to assess the quality of different partitions. This index has shown the most robust performance with respect to the different evaluation scenarios considered in [Vendramin et al. 2010], which reports an extensive experimental comparison of the performances of 40 clustering validity indexes. However, other relative validity indexes can be incorporated into the proposed system if so desired.

If the user is interested in a specific number of clusters, then internal indexes, such as the popular within-cluster Sum of Squared errors (SSQ), can also be used. For example, the data analyst could have some a priori knowledge about the number of clusters. Another plausible reason could be the presence of constraints on the maximum number of clusters, due to interpretability concerns.

Finally, if one is interested in assessing how well a given algorithm for clustering data streams performs on controlled experiments, then external indexes can be used. In this context, the obtained data partition can be evaluated with respect to an external (absolute rather than relative) criterion that, in a supervisory way, quantifies the degree of compatibility between the obtained partition and the correct partition, which is formed from known clusters. Among the most well-known external criteria are the Adjusted Rand Index (ARI) [Hubert and Arabie 1985] and the Jaccard coefficient [Kaufman and Rousseeuw 1990; Jain and Dubes 1988]. In addition to the obvious appeal that external indexes have for research purposes (e.g., the researcher might want to assess the performance of a new algorithm to be proposed), note that they can also be helpful in real-world applications, where a domain expert could provide a reference partition for a small subsample of the data, from which the data analyst is supposed to choose among a set of candidate algorithms to be employed in the complete sample.

To illustrate in a more concrete manner how the information about the clustering validity indexes can be helpful, we show in Fig. 10 the results of three indexes (SSQ, SS, and ARI) and the number of clusters computed since the beginning of the data stream (time=0) up to a certain point in time (time=62), when the data partition presented in Fig. 9 has been obtained. The proposed system allows monitoring all of these features concomitantly, from the observation of different windows. In our example, we assume that a reference partition is available. Thus, external indexes can be computed, but not necessarily (a reference partition is not always available). Figures 10(a) and 10(b) illustrate the obtained results, in terms of SSQ and SS, respectively, for both the

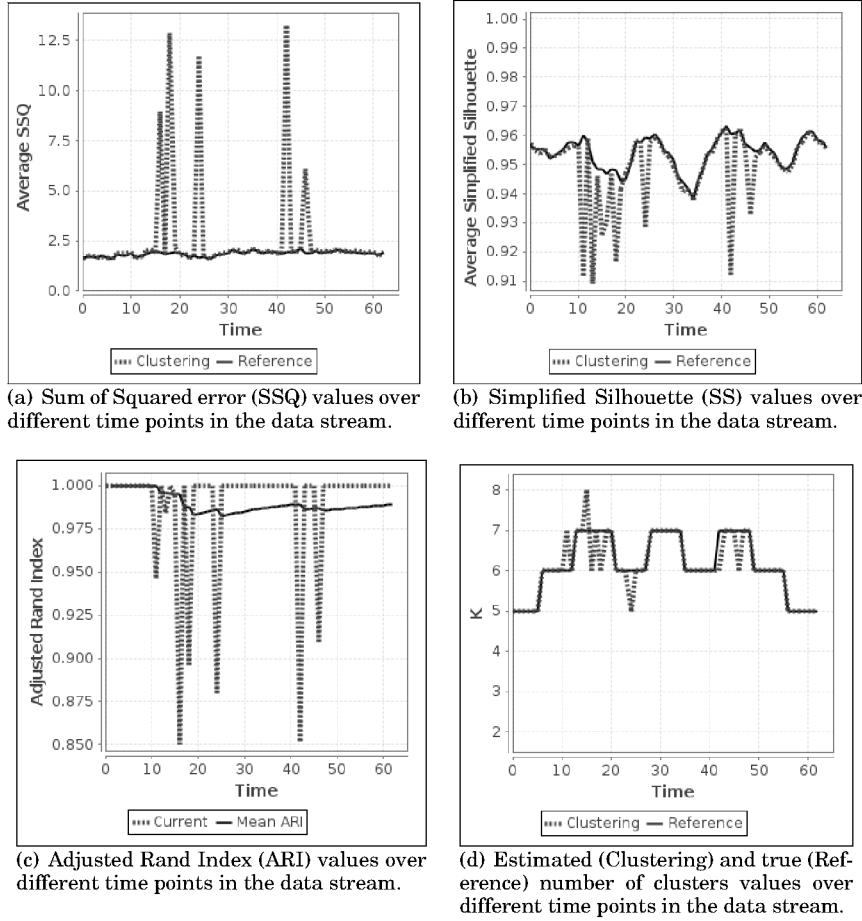


Fig. 10. Results of three clustering validity indexes (SSQ, ARI, and SS) and the number of clusters computed since the beginning of the data stream (time=0) up to a certain point (time=62).

induced data partition (dashed line) and the reference partition (solid line). The ARI results are also shown in Fig. 10(c), and the number of clusters can be seen in Fig. 10(d).

The information provided by means of the visualization of the clustering process as well as by the numerical values obtained from clustering validity indexes can be complemented by means of measures of the processing costs and memory use. From these measures, it is possible to visually monitor the used and available resources, thus making it possible to tune the parameters of the algorithms to obtain a better use of the system. For example, one might have information loss if the processing speed (number of objects per second) is significantly less than the stream speed. In this case, one can reduce the chunk size and, as a consequence, the number of objects being processed. On the other hand, this type of tuning could negatively impact the quality of the obtained data partitions. Aimed at attempting to avoid such an undesirable effect, visualization screens allow the user to observe the structure of the clusters and identify which ones have a degrading effect on the induced data partition. In practice,

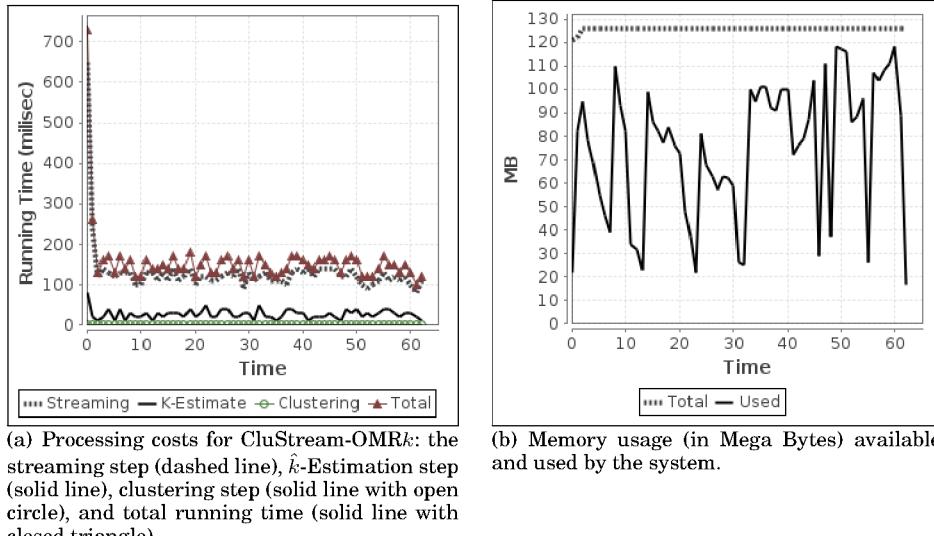


Fig. 11. Overview of the system results using CluStream-OMRk.

this feature of the system is helpful for the user, who usually wants to find a suitable trade-off between the memory usage and processing cost.

For the sake of illustration, Figure 11 presents three system screens that allow monitoring running the times (Fig. 11(a)) and memory usage (Fig. 11(b)), for CluStream-OMRk. More specifically, Fig. 11(a) shows partial and overall running times for each phase of the framework in Fig. 5, namely the streaming step, \hat{k} -Estimation step, clustering step, and total running time.

6. EXPERIMENTAL RESULTS

We empirically evaluated the proposed support system by using three algorithms for clustering data streams, namely, Stream LSearch [O'Callaghan et al. 2002], CluStream [Aggarwal et al. 2003], and StreamKM++ [Ackermann et al. 2010], which were combined with two algorithms for estimating the number of clusters, which are OMRk [Vendramin et al. 2010] and BkM [Steinbach et al. 2000]. Additionally, we proposed two incremental variants of Stream LSearch, namely, Stream LSearch-IOMRk and Stream LSearch IBkM (see Section 4.2). Thus, eight algorithmic instantiations were assessed: CluStream-OMRk (CLS-OMRk), Stream LSearch-OMRk (SLS-OMRk), Stream LSearch-IOMRk (SLS-IOMRk), StreamKM++-OMRk (SKM-OMRk), CluStream-BkM (CLS-BkM), Stream LSearch-BkM (SLS-BkM), Stream LSearch-IBkM (SLS-IBkM), and StreamKM++-BkM (SKM-BkM). Because of the random nature of these algorithms, we run them 10 times for each dataset. Following [Vendramin et al. 2010], OMRk was run by initializing 10 different partitions for every k value. The related parameter of BkM (number of iterations) was also set to 10, such that fair comparisons can be performed (as a complementary contribution of our work). For both algorithms, k -Means was programmed to stop when one of the following criteria are satisfied: (i) 5 iterations have been completed - empirical evidence suggests that this procedure usually produces satisfactory results [Anderberg 1973]; or (ii) the maximum absolute difference between centroids in two consecutive iterations is less than or equal to 10^{-3} .

Our experiments were conducted on a 2.20 GHz i7 quad-core PC (8GB memory) running the Ubuntu operating system. All of the algorithms were implemented in Java.

The synthetic datasets were processed in chunks formed by 1,000 objects ($N = 1000$). For each chunk, we computed the Adjusted Rand Index (ARI) values and the respective computational costs of the studied algorithms. Please note that, in the data stream settings, one must assume that if the buffer is full, more data are not sent until the previous chunk has been processed. Additionally, in practice, usually adjustments are made to the parameters of the algorithms in order to take advantage of the available computational resources.

6.1. Results on Synthetic Datasets

To evaluate the algorithms' ability to detect the evolution of clusters, we generated synthetic datasets from Gaussian distributions that had different dimensions $l = \{2, 4, 8, 16, 32, 64\}$. We randomly changed the number of clusters (Gaussians) from 2 to 8. In addition, their means and variances were randomly changed to reflect the evolution of the data stream over time. Each synthetic dataset has 200,000 objects. The employed data generator is an extension of *RBFGenerator*, which is available in the MOA software [Bifet et al. 2010], to generate hyper-spherical clusters from Gaussian Distributions. In fact, we generated hard data partitions, which are specially suitable for assessing the performance of several algorithms, such as Stream LSearch, Clu-stream, and StreamKM++. As in MOA software, our generator makes it possible to choose a number of parameter values, such as the maximum and minimum number of clusters, the cluster radius, number of attributes, and stream velocity, as well as making possible to control for the deletion and insertion of clusters.

The parameters of the stream clustering algorithms were set aimed at favoring fair comparisons among them. For CluStream, we set: $Init = 1,000$, $m = 50$, $\delta = 1,000$, and 100 micro-clusters, and the radius factor (a factor of the maximum boundary of a micro-cluster [Aggarwal et al. 2003]) was set to 2. For StreamKM++, the size of the coresset was set to 100. For Stream LSearch, because of the evolving nature of the data stream, the storage of the cluster prototypes obtained from the previous data chunks was not performed because out-of-date prototypes could dominate the overall stream behavior, thus possibly deteriorating the clustering quality.

6.1.1. Summary of the Results. Table I presents the average results obtained for the Adjusted Rand Index (ARI) and the processing time over 10 different initializations of the algorithms. Because the chunks are of size 1,000 and there are 200,000 objects in each dataset, we have 200 ARI and processing cost values (the average over 10 different initializations) for every pair of dataset-algorithm. Thus, we report the mean and the standard deviations computed over 200 values. One can observe that all of the studied algorithms provided very good ARI values and small standard deviations, which suggests that high-quality data partitions have been obtained. Additionally, SKM-BkM and SKM-OMR k provide the lowest average processing time of approximately 45.30 and 82.70 milliseconds, respectively to process the high-speed streams.

Considering the number of clusters, Fig. 12 illustrates the true values of k and the respective differences between them and the estimated values for each algorithm in a single run for a two-dimensional dataset. When the differences are zero, then it means that the algorithms found the correct values for k . Otherwise, the k value for the algorithms was superior (positive values) or inferior (negative values) to the groundtruth. We can observe that the BkM versions of algorithms show higher variance compared to the OMR k version of algorithms. Specifically, the SLS-OMR k , CLS-OMR k and CLS-BkM found the correct number of clusters in 89% of cases. The worst algorithms are the BkM versions of Stream LSearch, which yield to accuracies around 28%.

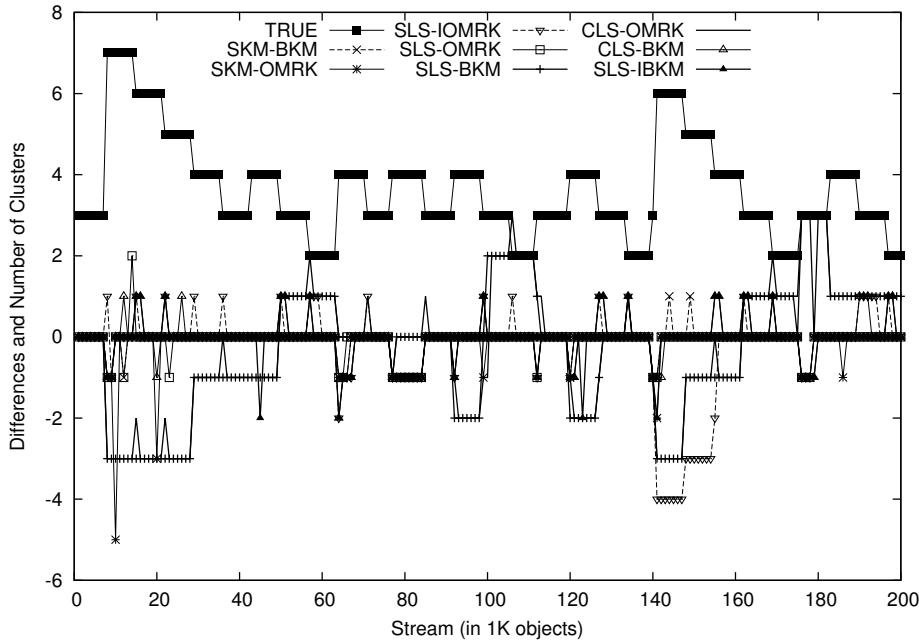


Fig. 12. Illustrative Example of the Evolution of Number of Clusters for a Synthetic Dataset.

Table I. ARI and Processing Time : Synthetic Data.

	ARI $\mu(\pm\sigma)$	Processing Time (milliseconds) $\mu(\pm\sigma)$
CLS-BkM	0.99 (± 0.02)	436.86 (± 53.17)
CLS-OMRk	0.99 (± 0.01)	503.28 (± 52.21)
SKM-BkM	0.98 (± 0.05)	45.30 (± 5.81)
SKM-OMRk	0.98 (± 0.03)	82.70 (± 5.69)
SLS-BkM	0.98 (± 0.03)	1,273.70 (± 596.07)
SLS-IBkM	0.97 (± 0.04)	1,359.62 (± 755.87)
SLS-OMRk	0.99 (± 0.01)	5,573.66 (± 466.03)
SLS-IOMRk	0.98 (± 0.04)	1,594.14 ($\pm 1,306.10$)

The average processing costs were computed for the ARI average values. Fig. 13 illustrates the overall mean processing time over the synthetic datasets. Note that SKM-BkM has a low processing time (approximately 45 milliseconds) and, thus, is eligible for high-speed streams. The performance of SKM-OMRk (approximately 100 milliseconds) can also be considered to be very good. CLS-OMRk and CLS-BkM have processing costs around 500 milliseconds. Because CluStream requires some time to create micro clusters, its processing cost is higher at the very beginning. Once the micro-clusters have been created, CluStream can process the remaining chunks at faster speeds. Finally, SLS-BkM and SLS-OMRk have shown the worse processing times. However, their incremental versions (SLS-IOMRk and SLS-IBkM) have shown important performance gains, especially for SLS-IOMRk, for which the gain is 72% w.r.t. SLS-OMRk. The SLS-BkM and SLS-IBkM have similar processing costs. This similarity arises because of the number of operations that are required to find the best data partition with k and $k + 1$ during the incremental update of the SLS-IBkM algo-

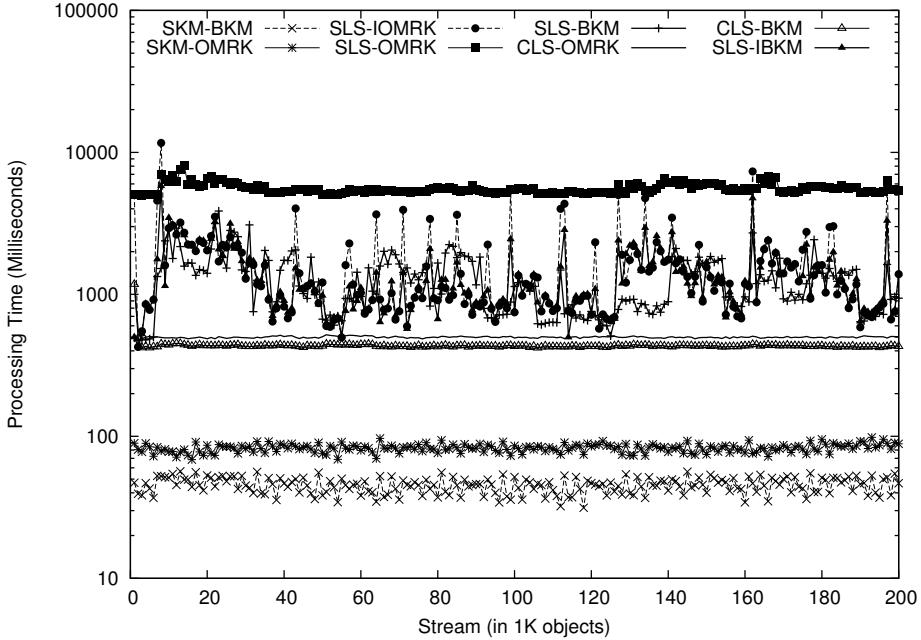


Fig. 13. Stream Processing Time for Synthetic Datasets.

rithm; thus, the total processing cost is the cost of BkM plus the number of executions of k -Means algorithms with k and $k + 1$ clusters (approximately 20 executions).

In most cases, BkM-based algorithms have shown better processing times compared to OMR k -based algorithms for two major reasons. First, the BkM's guided search procedure often finds data partitions with k^* less than \sqrt{N} . In particular, once BkM realizes that the quality of the data partition decreases for an increasing number of clusters, it stops inducing clusters, while OMR k systematically runs k -Means from two to \sqrt{N} clusters, thus possibly wasting computational resources when k^* is small (*i.e.*, closer to 2 than to \sqrt{N}). Second, for increasing values of k , BkM uses only those objects that belong to the selected cluster in order to run k -Means (with $k = 2$), while OMR k runs k -Means on the whole dataset.

To provide some reassurance about the validity and non-randomness of the obtained results, we present the results of statistical tests by following the approach proposed by Demšar [Demšar 2006]. In brief, this approach is aimed at comparing multiple algorithms on multiple datasets, and it is based on the use of the well-known Friedman test with a corresponding post-hoc test. The Friedman test is a non-parametric statistical test that is equivalent to the repeated-measures ANOVA. If the null hypothesis, which states that the algorithms under study have similar performances, is rejected, then we proceed with the Nemenyi post-hoc test [Hollander and Wolfe 1999] for pair-wise comparisons between the algorithms (with $\alpha = 5\%$). By considering the ARI values, there is no statistically significant difference between the performances of the algorithms.

Table II summarizes the obtained results for the pair-wise comparisons of the running time of the algorithms. The \odot symbol indicates that there is no statistically significant difference between the performance of the algorithms in row i and column j . Analogously, the $+$ symbol indicates that the results for the algorithm in i are significantly better than those for the algorithm in j . Finally, the $-$ symbol denotes that

Table II. Significant differences for processing times on synthetic data.

	SKM-B _k M	SKM-OMR _k	CLS-B _k M	CLS-OMR _k	SLS-B _k M	SLS-OMR _k	SLS-IB _k M	SLS-IOMR _k
SKM-B _k M	+		+	+	+	+	+	+
SKM-OMR _k	-		+	+	+	+	+	+
CLS-B _k M	-	-	+	+	+	+	+	+
CLS-OMR _k	-	-	-	+	+	+	+	+
SLS-B _k M	-	-	-	-	+	+	○	○
SLS-OMR _k	-	-	-	-	-	+	-	-
SLS-IB _k M	-	-	-	○	+	+	○	○
SLS-IOMR _k	-	-	-	○	+	○		

the results for the algorithm in row i are significantly worse than those achieved for the algorithm in column j . One can observe from Table II that SKM-OMR $_k$ and SKM-B $_k$ M provide significantly better results (w.r.t. the processing time) than the other algorithms.

6.1.2. Detailed Results. In this section, we present some detailed results that deserve a more careful consideration with respect to the summary provided in Section 6.1.1. Among the 400 different scenarios (8 algorithms \times 5 different quantities of attributes \times 10 trials) where the studied algorithms have been assessed, we report some experimental results for the datasets with 64 attributes by considering the OMR $_k$ and IOMR $_k$ algorithm only. These results were obtained from a single trial, thus simulating a real-world scenario, where averages over different trials cannot be computed.

Fig. 14 shows the running times for each component of the algorithms. For example, Fig. 14(a) shows the results for the streaming algorithm step, \hat{k} -Estimation step, clustering step, and total running time for the CluStream-OMR $_k$ algorithm. One can note from Fig. 14(a) that CluStream-OMR $_k$ has a high initial cost compared to the other time points. This finding is due to running k -Means on the first data chunk in order to find q micro clusters. Recall from Section 2.2 that subsequently the next objects from the data stream are either inserted into the closest available micro-clusters (this step is $O(q * l)$, where l is the number of attributes) or a new micro-cluster is created by considering two approaches: either (i) removing a micro-cluster whose average activity time is less than a user-defined threshold, which is $O(q)$, or (ii) merging the two closest micro-clusters, which is $O(q^2 * l)$. We can also observe from Fig. 14(a) that the micro-clustering component (also referred to as streaming clustering) cost is higher than the \hat{k} -Estimation step performed by OMR $_k$. This finding occurs because of the cluster merging operations, which are computationally demanding. To provide some intuition about this issue, Fig. 15 shows the number of object insertions into the closest micro-clusters, the number of operations for removing old micro-clusters, the number of merging operations, and the total number of operations. We observed that the processing of approximately 75% of the chunk objects required the creation of new clusters by the merging of existing micro-clusters.

Considering the StreamKM++ algorithm, Fig. 14(b) illustrates that the streaming step is less computationally costly than the \hat{k} -Estimation step. This finding arises because the frequency of insertions into the buckets is greater than the number of merging operations, which are more costly. In particular, merging operations are performed when neighboring buckets are full (see Section 2.3).

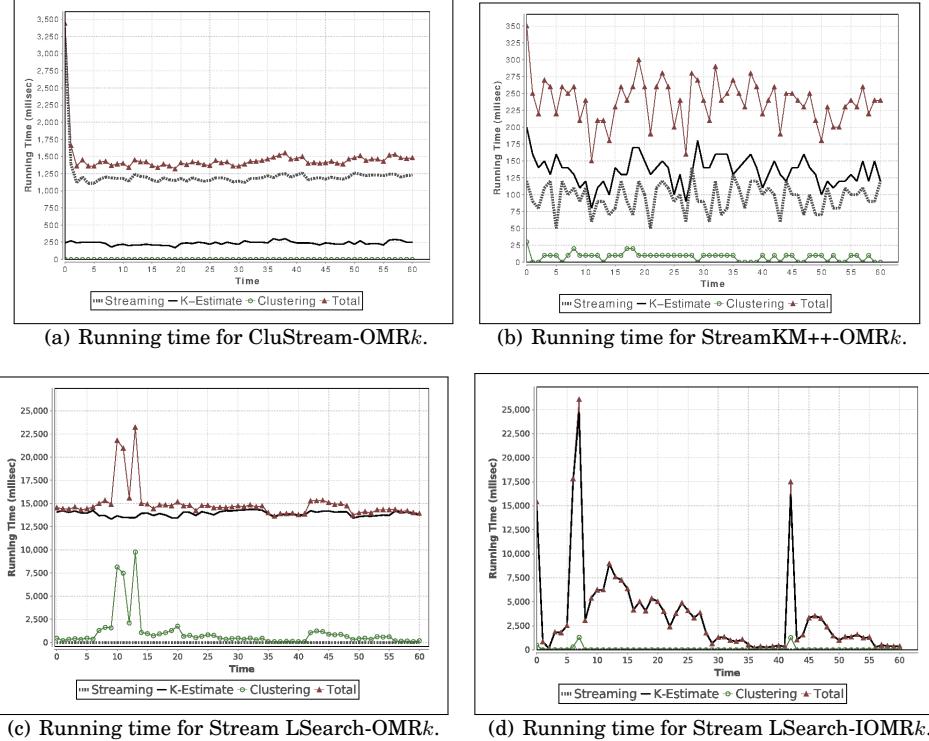


Fig. 14. The running time of CluStream-OMR k , StreamKM++-OMR k , Stream LSearch-OMR k , and Stream LSearch-IOMR k algorithms over time, using a 64-dimensional synthetic data set.

The results obtained for the Stream LSearch OMR k algorithm (Fig. 14(c)) show that the \hat{k} -Estimation step dominates the total computational cost. However, we can note that its incremental version (Fig. 14(d)) shows a reduction in the average processing rate of 13.056 to 3.543 milliseconds, which represents a performance gain of 73%. Fig. 16 shows not only that the number of clusters has been suitably estimated but also that the obtained data partitions are very good. In our experiments, StreamKM++-OMR k and StreamKM++-B k M have shown the best trade-off between the partition quality and the computational efficiency, especially when the number of attributes is reasonably high.

6.2. Results on the KDDCup'99 data set

We used the KDD-CUP'99 intrusion detection dataset, which contains 494,020 connection records. As in [O'Callaghan et al. 2002; Aggarwal et al. 2003; Ackermann et al. 2010], we employed the 34 continuous attributes and the 10% version of the full dataset, which has been widely used to evaluate algorithms for clustering data streams [O'Callaghan et al. 2002; Aggarwal et al. 2003; Ackermann et al. 2010]. Each object refers to either a normal connection or an intrusion. The dataset was converted into a data stream by taking the data input order as the streaming order, and we normalized it to keep the attribute values within [0, 1]. Chunks formed by 2,000 objects were processed ($N = 2000$), and no more data are sent until the previous chunk has been processed.

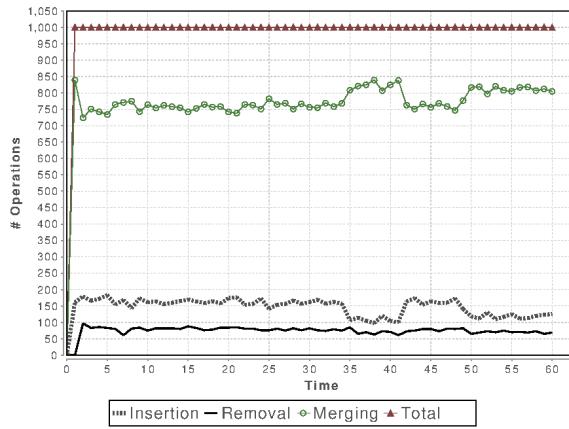


Fig. 15. Number of CluStream operations: insertion, removal, merging, and total.

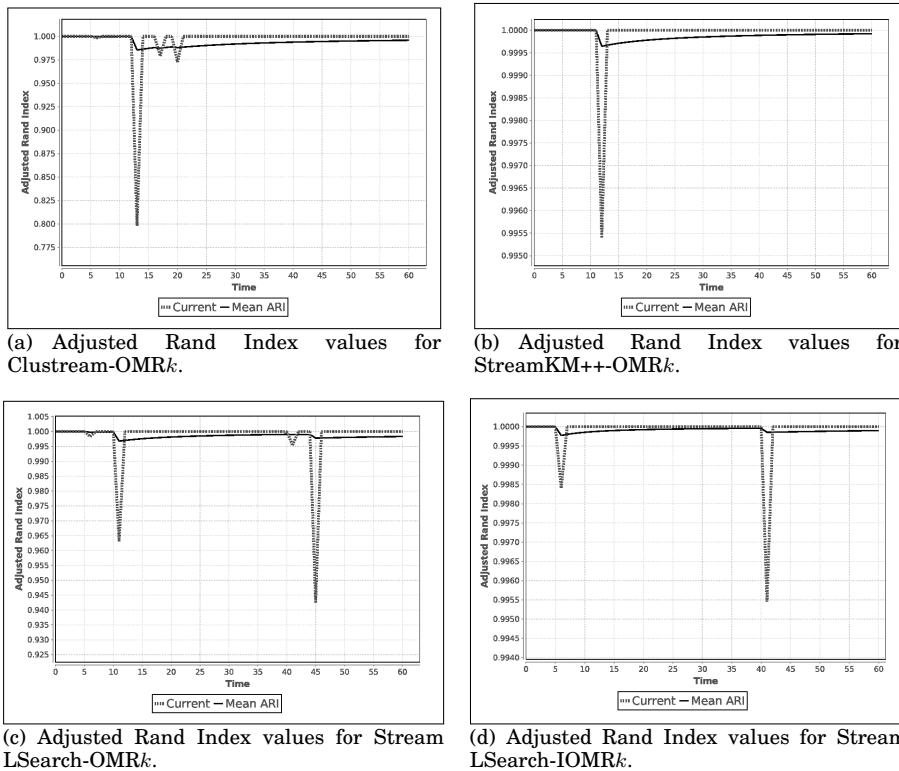


Fig. 16. Adjusted Rand Index values over time, using a 64-dimensional synthetic data set.

Table III. Silhouettes and Processing Time for the KDDCup'99 dataset.

	Simplified Silhouette $\mu(\pm\sigma)$	Processing time (milliseconds) $\mu(\pm\sigma)$
CLS-BkM	0.82 (± 0.17)	1,320.48 ($\pm 1,316.82$)
CLS-OMRk	0.85 (± 0.19)	1,760.36 ($\pm 1,361.64$)
SKM-BkM	0.90 (± 0.13)	129.17 (± 33.58)
SKM-OMRk	0.92 (± 0.11)	352.22 (± 68.49)
SLS-BkM	0.89 (± 0.13)	1,820.31 ($\pm 1,128.43$)
SLS-IBkM	0.98 (± 0.03)	1,875.54 ($\pm 3,457.29$)
SLS-OMRk	0.90 (± 0.13)	51,698.10 ($\pm 24,697.32$)
SLS-IOMRk	0.98 (± 0.02)	4,984.70 ($\pm 9,087.23$)

The correct number of clusters is *a priori* unknown for this dataset. In addition, it is usually unsafe to assume that the classes correspond to clusters. It is probably for this reason that most of the studies reported in the literature, *e.g.*, [O'Callaghan et al. 2002; Aggarwal et al. 2003; Ackermann et al. 2010], focus on the so-called mean squared error of the data partition. However, it is well-known that this criterion is not appropriate for evaluating partitions with a different number of clusters. Therefore, we assess the obtained partitions by means of a relative validity index (Simplified Silhouette criterion).

The parameters of the stream clustering algorithms were set aimed at favoring fair comparisons among them. Following [Aggarwal et al. 2003], the CluStream's parameters were set as $Init = 2,000$, $m = 50$, $\delta = 512$, and 200 micro-clusters, and the radius factor was set to 2. For StreamKM++, the size of the coresnet was set to 200. For Stream LSearch, because of the evolving nature of the data stream, the storage of cluster prototypes obtained from previous data chunks was not performed because out-of-date prototypes could dominate the overall stream behavior, thus possibly deteriorating the clustering quality.

Table III presents the results that were obtained for both the Simplified Silhouette criterion and the processing time. The clustering algorithms were evaluated for every set of 2,000 objects. Thus, for this dataset, we have 247 silhouette values and processing time. Table III reports the mean and standard deviations for such 247 values (per algorithm). According to the SS values, good results were obtained that range from 0.82 to 0.98. Additionally, SKM-BkM and SKM-OMRk provide the lowest average processing times approximately 129 and 352 milliseconds, respectively, and are eligible to process high-speed streams. We also note that BkM-based algorithms have been shown to be faster than OMRk-based algorithms. Specifically, SKM-BkM is 2.72 times faster than SKM-OMRk, CLS-BkM is 1.33 times faster than CLS-OMRk, and SLS-BkM is 28.40 times faster than SLS-OMRk. The incremental versions of SLS-OMRk and SLS-BkM have shown performance gains. In particular, SLS-IOMRk is 10.37 times faster than SLS-OMRk.

Fig. 17 illustrates some specific aspects of the processing times (on log-scale) of the algorithms that deserve further comment with respect to a series of objects that correspond to the *attack connections*. For example, the CLS-BkM and CLS-OMRk processing times are low at *Chunk #119*, where there is a *smurf attack*. In this type of attack, CluStream simply updates the closest micro-cluster, which is less computationally expensive than merging micro-clusters. When there are attacks of different types, the processing time increases because of the merge operations performed by CluStream that have quadratic time complexity with respect to the number of micro-clusters. The SLS-BkM and SLS-OMRk algorithms show the highest processing times mainly because Stream LSearch uses all of the objects from a chunk, while CluStream and StreamKM++ make use of a sample of the chunk only. However, the SLS-IBkM and SLS-IOMRk algorithms have shown to be competitive with the CluStream and

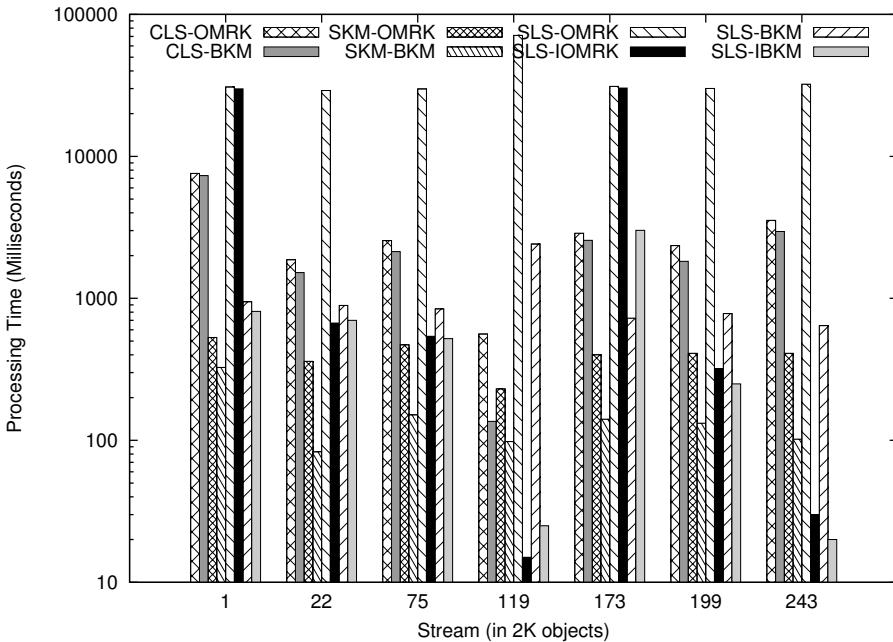


Fig. 17. Stream Processing Time for the KDDCup'99 data.

StreamKM++. At *Chunk #1*, the incremental variants of the SLS-OMR k and SLS-BkM algorithms have processing costs that are close to the non-incremental versions. The reason is that, in the first chunk, the data partition is initialized by running the respective \hat{k} -Estimation algorithms BkM or OMR k . Similar to what was observed for the synthetic datasets, the BkM-based algorithms have shown low processing times compared with than their OMR k -based counterparts.

Table IV presents the results of statistical tests of significance for pair-wise comparisons between the algorithms. Following the same notation adopted for Table II, one can observe that SLS-IBkM and SLS-IOMR k have been shown to be the best algorithms with regard to the SS criterion (six wins and no losses). Considering the processing times (Table V), SLS-IBkM and SLS-IOMR k (with six wins) are also the most computationally efficient algorithms. By accounting for both the partition quality and the processing times, SLS-IBkM exhibited the best trade-off solutions.

We can observe that, in terms of the data partition quality, all of the algorithms provided competitive results on synthetic and real scenarios. However, in terms of the processing times, the performances of the algorithms are distinct. On the synthetic data, the SKM-OMR k algorithm shows the best trade-off between quality and performance. For real-world scenarios, we observe that SLS-IBkM has shown good results. The main reason for SLS-IBkM to perform well on the KDDCup'99 data is due to its incremental component, which can avoid unnecessary estimation of the number of clusters. On the KDDCup'99 data, the classes are unbalanced, and the variability among the classes is low. If a specific type of attack is well represented by a set of clusters that constitute a small portion of data, the objects of this same class that continuously arrive will cause updating of the clusters, which is computationally less expensive than estimating \hat{k} . Because this behavior is frequent on the KDDCup'99 dataset, in only a few cases the number of clusters must be estimated (17/247 chunks).

Table IV. Statistically significant differences for silhouettes on the KDDCup'99 data.

	SLS-IB k M	SLS-IOMR k	SLS-OMR k	SKM-OMR k	SKM-B k M	SLS-B k M	CLS-OMR k	CLS-OMR k
SLS-IB k M								
SLS-IOMR k	○							
SLS-OMR k	—	—		○		+	+	+
SKM-OMR k	—	—	○			○	○	+
SKM-B k M	—	—	—	○		○	○	+
SLS-B k M	—	—	—	○	○		—	+
CLS-OMR k	—	—	—	—	—	—	—	—
CLS-OMR k	—	—	—	—	—	—	—	—

Table V. Statistically significant differences for processing time on the KDDCup'99 data.

	SLS-IB k M	SLS-IOMR k	SLS-OMR k	SKM-IB k M	SKM-OMR k	CLS-IB k M	SLS-B k M	CLS-OMR k	SLS-OMR k
SLS-IB k M									
SLS-IOMR k	○								
SLS-OMR k	—	—			+	+	+	+	+
SKM-IB k M	—	—	—		+	+	+	+	+
SKM-OMR k	—	—	—	—	○	○	+	+	+
CLS-IB k M	—	—	—	—	—	—	—	—	—
SLS-B k M	—	—	—	—	—	—	—	○	+
CLS-OMR k	—	—	—	—	—	—	—	—	—
SLS-OMR k	—	—	—	—	—	—	—	—	—

7. CONCLUDING REMARKS

Many clustering algorithms based on k -Means for processing data streams have been studied. Most of them assume that the number of clusters, k , is known and fixed *a priori* by the user. Aimed at relaxing this assumption, which is often unrealistic in practical applications, we proposed a support system that allows not only estimating the number of clusters automatically from data but also monitoring the process of data stream clustering. The potential of the proposed system has been illustrated by means of a prototype that implements eight algorithms for clustering data streams, specifically, Stream LSearch-OMR k , Stream LSearch-B k M, Stream LSearch-IOMR k , Stream LSearch-IB k M, CluStream-OMR k , CluStream-B k M, StreamKM++-OMR k and StreamKM++-B k M. These algorithms are essentially combinations of three state-of-the-art algorithms for clustering data streams with fixed k , namely, Stream LSearch [O'Callaghan et al. 2002], CluStream [Aggarwal et al. 2003], and StreamKM++ [Ackermann et al. 2010], with two well-known algorithms for estimating the number of clusters, which are Ordered Multiple Runs of k -Means (OMR k) and Bisecting k -Means (B k M) and its incremental versions IB k M and IOMR k . Such combinations have been designed to retain the desirable properties of the data stream algorithms, while additionally allowing them to automatically estimate the number of clusters from the data.

We experimentally evaluated the performance of the proposed system with both the synthetic and real-world data streams. The implemented prototype shows that such a system not only allows estimating the number of clusters automatically from the data but also monitors the process of data stream clustering. We also illustrated how

the proposed system can support a variety of data inputs, data visualization and projection techniques, data stream algorithms, and clustering validity measures. Finally, as an additional contribution, we experimentally compared the performance of state-of-the-art clustering algorithms for data streams on both synthetic and real-world data. From this perspective, analyses of statistical significance suggested that algorithms based on OMR k yield to the best data partitions, while the algorithms based on BkM are more computationally efficient. Additionally, StreamKM++-OMR k and Stream LSearch-IBkM provided the best trade-off relationship between accuracy and efficiency.

As future work, the incorporation of other clustering algorithms into the proposed system is promising – e.g., fuzzy clustering algorithms [Crespo and Weber 2005; Pedrycz and Weber 2008]. Additionally, aimed at further improving the data visualization capabilities of the proposed system, more sophisticated methods can be used, in such a way that the obtained clustering becomes more interpretable to the data analyst.

REFERENCES

- Marcel R. Ackermann, Christiane Lammersen, Marcus Märtens, Christoph Raupach, Christian Sohler, and Kamil Swierkot. 2010. StreamKM++: A Clustering Algorithms for Data Streams. In *Proc. of the ALENEX*. 173–187.
- Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. 2003. A framework for clustering evolving data streams. In *Proc. of the VLDB*. 81–92.
- Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. 2004. A framework for projected clustering of high dimensional data streams. In *Proceedings of the 30th international conference on Very large data bases (VLDB '04)*. VLDB Endowment, 852–863.
- Michael R. Anderberg. 1973. *Cluster Analysis for Applications*. Academic Press.
- David Arthur and Sergei Vassilvitskii. 2007. k-means++: the advantages of careful seeding. In *Proc. of the SODA'07*. 1027–1035.
- Jürgen Beringer and Eyke Hüllermeier. 2006. Online clustering of parallel data streams. *Data and Knowledge Engineering* 58 (2006), 180 – 204.
- Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. MOA: Massive Online Analysis. *Journal of Machine Learning Research* 11 (2010), 1601–1604.
- Abdelhamid Bouchachia. 2011. Evolving clustering: an asset for evolving systems. In *IEEE SMC Newsletter*, Vol. 36. 1–6.
- T. Calinski and J. Harabasz. 1974. A dendrite method for cluster analysis. *Communications in Statistics* 3 (1974), 1–27.
- Thiago F. Covões and Eduardo R. Hruschka. 2011. Towards improving cluster-based feature selection with a simplified silhouette filter. *Information Sciences* 181, 18 (2011), 3766–3782.
- Fernando Crespo and Richard Weber. 2005. A methodology for dynamic data mining based on fuzzy clustering. *Fuzzy Sets and Systems* 150 (2005), 267 – 284.
- David L. Davies and Donald W. Bouldin. 1979. A Cluster Separation Measure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 1 (1979), 224 –227.
- Jonathan de Andrade Silva and Eduardo Raul Hruschka. 2011. Extending k-Means-Based Algorithms for Evolving Data Streams with Variable Number of Clusters. In *Fourth International Conference on Machine Learning and Applications - ICMLA'11*, Vol. 2. 14–19.
- Janez Demšar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research* 7 (2006), 1–30.
- J. C. Dunn. 1974. Well separated clusters and optimal fuzzy-partitions. *Journal of Cybernetics* 4 (1974), 95–104.
- Brian S. Everitt, Sabine Landau, and Morven Leese. 2001. *Cluster Analysis*. Arnold Publishers.
- Dominik Fisch, Dominik Fisch, Martin Jänicke, Edgar Kalkowski, and Bernhard Sick. 2012. Techniques for Knowledge Acquisition in Dynamically Changing Environments. *ACM Transactions on Autonomous and Adaptive Systems* 7 (2012), 16:1–16:25.
- Joao Gama. 2010. *Knowledge Discovery from Data Streams*. Chapman Hall/CRC.

- Guha, Meyerson, Mishra, Motwani, and O'Callaghan. 2003. Clustering Data Streams: Theory and Practice. *IEEE Transactions on Knowledge and Data Engineering* 15 (2003).
- Jiawei Han and Micheline Kamber. 2000. *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*.
- Myles Hollander and Douglas A. Wolfe. 1999. *Nonparametric statistical methods* (2nd ed.). Wiley.
- E.R. Hruschka, L.N. de Castro, and R. J G B Campello. 2004. Evolutionary algorithms for clustering gene-expression data. In *Data Mining, 2004. ICDM '04. Fourth IEEE International Conference on*. 403–406.
- Eduardo R. Hruschka, Ricardo J. G. B. Campello, and Leandro Nunes de Castro. 2006. Evolving clusters in gene-expression data. *Information Sciences* 176 (2006), 1898–1927.
- L. Hubert and P. Arabie. 1985. Comparing Partitions. *Journal of Classification* 2 (1985), 193–218.
- Anil K. Jain. 2009. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters* 31 (2009), 651–666.
- Anil K. Jain and Richard C. Dubes. 1988. *Algorithms for clustering data*. Prentice-Hall, Inc.
- L. Kaufman and P. J. Rousseeuw. 1990. *Finding groups in data: an introduction to cluster analysis*. John Wiley and Sons, New York.
- Edwin Lughofer. 2011. *Evolving Fuzzy Systems - Methodologies, Advanced Concepts and Applications*. Studies in Fuzziness and Soft Computing, Vol. 266. Springer.
- Edwin Lughofer. 2012. A dynamic split-and-merge approach for evolving cluster models. *Evolving Systems* 3 (2012), 135–151.
- Moamar S. Mouchaweh. 2010. Learning in Dynamic Environments: Application to the Identification of Hybrid Dynamic Systems. In *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*. 555–560.
- Murilo C. Naldi, Ricardo J.G.B. Campello, Eduardo R. Hruschka, and André C.P.L.F. Carvalho. 2011. Efficiency Issues of Evolutionary k-Means. *Applied Soft Computing* 11 (2011), 1938–1952.
- Murilo C. Naldi, André Fontana, and Ricardo J. G. B. Campello. 2009. Comparison Among Methods for k Estimation in k-means. In *Proc. of the ISDA'09*. 1006–1013.
- Liadan O'Callaghan, Adam Meyerson, Rajeev Motwani, Nina Mishra, and Sudipto Guha. 2002. Streaming-Data Algorithms for High-Quality Clustering. In *Proc. of the ICDE*. 685–695.
- N. R. Pal and J. C. Bezdek. 1995. On cluster validity for the fuzzy c-means model. *IEEE Transactions on Fuzzy Systems* 3 (1995), 370–379.
- K. Pearson. 1901. On lines and planes of closest fit to systems of points in space. *Philos. Mag.* 2, 6 (1901), 559–572.
- Witold Pedrycz and Richard Weber. 2008. Editorial: Special Issue on Soft Computing for Dynamic Data Mining. *Applied Soft Computing* 8 (2008), 1281–1282.
- Andres Quiroz, Manish Parashar, Nathan Gnanasambandam, and Naveen Sharma. 2012. Design and Evaluation of Decentralized Online Clustering. *ACM Transactions on Autonomous and Adaptive Systems* 7 (2012), 34:1–34:31.
- Moamar Sayed Mouchaweh and Edwin Lughofer. 2012. *Learning in non-stationary environments: Methods and Applications*. Springer.
- Jonathan A. Silva, Elaine R. Faria, Rodrigo C. Barros, Eduardo R. Hruschka, André C. P. L. F. de Carvalho, and João Gama. 2013. Data Stream Clustering: A Survey. *ACM Comput. Surv.* 46, 1 (2013), 13:1–13:31.
- Michael Steinbach, George Karypis, and Vipin Kumar. 2000. A comparison of document clustering techniques. In *Proc. of the KDD Workshop on Text Mining*. 109–111.
- Lucas Vendramin, Ricardo J. G. B. Campello, and Eduardo R. Hruschka. 2010. Relative clustering validity criteria: A comparative overview. *Statistical Analysis and Data Mining* 3 (2010), 209–235.
- Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. 2007. Top 10 algorithms in data mining. *Knowledge and Information Systems* 14 (2007), 1–37.
- Zhenwei Yu, Jeffrey J. P. Tsai, and Thomas Weigert. 2008. An Adaptive Automatically Tuning Intrusion Detection System. *ACM Transactions on Autonomous and Adaptive Systems* 3 (2008), 10:1–10:25.
- Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proc. of the SIGMOD'96*. 103–114.