Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Reduzindo Custos da Deduplicação de Dados Utilizando Heurísticas e Computação em Nuvem

Dimas Cassimiro do Nascimento Filho

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Banco de Dados

Dr. Carlos Eduardo Santos Pires
(Orientador)

Campina Grande, Paraíba, Brasil

"REDUZINDO CUSTOS DA DEDUPLICAÇÃO DE DADOS UTILIZANDO HEURÍSTICAS E COMPUTAÇÃO EM NUVEM"

DIMAS CASSIMIRO DO NASCIMENTO FILHO

TESE APROVADA EM 10/11/2017

CARLOS EDUARDO SANTOS PIRES, Dr., UFCG
Orientador(a)

CLÁUDIO ELIZIO CALAZANS CAMPELO, PhD., UFCG
Examinador(a)

LEANDRO BALBY MARINHO, Dr., UFCG
Examinador(a)

RENATA DE MATOS GALANTE, Dra., UFRGS
Examinador(a)

JOSÉ MARIA DA SILVA MONTEIRO FILHO, Dr., UFC
Examinador(a)

CAMPINA GRANDE - PB

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

N244r Nascimento Filho, Dimas Cassimiro do.

Reduzindo custos da deduplicação de dados utilizando heurísticas e computação em nuvem / Dimas Cassimiro do Nascimento Filho. – Campina Grande, 2017.

237 f. : il. color.

Tese (Doutorado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2017.

"Orientação: Prof. Dr. Carlos Eduardo Santos Pires".

Referências.

1. Qualidade de Dados. 2. Deduplicação de Dados. 3. Big Data. 4. Computação em Nuvem. 5. Heurísticas. I. Pires, Carlos Eduardo Santos. II. Título.

CDU 004.41(043)

Resumo

Na era de Big Data, na qual a escala dos dados provê inúmeros desafios para algoritmos clássicos, a tarefa de avaliar a qualidade dos dados pode se tornar custosa e apresentar tempos de execução elevados. Por este motivo, gerentes de negócio podem optar por terceirizar o monitoramento da qualidade de bancos de dados para um serviço específico, usualmente baseado em computação em nuvem. Neste contexto, este trabalho propõe abordagens para redução de custos da tarefa de deduplicação de dados, a qual visa detectar entidades duplicadas em bases de dados, no contexto de um serviço de qualidade de dados em nuvem. O trabalho tem como foco a tarefa de deduplicação de dados devido a sua importância em diversos contextos e sua elevada complexidade. É proposta a arquitetura em alto nível de um serviço de monitoramento de qualidade de dados que emprega o provisionamento dinâmico de recursos computacionais por meio da utilização de heurísticas e técnicas de aprendizado de máquina. Além disso, são propostas abordagens para a adoção de algoritmos incrementais de deduplicação de dados e controle do tamanho de blocos gerados na etapa de indexação do problema investigado. Foram conduzidos quatro experimentos diferentes visando avaliar a eficácia dos algoritmos de provisionamento de recursos propostos e das heurísticas empregadas no contexto de algoritmos incrementais de deduplicação de dados e de controle de tamanho dos blocos. Os resultados dos experimentos apresentam uma gama de opções englobando diferentes relações de custo e benefício, envolvendo principalmente: custo de infraestrutura do serviço e quantidade de violações de SLA ao longo do tempo. Outrossim, a avaliação empírica das heurísticas propostas para o problema de deduplicação incremental de dados também apresentou uma série de padrões nos resultados, envolvendo principalmente o tempo de execução das heurísticas e os resultados de eficácia produzidos. Por fim, foram avaliadas diversas heurísticas para controlar o tamanho dos blocos produzidos em uma tarefa de deduplicação de dados, cujos resultados de eficácia são bastante influenciados pelos valores dos parâmetros empregados. Além disso, as heurísticas apresentaram resultados de eficiência que variam significativamente, dependendo da estratégia de poda de blocos adotada. Os resultados dos quatro experimentos conduzidos apresentam suporte para demonstrar que diferentes estratégias (associadas ao provisionamento de recursos computacionais e aos

algoritmos de qualidade de dados) adotadas por um serviço de qualidade de dados podem influenciar significativamente nos custos do serviço e, consequentemente, os custos repassados aos usuários do serviço.

palavras-chave: Qualidade de Dados, Deduplicação de Dados, Big Data, Computação em Nuvem, Heurísticas

Abstract

In the era of Big Data, in which the scale of the data provides many challenges for classical algorithms, the task of assessing the quality of datasets may become costly and complex. For this reason, business managers may opt to outsource the data quality monitoring for a specific cloud service for this purpose. In this context, this work proposes approaches for reducing the costs generated from solutions for the data deduplication problem, which aims to detect duplicate entities in datasets, in the context of a service for data quality monitoring. This work investigates the deduplication task due to its importance in a variety of contexts and its high complexity. We propose a high-level architecture of a service for data quality monitoring, which employs provisioning algorithms that use heuristics and machine learning techniques. Furthermore, we propose approaches for the adoption of incremental data quality algorithms and heuristics for controlling the size of the blocks produced in the indexing phase of the investigated problem. Four different experiments have been conducted to evaluate the effectiveness of the proposed provisioning algorithms, the heuristics for incremental record linkage and the heuristics to control block sizes for entity resolution. The results of the experiments show a range of options covering different tradeoffs, which involves: infrastructure costs of the service and the amount of SLA violations over time. In turn, the empirical evaluation of the proposed heuristics for incremental record linkage also presented a number of patterns in the results, which involves tradeoffs between the runtime of the heuristics and the obtained efficacy results. Lastly, the evaluation of the heuristics proposed to control block sizes have presented a large number of tradeoffs regarding execution time, amount of pruning approaches and the obtained efficacy results. Besides, the efficiency results of these heuristics may vary significantly, depending of the adopted pruning strategy. The results from the conducted experiments support the fact that different approaches (associated with cloud computing provisioning and the employed data quality algorithms) adopted by a data quality service may produce significant influence over the generated service costs, and thus, the final costs forwarded to the service customers.

key words: Data Quality, Deduplication, Big Data, Cloud Computing, Heuristics

Agradecimentos

A Deus, pelo dom da vida e pela força e esperança nos momentos difíceis.

Ao meu orientador (Carlos Eduardo), por toda paciência durante este longo caminho e pela dedicação nas incontáveis reuniões e revisões necessárias para aprimorar brilhantemente o trabalho desenvolvido.

Aos meus pais (Teresinha Mendonça e Dimas Cassimiro), por todo apoio incondicional e por me ensinarem durante toda a vida que com disciplina, organização e resiliência é possível realizar qualquer sonho. Aos meus irmãos (Lucila e Diógenes), pelos ótimos momentos de alegria e esperança durante nossas vidas. Ao meu "amore" (Nathalia Diniz), por todo carinho, paciência e momentos de alegria.

Aos meus amigos do Laboratório de Qualidade de Dados (Demetrio, Brasileiro e Nóbrega), pelas incontáveis contribuições e discussões sobre o meu trabalho, e pela companhia em várias aventuras durante o doutorado.

Aos lendários moradores do Chiqueirinho (Dr. Rodrigo. Dr. Gabriel e Dr. Ryan), pela ótima convivência durante esses anos e pela mentoria sobre assuntos não acadêmicos (tempo ao tempo!). Ao meu brother Luis, pelas contribuições neste trabalho e por incontáveis discussões aleatórias sobre ciência, filosofia, epistemologia, finanças e empreendedorismo.

Aos meus grandes amigos "do tempo de graduação" (Camila, Fabiano, Vicente, Vinicius, Fernando e Zé Filho), pela amizade que tanto me ajudou a seguir em frente.

Aos meus padrinhos (Francisca e Gilson - *In memoriam*), por todos os sorrisos e palavras de incentivo ao longo da minha vida.

A todos os meus professores da UFCG, os quais foram indescritivelmente competentes em seus ensinamentos (Esta é minha forma de agradecer!).

The purpose of today's training is to defeat yesterday's understanding. (Myiamoto Musashi)
Fall seven times and stand up eight. (Japanese Proverb)
For when I am weak, then I am strong. (Corinthians 12:10)

Conteúdo

1 Introdução **1**

1.1 Problematização 3

1.2 Escopo na Área de Qualidade de Dados 6

1.3 Objetivos 6

1.3.1 Objetivo Geral 6

1.3.2 Objetivos Específicos 7

1.4 Relevância 8

1.5 Metodologia 9

1.6 Principais Contribuições 9

1.7 Indicadores de Pesquisa Alcançados 10

1.8 Estrutura do Documento 11

2 Fundamentação Teórica **14**

2.1 Qualidade de Dados 14

2.2 Deduplicação de Dados 17

2.2.1 Indexação 19

2.2.2 Técnicas de Classificação de Entidades Duplicadas 20

2.2.3 Relevância do Problema de Deduplicação de Dados 21

2.3 Classificação Coletiva para Deduplicaçação de Dados 22

2.3.1 Grafo de Similaridade 23

2.3.2 Deduplicação Incremental de Dados 24

2.3.3 Algoritmos de Agrupamento Automático em Grafos 29

2.4 Software como Serviço 32

2.5 Computação em Nuvem 33

2.6 Classificação em Aprendizado de Máquina 35
2.7 Considerações Finais 37

3 Trabalhos Relacionados **39**
3.1 Provisionamento de Recursos em Computação em Nuvem 39
3.2 Algoritmos para Deduplicação Incremental de Dados 41
3.3 Algoritmos Utilizados no Controle do Tamanho de Blocos de Entidades . . 43

4 Arquitetura de um Serviço de Monitoramento de Qualidade de Dados (SMQD) **48**
4.1 SLA de Qualidade de Dados 49
4.2 Arquitetura Proposta 51
4.3 Considerações Finais 55

5 Provisionamento Dinâmico de Recursos Computacionais em um SMQD **56**
5.1 Modelo de Custo 57
5.1.1 Custos do Serviço 58
5.2 Algoritmos para Provisionamento de Recursos Computacionais 60
5.2.1 Formalização do Problema 61
5.2.2 Algoritmos de Provisionamento baseados em Aprendizado de Máquina 63
5.2.3 Algoritmos de Provisionamento para Processamento de Tarefas Disparadas por um Único DQSLA 65
5.2.4 Algoritmos de Provisionamento para Processamento de Tarefas Disparadas Inicialmente por Múltiplos DQSLAs 68
5.3 Considerações Finais 74

6 Estratégias para Deduplicação Incremental de Dados **76**
6.1 Métricas para Avaliação de Algoritmos Incrementais de Deduplicação de Dados 77
6.1.1 Heurísticas para Deduplicação Incremental de Dados 84
6.2 Considerações Finais 88

7 Controle do Tamanho de Blocos para Deduplicação de Dados **90**
7.1 Motivação 91

7.2 Exemplo 93
7.3 Notação Adotada 96
7.3.1 Métricas de qualidade 97
7.3.2 Poda de Coleção de Blocagens 99
7.3.3 Poda de Coleção de Blocagens para o Controle do Tamanho dos Blocos 100
7.4 Abordagem proposta 101
7.4.1 Co Ocorrência de Entidades em Blocos 101
7.4.2 Esquema de Pontuação de Co Ocorrência 102
7.4.3 Pontuação de Co Ocorrência Agregada 102
7.4.4 Visão Geral da Abordagem 103
7.4.5 Metaheurística MkPCBS 105
7.5 Considerações Finais 106

8 Avaliação **107**
8.1 Hipóteses e Experimentos 107
8.2 Experimento I 109
8.2.1 Design 109
8.2.2 Modelo de Execução 110
8.2.3 Resultados 112
8.2.4 Discussão 114
8.3 Experimento II 115
8.3.1 Design 115
8.3.2 Modelo de execução 117
8.3.3 Implementação 121
8.3.4 Resultados 121
8.3.5 Discussão 122
8.4 Experimento III 130
8.4.1 Design Experimental 132
8.4.2 Implementação 134
8.4.3 Resultados 134
8.4.4 Discussão 138

8.5 Experimento IV 145
8.5.1 Bases de Dados 145
8.5.2 Design Experimental 146
8.5.3 Resultados 148
8.5.4 Discussão 149
8.6 Ameaças à Validade 166

9 Conclusões e Trabalhos Futuros **169**
9.1 Conclusões 169
9.2 Trabalhos Futuros 172

A Tabelas de Símbolos **187**

B Semântica das Operações do Algoritmo Greedy **191**

C Lista de Algoritmos Propostos **193**
C.1 Algoritmos para Adaptação de Classe de Configuração 193
C.2 Algoritmos para Estimativa de Classe de Configuração Inicial 194
C.3 Heurísticas para Seleção de Algoritmos de Provisionamento e Ajuste de Classe de Configuração 198
C.3.1 Heurística *Probabilistic Best Performing Allocation* 198
C.3.2 Heurística *Tunable Allocation* 202
C.3.3 Heurística *Adaptive Allocation* 204
C.3.4 Heurística *Sliced Training Data* 206
C.4 Heurísticas para Deduplicação Incremental de Dados 207
C.5 Heurísticas para Controlar o Tamanho de Blocos 213
C.5.1 Heurísticas para Reduzir o Tamanho de Blocos ($Shrink$) 213
C.5.2 Heurística para Dividir Blocos 219
C.5.3 Heurística para Unir Blocos 221
C.5.4 Heurística para Excluir Blocos 224

D Prova de Lemas, Proposições e Teoremas **227**

D.1 Desbalanceamento de Carga Associado à Alocação de Máquinas Virtuais com Configuração Heterogênea 227
D.2 Convergência do Algoritmo Hill Climbing Estimation 230
D.3 Prova de Lemas e Teoremas Relacionados ao Problema de Provisionamento de Recursos Computacionais em um SMQD 231
D.4 Desigualdade Relacionada à Precisão de Agrupamentos 233

E Regras de Similaridade e Funções de Chave de Bloco 234
E.1 Experimento III 234
E.2 Experimento IV 235

F Tamanho da Janela Fixa Empregado pelo Algoritmo *Sorted Neighborhood* 236

Lista de Símbolos

BNF - *Backus Naur Form*

CC - *Correlation Clustering*

CPr - *Precisão de Agrupamento*

CRc - *Cobertura de Agrupamento*

CRM - *Customer Relationship Management*

DID - *Deduplicação Incremental de Dados*

DT - *Delay Time*

DQSLA - *Data Quality SLA*

ERP - *Enterprise Resource Planning*

KNN - *K Nearest Neighbors*

IaaS - *Infrastructure as a Service*

PaaS - *Platform as a Service*

PC - *Pair Completeness*

PCPr - *Precisão de Agrupamento Penalizada*

PQ - *Pair Quality*

QoS - *Quality of Service*

RF - *Random Forests*

RR - *Reduction Ratio*

SaaS - *Software as a Service*

SLA - *Service Level Agreement*

SMQD - *Serviço de Monitoramento de Qualidade de Dados*

SVM - *Support Vector Machine*

TD - *Training Data*

VM - *Virtual Machine*

XML - *eXtensible Markup Language*

Lista de Figuras

2.1 Etapas do processo de deduplicação de dados (adaptada de [15]). 19
2.2 Etapas do processo de classificação coletiva de entidades duplicadas. 23
2.3 Etapas do processo de deduplicação incremental de dados no contexto de classificação coletiva. 25
2.4 Um grafo de similaridade G e um conjunto de incrementos de dados a serem realizados sobre G. 26
2.5 Resultado da aplicação de um conjunto de incrementos de dados sobre o grafo de similaridade G. 27
2.6 Exemplo de funcionamento dos algoritmos (a) Center e (b) Merge-Center (extraída de [33]). 31
2.7 Exemplo de agrupamento em forma de estrela produzido pelo algoritmo Star [2]. 32
2.8 Pilha de serviços baseados em computação em nuvem. 35

4.1 Arquitetura proposta para o serviço de monitoramento de qualidade de dados. 54

5.1 Relação entre classes de configuração e os recursos computacionais alocados. 58
5.2 Diferenças entre o tempo de execução e o tempo de restrição produzidos pela sub provisão, alocação ótima e super provisão de recursos. 61
5.3 Ilustração das regiões de classes de configuração: sub provisão, alocação ótima e super provisão de recursos computacionais para a execução da tarefa τ. 63
5.4 Ilustração do funcionamento do algoritmo *Hill Climbing Provisioning*. . . . 68
5.5 Ilustração do funcionamento do algoritmo *Sliced Tweak Solution*. 70
5.6 Ilustração do funcionamento do algoritmo *Best Performing Allocation*. . . . 73

6.1 (a) Resultados de agrupamentos em que $CPr = 1$, ainda que $|\mathcal{L}_G| \neq |\mathcal{G}|$. (b) Resultados de agrupamentos em que $PCPr$ é alto $(0,87)$, porém a precisão dos agrupamentos é bastante baixa. 79
6.2 (a) Exemplo de grafo de similaridade em que a minimização da função de penalidade gera perda de cobertura. (b) Exemplo de grafo de similaridade em que um resultado não ótimo de função de penalidade produz cobertura perfeita. 86
6.3 Exemplo de grafo de similaridade em que a minimização da função de penalidade gera perda de precisão. (b) Exemplo de grafo de similaridade em que um resultado não ótimo de função de penalidade produz precisão perfeita. . 86

7.1 Visão geral da abordagem proposta para poda de coleção de blocagens para controle do tamanho de blocos. 104

8.1 Custo acumulado do serviço produzido pelo algoritmo mais eficaz de cada grupo avaliado. 113
8.2 Quantidade de violações de SLA produzida pelo algoritmo mais eficaz de cada grupo avaliado. 113
8.3 Custo acumulado do serviço produzido, por algoritmo de provisionamento. 122
8.4 Quantidade de violações de SLA produzida, por algoritmo de provisionamento. 122
8.5 Custo acumulado do serviço produzido, por algoritmo de provisionamento. 123
8.6 Quantidade de violações de SLA produzida, por algoritmo de provisionamento. 123
8.7 Custo acumulado do serviço produzido, por algoritmo de provisionamento. 124
8.8 Quantidade de violações de SLA produzida, por algoritmo de provisionamento. 124
8.9 Custo acumulado do serviço produzido, por algoritmo de provisionamento. 125
8.10 Quantidade de violações de SLA produzida, por algoritmo de provisionamento. 125
8.11 Custo acumulado do serviço produzido, por algoritmo de provisionamento. 126
8.12 Quantidade de violações de SLA produzida, por algoritmo de provisionamento. 126
8.13 Resultados de eficiência utilizando a base de dados Cora. 138
8.14 Resultados de eficiência utilizando a base de dados DBLPM4. 139
8.15 Resultados de eficiência utilizando a base de dados Febrl10k. 139
8.16 Resultados de eficiência utilizando a base de dados CoraATDV. 154

8.17 Resultados de eficiência utilizando a base de dados CoraATDV. 154
8.18 Resultados de eficiência utilizando a base de dados DBLPM4. 155
8.19 Resultados de eficiência utilizando a base de dados DBLPM4. 155
8.20 Resultados de eficiência utilizando a base de dados DS3. 156
8.21 Resultados de eficiência utilizando a base de dados DS3. 156
8.22 Resultados de eficiência utilizando a base de dados FebrlData. 157
8.23 Resultados de eficiência utilizando a base de dados FebrlData. 157

C.1 Ilustração do funcionamento do algoritmo *Binary Tweak Solution*. 193
C.2 Ilustração do funcionamento do algoritmo *Tunable Tweak Solution*. 196
C.3 Ilustração do funcionamento do algoritmo *Probabilistic Best Performing Allocation*. 200
C.4 Ilustração do funcionamento do algoritmo *Tunable Allocation*. 204
C.5 Ilustração do funcionamento do algoritmo *Adaptive Allocation*. 206

Lista de Tabelas

3.1 Comparativo entre abordagens para deduplicação incremental de dados no contexto de classificação coletiva. 43

3.2 Comparativo entre abordagens para controle do tamanho de blocos. 47

5.1 Problema de provisionamento de recursos computacionais no contexto de um SMQD. 61

6.1 objetivos sumarizados de um método de DID. 84

6.2 Objetivos formalizados de um filtro de componente de cobertura $\mathcal{F}$ para $\dot{T}(\Delta G)$. 89

7.1 Base de dados D_1. 94

8.1 Caracterização da Carga de Trabalho do Experimento I. 110

8.2 Chamadas intercambiáveis dos algoritmos de provisionamento utilizados no Experimento I. 111

8.3 Valores de parâmetros globais utilizados no Experimento I. 111

8.4 Notação do modelo matemático adotado no Experimento I. 112

8.5 Carga de trabalho do Experimento II. 116

8.6 Valores de parâmetros globais utilizados no Experimento II. 117

8.7 Design experimental do Experimento II. 118

8.8 Estatísticas das bases de dados utilizadas no Experimento III (empregando o algoritmo $Center$). 131

8.9 Algoritmos de DID gerados a partir da metaheurística CFG (utilizados na avaliação das bases de dados $Cora$ e $DBLPM4$). 133

8.10 Resultados de eficácia dos algoritmos de DID no Experimento III utilizando a base de dados $Febrl10k$. 135
8.11 Resultados de eficácia dos algoritmos de DID no Experimento III utilizando a base de dados $Cora$. 136
8.12 Resultados de eficácia dos algoritmos de DID no Experimento III utilizando a base de dados $DBLPM4$. 137
8.13 Lista de algoritmos gerados a partir da metaheurística $MkPCBS$. 146
8.14 Resultados de Qualidade utilizando a base de dados Cora. 150
8.15 Resultados de qualidade utilizando a base de dados DBLPM4, ... 151
8.16 Resultados de qualidade utilizando a base de dados DS3, ... 152
8.17 Resultados de qualidade utilizando a base de dados FebrlData, ... 153

A.1 Notação adotada (Computação em Nuvem). 188
A.2 Notação adotada (deduplicação incremental de dados no contexto de classificação coletiva). 189
A.3 Notação adotada (controle do tamanho de blocos). 190

E.1 Funções de chave de bloco empregadas no Experimento IV (F_{key}). 235

Capítulo 1

Introdução

Atualmente, as organizações produzem diariamente uma significativa quantidade de dados. Estima-se que a quantidade de dados armazenados em bancos de dados ao redor do mundo dobra a cada 20 meses [88]. Estes dados eram comumente vistos como meras entradas para a realização de tarefas do nível operacional nas empresas. No entanto, com o passar do tempo, percebeu-se a importância que os dados representavam para as organizações e os mesmos se tornaram um ativo estratégico e, muitas vezes, de principal importância para a realização de uma série de tarefas recorrentes no contexto empresarial, dentre as quais é possível destacar: a tomada de decisão e a descoberta de conhecimento.

O monitoramento de qualidade de dados é um processo contínuo que avalia uma base de dados para determinar se a mesma está de acordo com os objetivos planejados, ou seja, se provê dados com a quantidade e qualidade esperadas para suportar determinada tarefa. O processo de monitoramento de qualidade de dados baseia-se na seguinte premissa fundamental: qualidade de dados está intrinsecamente relacionada à intenção da utilização dos dados [7]. Por exemplo, no intuito de alcançar o potencial da utilização de um armazém de dados [41], é necessário desenvolver estratégias para manter níveis aceitáveis de confiança sobre os dados. Similarmente, outros níveis de confiança são também necessários em relação aos aspectos legais envolvendo os dados, aspectos de confidencialidade e padrões industriais. Desse modo, ao estabelecer padrões de qualidade de dados e adotar uma estratégia para um monitoramento contínuo de qualidade de dados que garanta a adoção dos padrões estabelecidos, a tendência é que ocorra um significativo aumento (ou pelo menos sejam mantidos níveis aceitáveis) da qualidade dos dados de uma organização ao longo do tempo. Por sua

vez, o aumento na qualidade dos dados irá resultar na redução do tempo gasto na avaliação e correção de dados, na otimização do tempo necessário para a entrega de dados confiáveis e na melhoria de confiança sobre as decisões [49]. Neste contexto, existe uma necessidade evidente de incorporar padrões de qualidade de dados em todo o ciclo de manipulação de dados, envolvendo tanto aspectos de governança de dados quanto aspectos técnicos de qualidade de dados [70].

Na prática, existem diferentes tipos de problemas de qualidade de dados. Problemas como valores ausentes, baixa acurácia, entidades duplicadas e heterogeneidade no formato de armazenamento são frequentes em bases de dados reais. Os problemas de qualidade de dados podem ser provenientes de diversos motivos e, de acordo com o contexto, podem apresentar algumas causas mais recorrentes do que outras. Uma das principais causas para a existência de problemas de qualidade de dados são erros humanos [41], tais como: erros de digitação, uso proposital de abreviações para acelerar a digitação, omissão de valores para simplificar cadastros recorrentes e introdução de valores errados devido ao entendimento incompleto dos formulários [36]. Ainda que já tenham sido desenvolvidas várias técnicas para detecção de problemas de qualidade de dados, as soluções existentes são frequentemente desafiadas por fatores como a quantidade e a diversidade dos dados processados [70]. Por este motivo, a área de Qualidade de Dados tem sido investigada por diversos autores. Apesar dos significativos avanços, estudos recentes apontaram que aproximadamente 68% dos problemas de qualidade nos dados são detectados por meio de reclamacões de usuários ou por pura sorte [70].

Um ciclo de monitoramento de qualidade de dados usualmente envolve uma etapa de avaliação, a qual visa acessar, processar e avaliar os dados de acordo com objetivos de qualidade de dados que, por sua vez, precisam levar em consideração os objetivos envolvidos no negócio [49]. Por sua vez, cada objetivo de qualidade de dados está relacionado com alguma dimensão [5] de qualidade de dados e pode gerar uma ou mais regras de qualidade de dados, as quais são mapeadas para algoritmos de qualidade de dados. Algumas tarefas de qualidade de dados, tais como deduplicação de dados e detecção de *outliers*, são especialmente desafiadoras de serem solucionadas e, na prática, requerem a utilização de algoritmos bastante sofisticados. Dependendo da complexidade destes algoritmos e da quantidade de dados que precisa ser processada, o monitoramento de qualidade de dados pode requerer uma grande quantidade de recursos computacionais.

Por exemplo, a tarefa de deduplicação de dados [1], a qual visa identificar entidades duplicadas em bases de dados e está relacionada com a dimensão de qualidade de dados denominada Duplicação [75], apresenta uma complexidade bastante elevada. Nesta tarefa, quando duas bases de dados (A e B) precisam ser processadas para identificar entidades duplicadas, potencialmente cada entidade em A precisa ser comparada com cada entidade em B, resultando em um número máximo de $(|A|\times|B|)$ comparações entre entidades [12]. Além da complexidade intrínseca relacionada à resolução deste problema, este processo de deduplicação de dados ainda precisa ser frequentemente re-executado quando os dados são alterados ao longo do tempo [30]. Considerando a complexidade intrínseca desta tarefa, este trabalho tem como foco a proposição e avaliação de técnicas para redução de custos computacionais da tarefa de deduplicação de dados.

1.1 Problematização

No contexto da tomada de decisão, a baixa qualidade dos dados influencia negativamente a interpretação das análises realizadas a partir destes dados e, consequentemente, compromete as decisões tomadas. Por exemplo, um processo de planejamento de uma cadeia de produção (envolvendo compra e estoque de matéria prima, produção e armazenamento de produtos) será muito provavelmente prejudicado caso sejam tomadas decisões baseadas em relatórios que contemplam dados de vendas com baixa acurácia, informações incompletas de fornecedores e/ou dados duplicados sobre o estoque de produtos. Por sua vez, no processo de descoberta de conhecimento, torna-se mais difícil aos gestores tomarem decisões inteligentes e bem fundamentadas caso sejam utilizadas informações e conhecimentos obtidos a partir de dados afetados por problemas de qualidade. Por exemplo, a presença de valores faltantes, fora do padrão (*outliers*) e/ou entidades duplicadas podem afetar significativamente o resultado proveniente de algoritmos de agrupamento automático.

Normalmente, a insuficiência de qualidade nos dados leva à tomada de más decisões, ou seja, uma decisão cujas consequências não atingem determinados valores de métricas de avaliação preestabelecidas. Más decisões, por sua vez, podem causar desde pequenos

[1] também chamada de resolução de entidades ou reconciliação de entidades (usualmente quando a quantidade de bases de dados envolvidas no processo é maior que 1)

inconvenientes operacionais, passando pela diminuição dos lucros até produzir a parada das atividades de uma organização [4]. Estas más decisões também conduzem a uma perda de credibilidade no sistema de informação. Além disso, problemas de qualidade de dados produzem sérias implicações na satisfação dos clientes e, por conseguinte, no sucesso da implantação de sistemas.

Na prática, o processo de monitoramento de qualidade de dados é desafiado pelos seguintes fatores: i) a quantidade de dados processados; ii) a heterogeneidade das fontes e formatos dos dados; iii) a complexidade computacional dos algoritmos de qualidade de dados; iv) a robustez da infraestrutura de hardware necessária para executar esses algoritmos; e v) a velocidade com que as mudanças (inserções, atualizações e remoções) afetam os dados. Considerando estes desafios, gerentes de negócio podem preferir terceirizar o processo de armazenamento e monitoramento contínuo da qualidade dos dados, seja por motivos operacionais ou financeiros.

No intuito de prestar serviços terceirizados, muitas empresas passaram a adotar o paradigma de Software como Serviço (em inglês, *Software as a Service - SaaS*) para disponibilizar serviços na Internet. Dentre as vantagens decorrentes da utilização de SaaS como modelo de negócio, podem ser destacadas: a cobrança proporcional ao serviço prestado e a facilidade de personalização do serviço para diferentes clientes. Estas características representam fortes atrativos para os clientes em decorrência da heterogeneidade dos objetivos das empresas quanto aos serviços de qualidade de dados, as diferentes regras de negócio e de evolução dos dados e, principalmente, as quantidades de dados distintas geradas pelas empresas.

Além disso, a utilização de SaaS acarreta em vantagens diretas sobre a visão mais tradicional na qual o software é vendido e instalado diretamente no ambiente empresarial (frequentemente sob restrições relacionadas ao número de instalações) ou computadores de usuários finais. Funcionalidades utilizadas na categoria de SaaS não exigem custos de instalação, diminuem a demanda de aquisições de hardware nas empresas, podem ser acessadas de forma mais flexível por qualquer dispositivo com acesso a Internet e são cobradas de acordo com o tipo e qualidade do serviço prestado. Atualmente, os serviços são usualmente disponibilizados utilizando como base um paradigma denominado computação em nuvem [79]. Este paradigma baseia-se na possibilidade de alocar, de maneira dinâmica, diferentes infraestruturas de hardware distribuídas para atender diferentes demandas de clientes. Além disso, o

poder computacional das infraestruturas alocadas pode ser aumentado ou diminuído ao longo do tempo, de acordo com a demanda requerida pelos clientes. Na prática, esta capacidade permite uma diminuição nos custos de utilização do serviço ao longo do tempo.

No contexto de um ambiente de nuvem, os algoritmos responsáveis pelo gerenciamento dinâmico da alocação de recursos são denominados algoritmos de provisionamento. Desse modo, os clientes de serviços baseados em computação em nuvem pagam aos provedores proporcionalmente ao poder computacional da infraestrutura alocada pelo serviço ao longo do tempo. O contrato entre provedores de serviços e os clientes dos serviços é usualmente baseado na definição de níveis de acordo de serviço (em inglês, *Service Level Agreement - SLA*). Em essência, um SLA é um contrato no qual o provedor especifica os níveis de Qualidade de Serviço (em inglês, *Quality of Service - QoS*) que um serviço deve garantir [24].

No contexto de monitoramento de qualidade de dados como um serviço, outra estratégia que pode ser adotada para a diminuição de custos consiste em utilizar algoritmos incrementais [15; 31; 86] de qualidade de dados. Na prática, estes algoritmos realizam as seguintes tarefas: i) identificam a porção da base de dados afetada por um conjunto de incrementos nos dados; ii) avaliam apenas a porção dos dados afetada pelos incrementos ocorridos; e iii) atualizam, utilizando os resultados da porção dos dados avaliada, a avaliação de qualidade da base de dados como um todo. Desse modo, a utilização de algoritmos incrementais no processo de monitoramento contínuo de qualidade de dados pode acarretar em um impacto bastante significativo na diminuição dos custos de um serviço de monitoramento de qualidade de dados.

Por fim, outra abordagem que pode ser explorada no contexto de avaliação de qualidade de dados em nuvem é o controle do tamanho dos blocos gerados por uma técnica de indexação (Seção 2.2.1), a qual visa reduzir a quantidade de comparações entre entidades, aplicada no contexto do problema de deduplicação de dados. Este controle permite que uma quantidade significativamente menor de comparações seja realizada entre as entidades da base de dados avaliada. Além disso, a geração de blocos de tamanho similares facilita a distribuição da execução das comparações entre as entidades entre várias máquinas virtuais em um ambiente de computação em nuvem.

Neste cenário, é identificada como possibilidade de pesquisa científica a investigação de

diversos aspectos relacionados à disponibilização de tarefas relacionadas a um Serviço de Monitoramento de Qualidade de Dados em nuvem (SMQD), tais como: i) proposição de uma arquitetura em alto nível para disponibilização de um SMQD; ii) proposição de estratégias para estimativa de custos atribuídos à disponibilização de um SMQD; e iii) proposição de novas abordagens para diminuição de custos relacionados à execução de algoritmos de deduplicação de dados

1.2 Escopo na Área de Qualidade de Dados

O estado da arte define um série de dimensões de qualidade de dados, ou seja, os dados podem ser avaliados de acordo com diversas perspectivas diferentes. Este trabalho se limita a investigar a dimensão de qualidade de dados denominada Duplicação [75], a qual está diretamente relacionada ao problema de deduplicação de dados, ou seja, identificar entidades duplicadas em bases de dados. A escolha da investigação do problema se deu pelo fato da complexidade intrínseca (ou seja, a quantidade de comparações entre entidades aumenta de maneira quadrática em relação ao tamanho da base de dados) associada aos algoritmos de deduplicação de dados e pelo fato de ser um problema reconhecidamente [12] importante e recorrentemente investigado no estado da arte.

1.3 Objetivos

Nesta seção, são apresentados o objetivo geral e objetivos específicos deste trabalho.

1.3.1 Objetivo Geral

O objetivo geral deste trabalho consiste em propor e avaliar abordagens que visam reduzir custos financeiros associados a execuções de tarefas de deduplicação de dados no contexto de um SMQD. Para este fim, são investigadas diversas escolhas que podem influenciar nos custos de um SMQD ao executar algoritmos de deduplicação de dados, tais como: a forma de modularização da arquitetura do serviço, a adoção de modelos matemáticos para avaliação de custos, a utilização de algoritmos incrementais de deduplicação de dados, o emprego

de algoritmos de provisionamento de recursos computacionais (os quais levam em consideração as características do contexto específico de um SMQD) e a utilização de algoritmos para controlar o tamanho dos blocos gerados na etapa de indexação em uma tarefa de deduplicação de dados. Desse modo, a hipótese geral do trabalho consiste em avaliar se a adoção de algoritmos para provisionamento de recursos computacionais em um ambiente de nuvem, de algoritmos incrementais de deduplicação de dados e de heurísticas para controlar tamanho de blocos gerados no contexto do problema de deduplicação de dados pode reduzir significativamente os custos em um SMQD.

1.3.2 Objetivos Específicos

Considerando o objetivo geral proposto, este trabalho possui os seguintes objetivos específicos:

1. Propor uma abordagem para a definição de acordos de nível de serviço (SLA) entre clientes e provedores de SMQD, levando em consideração aspectos específicos de qualidade de dados;

2. Propor um modelo matemático para a estimativa de custos relacionados à disponibilização de um SMQD;

3. Propor uma arquitetura em alto nível para um SMQD;

4. Propor e avaliar estratégias para a alocação dinâmica de recursos computacionais no contexto de um SMQD em ambiente de nuvem, levando em consideração a caracterização da complexidade das tarefas de qualidade de dados a serem executadas, possíveis restrições de tempo associadas a estas tarefas e os valores de parâmetros de algoritmos de qualidade de dados;

5. Propor e avaliar novas métricas para avaliação de algoritmos incrementais de qualidade de dados, visando avaliar aspectos não explorados pelas métricas existentes no estado da arte, tais como eficiência relativa e eficácia ponderada;

6. Propor e avaliar novos algoritmos para deduplicação incremental de dados no contexto de classificação coletiva;

7. Propor e avaliar heurísticas para controlar o tamanho de blocos produzidos no contexto do problema de deduplicação de dados, as quais visam reduzir os custos da deduplicação de dados ao limitar a quantidade de comparações a ser realizada entre entidades da base de dados avaliada.

1.4 Relevância

Primeiramente, é importante destacar que a existência de dados de baixa qualidade é responsável por bilhões de dólares de prejuízo para empresas ao redor do mundo a cada ano [5]. Além disso, a natureza e a escala dos atuais sistemas computacionais tendem a tornar cada vez mais inadequadas as abordagens clássicas relacionadas aos algoritmos de qualidade de dados [70]. Por estes motivos, esforços no sentido de diminuir custos, aumentar eficácia e/ou diminuir o tempo de execução de algoritmos de qualidade de dados são bastante importantes.

Além disso, a utilização da categoria de SaaS tornou-se uma tendência para a disponibilização de serviços na web por empresas líderes na área de tecnologia da informação. De acordo com a empresa de consultoria Gartner, estima-se que os negócios envolvendo SaaS irão movimentar bilhões de dólares no mercado brasileiro[2]. Tendo em vista que uma gama cada vez maior de diferentes tipos de negócio tem sido ofertada como um serviço e que este setor engloba uma considerável parcela das economias globalizadas, esforços no intuito de investigar arquiteturas, acordos de nível de serviço e estratégias para diminuição de custos relacionados a novos serviços são bastante relavantes para o cenário atual.

No contexto deste trabalho, o qual engloba a investigação de desafios relacionados à redução de custos de tarefas de deduplicação de dados executadas um serviço em nuvem, a relevância se deve principalmente à proposição e avaliação de novas abordagens de provisionamento dinâmico de recursos, a proposição de novos algoritmos de deduplicação incremental de dados e a proposição de heurísticas para controlar o tamanho de blocos produzidos na etapa de indexação do problema investigado. Assim, estes algoritmos podem ser futuramente estendidos, melhorados e utilizados em modelos de negócio e/ou por pesquisadores em futuras investigações científicas.

[2]http://www.valor.com.br/empresas/2834130/servico-publico-na-nuvem-vai-movimentar-us-109-bi-no-ano-diz-gartner

1.5 Metodologia

No intuito de conduzir uma pesquisa científica visando propor soluções para os problemas e desafios investigados neste trabalho, foi empregada uma metodologia que dividiu a pesquisa de doutorado conduzida em micro projetos de pesquisa. Para cada projeto de pesquisa, foi realizado um planejamento prévio que especificou uma descrição detalhada e um cronograma para as seguintes atividades: i) fundamentação teórica e justificativa de relevância do problema ou da área de levantamento bibliográfico a ser investigado; ii) investigação de livros, artigos e relatórios técnicos relacionados ao problema investigado; iii) proposição de novas abordagens, algoritmos, modelos e/ou metodologias para tratar problema investigado; iv) definição de hipóteses e planejamento de um *design* experimental visando avaliar as hipóteses de pesquisa investigadas; v) execução dos experimentos planejados; vi) formatação e discussão dos resultados obtidos na atividade anterior; vii) listagem das lições aprendidas e conclusões provenientes da condução da pesquisa; e viii) definição de trabalhos futuros.

Neste trabalho, foram investigados e estendidos trabalhos principalmente relacionados à alocação de recursos computacionais em um ambiente de computação em nuvem, técnicas de aprendizado de máquina, heurísticas e algoritmos de deduplicação incremental de dados e heurísticas para controlar o tamanho de blocos produzidos na etapa de indexação do problema de deduplicação de dados.

1.6 Principais Contribuições

A principal proposta deste trabalho consiste em reduzir custos provenientes de soluções para o problema de deduplicação de dados no contexto de um SMQD. Para tal, foram investigados diferentes modelos, algoritmos e abordagens para reduzir custos associados à disponibilização de um SMQD em nuvem. Desse modo, foram desenvolvidos:

i) uma estrutura para representar um SLA de qualidade de dados;

ii) uma arquitetura em alto nível para um SMQD;

iii) um modelo de custo para um SMQD;

iv) a formalização do problema de alocação de recursos computacionais para a execução de tarefas de deduplicação de dados no contexto de um SMQD;

v) um conjunto de algoritmos para o tratamento do problema do item iv;

vi) métricas e definições para o problema de deduplicação incremental de dados no contexto de classificação coletiva;

vii) heurísticas para o problema de deduplicação incremental de dados no contexto de classificação coletiva;

viii) heurísticas para controlar o tamanho de blocos produzidos na etapa de indexação do problema de deduplicação de dados.

Além disso, foram realizadas avaliações experimentais para validar a eficácia e eficiência dos algoritmos propostos no trabalho, os quais indicam, em resumo: i) a adequação de técnicas de aprendizado de máquina para a alocação dinâmica de recursos computacionais no contexto de um SMQD; ii) diferentes possibilidades de resultados produzidos por heurísticas para o problema de deduplicação incremental de dados, as quais usualmente apresentam uma solução mais eficiente para o problema, mas sacrificando ligeiramente resultados de eficácia; e iii) a adequação da utilização de dados provenientes da coocorrência de entidades em diversas blocagens para controlar o tamanho dos blocos gerados na etapa de indexação no problema de deduplicação de dados.

1.7 Indicadores de Pesquisa Alcançados

Até o presente momento, os seguintes indicadores de pesquisa foram alcançados:

- Publicação do artigo *A Data Quality-aware Cloud Service* [19] no Workshop de Teses e Dissertações em Banco de Dados (WTDBD 2016) do XXI Simpósio Brasileiro de Banco de Dados (2016);

- Publicação do artigo *Applying Machine Learning Techniques for Scaling Out Data Quality Algorithms in Cloud Computing Environments* [60] no Jornal Applied Intelligence (2016).

- Publicação e apresentação do artigo *A Data Quality-aware Cloud Service based on Metaheuristic and Machine Learning Provisioning Algorithms* [59] nos Proceedings do 30th ACM/SIGAPP Symposium on Applied Computing (2015), em Salamanca - Espanha.

- Publicação do capítulo de livro *Data Quality Monitoring of Cloud Databases based on Data Quality SLAs* no livro *Big-Data Analytics and Cloud Computing: Theory, Algorithms and Applications* [59], Springer International Publishing, 1st ed (2015).

- Apresentação em forma de pôster do trabalho *A Data Quality-aware Cloud Service* na 7th Herrenhausen Conference - Big Data in a Transdisciplinary Perspective (2015), em Hanover - Alemanha.

- Apresentação em forma de pôster do trabalho *Data Quality Monitoring of Cloud Databases* na 10th IEEE International eScience Conference (2015), em Guarujá - SP.

Também foram submetidos e estão em avaliação os seguintes artigos:

- Submissão do artigo *Heuristic-based Approaches for Speeding up Incremental Record Linkage* para o Journal of Systems and Software (atualmente em fase de *rebuttal*);

- Submissão do artigo *Exploiting Block Co Occurrence to Control Block Sizes for Entity Resolution* para o Journal of Heuristics.

1.8 Estrutura do Documento

O restante deste documento está organizado da seguinte forma:

Capítulo 2 - Fundamentação Teórica: aborda os conceitos básicos e a terminologia inerente ao entendimento do presente trabalho. O texto discorre sobre conceitos importantes relacionados à qualidade de dados, tais como: dimensões de qualidade de dados, deduplicação de dados e indexação de dados. Além disso, são apresentados os principais conceitos e definições relacionados a software como serviço, computação em nuvem e provisionamento dinâmico de recursos computacionais. É também apresentado o conceito de aprendizado de máquina. Por fim, são apresentados os principais conceitos relacionados à técnica de

classificação coletiva de entidades duplicadas e algoritmos incrementais de deduplicação de dados.

Capítulo 3 - Trabalhos Relacionados: neste capítulo é apresentada a revisão bibliográfica acerca dos principais trabalhos relacionados encontrados na literatura e suas respectivas propostas. Essencialmente, os trabalhos estão divididos em três grupos: trabalhos que investigam provisionamento de recursos computacionais em ambientes de computação em nuvem, trabalhos que propõem algoritmos incrementais de qualidade de dados e trabalhos que tratam o problema de controle de tamanho de blocos gerados na etapa de indexação no problema de deduplicação de dados.

Capítulo 4 - Arquitetura de um Serviço de Monitoramento de Qualidade de Dados: este capítulo engloba as contribuições do trabalho envolvendo a concepção em alto nível de um SMQD, sendo estas: i) a proposição e a descrição da arquitetura proposta para um SMQD; e ii) a formalização e descrição de um SLA de Qualidade de Dados.

Capítulo 5 - Provisionamento Dinâmico de Recursos Computacionais em um SMQD: este capítulo apresenta as contribuições do trabalho no contexto do problema de alocação dinâmica de recursos computacionais no contexto de um SMQD, englobando: i) modelo matemático de custos associados a um SMQD; ii) formalização do problema de redução de custos em um SMQD utilizando algoritmos de provisionamento de recursos computacionais; iii) algoritmos de provisionamento dinâmico de recursos, no contexto de um SMQD, baseados em heurísticas e aprendizado de máquina.

Capítulo 6 - Estratégias para Deduplicação Incremental de Dados: este capítulo apresenta as contribuições do trabalho no contexto da tarefa de deduplicação incremental de dados, são estas: i) proposição de novas métricas para avaliação de algoritmos incrementais de deduplicação de dados; e ii) a proposição de novas heurísticas no contexto de deduplicação incremental de dados.

Capítulo 7 - Estratégias para Controlar o Tamanho de Blocos: este capítulo apresenta as contribuições do trabalho no contexto da tarefa de controlar o tamanho dos blocos gerados pela indexação de entidades no problema de deduplicação de dados. Neste contexto, foram propostas heurísticas para tratar o problema no contexto da indexação empregando múltiplas chaves de bloco.

Capítulo 8 - Avaliação dos Algoritmos Propostos: apresenta detalhes relacionados ao

design experimental, ambiente e métricas utilizadas na avaliação dos algoritmos de provisionamento de recursos, de deduplicação incremental de entidades e de controle do tamanho de blocos propostos neste trabalho. Além disso, são apresentados os resultados dos experimentos e respectivas discussões. Por fim, são apresentadas as ameaças à validade dos resultados obtidos.

Capítulo 9 - Conclusões: neste capítulo são apresentadas as conclusões inerentes ao trabalho desenvolvido e as principais perspectivas de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo aborda os conceitos básicos e a terminologia inerente ao entendimento do presente trabalho. Os principais conceitos apresentados são: i) qualidade de dados; ii) dimensões de qualidade de dados; iii) deduplicação de dados; iv) software como serviço; v) computação em nuvem; vi) Big Data; vii) aprendizado de máquina; viii) classificação coletiva de entidades duplicadas; e iv) deduplicação incremental de dados. Os principais símbolos apresentados neste capítulo e no restante do trabalho são sumarizados em tabelas de símbolos apresentadas no Apêndice A.

2.1 Qualidade de Dados

O termo Qualidade de Dados pode ser interpretado em diferentes contextos: uma área de pesquisa acadêmica, uma preocupação do processo de governança dos dados ou o resultado da avaliação dos dados sobre determinada perspectiva. Do ponto de vista acadêmico, a área de Qualidade de Dados é investigada por diferentes profissionais, incluindo: estatísticos, administradores de negócios e cientistas da computação [5].

Em uma perspectiva histórica, estatísticos foram os primeiros a investigar e catalogar problemas relacionados à qualidade dos dados, a partir da proposição de uma teoria matemática para avaliar entidades duplicadas em bases de dados estatísticas, em meados de 1960. Por sua vez, pesquisadores na área de gestão de negócio focaram, na década de 80, em controlar sistemas de manufatura de dados no intuito de detectar problemas de qualidade de dados. Apenas a partir da década de 90 o problema passou a ser investigado por cientis-

tas da computação, visando principalmente avaliar quantitativamente a qualidade de dados armazenados em bancos de dados, armazéns de dados e sistemas legados [5].

Na perspectiva da avaliação dos dados propriamente ditos, a noção de qualidade de dados é essencialmente centrada no cliente ou usuário do processo de avaliação [70]. Isto se deve ao fato de que um conjunto de dados é considerado ter alta qualidade, a partir da perspectiva do cliente, caso os dados satisfaçam suas necessidades. Por este motivo, a classificação de dados como de alta ou baixa qualidade consiste em uma avaliação inerentemente subjetiva, uma vez que a qualidade de uma mesma base de dados pode ser avaliada diferentemente por diversos clientes. No entanto, independente deste fator subjetivo, é possível avaliar quantitativamente diferentes aspectos relacionados à qualidade dos dados [5; 6].

Na prática, os dados usualmente representam entidades ou medições do mundo real. Desse modo, o senso comum tende a associar o termo Qualidade de Dados à noção de acurácia dos dados, ou seja, os dados possuem supostamente uma alta qualidade caso apresentem valores correspondentes aos das entidades ou medições do mundo real que representam. Por exemplo, seja o par $\langle Dimas\ Cassimiro, 27\rangle$ uma dupla que representa o nome e a idade reais do estudante Dimas Cassimiro, respectivamente. Assim, o par $\langle Zimas\ Casimiro, 30\rangle$ pode ser considerado pouco acurado, ao passo que o par $\langle Zemas\ Cassimiro, 50\rangle$ pode ser certamente considerado como não acurado. De fato, acurácia é uma importante métrica relacionada à qualidade dos dados. No entanto, a qualidade dos dados pode também ser avaliada quantitativamente por diversas perspectivas diferentes, tais como completude, consistência, facilidade de interpretação, validade, presença de valores discrepantes, presença de entidades duplicadas, dentre outras. Estas perspectivas são denominadas no estado da arte de dimensões [7, 58, 64] de qualidade de dados.

No intuito de satisfazer seus objetivos de negócio e utilizar dados na implementação e automatização de processos, é necessário que as organizações utilizem estratégias para realizar constantes medições sobre os requisitos de qualidade dos dados. Usualmente, os requisitos de qualidade de dados são definidos com base em dimensões. Neste contexto, é necessário que sejam criadas metodologias para guiar as etapas de análise dos dados no intuito de controlar e monitorar os níveis de qualidade dos dados.

Uma dimensão de qualidade de dados representa um aspecto específico das diferentes perspectivas sobre as quais a qualidade dos dados ou esquema dos dados pode ser avaliada.

É importante notar que tanto a avaliação das dimensões de qualidade de dados quanto dos esquemas são importantes para uma manutenção correta da qualidade dos dados. Isto porque enquanto dados de baixa qualidade influenciam fortemente a qualidade dos processos de negócios, a utilização de esquemas de baixa qualidade (por exemplo, um esquema relacional desnormalizado) resulta em potenciais redundâncias e anomalias durante a ciclo de utilização dos dados.

Na prática, é definida uma série de métricas associadas a uma dimensão de qualidade de dados e, para cada métrica, um ou mais métodos de avaliação podem ser utilizados [23]. Desse modo, os seguintes aspectos de uma dimensão de qualidade de dados devem ser definidos: i) quando a medida deve ser avaliada; ii) qual porção dos dados deve ser avaliada; iii) o mecanismo que realizará a medição; e iv) a escala na qual os resultados devem ser reportados. Na prática, a opção de realizar uma distinção clara entre dimensão de qualidade de dados e as respectivas métricas associadas depende da literatura adotada [5].

Algumas das dimensões mais clássicas e conhecidas são [5; 75]:

Acurácia: a proximidade entre os valores v e v', sendo v um valor associado à uma entidade ou fenômeno do mundo real e v' um valor utilizado para representar esta entidade ou fenômeno.

Completude: o grau de abrangência, profundidade e escopo de um dado, considerando sua utilidade para uma tarefa específica.

Atualidade: o grau de atualização de um valor em relação ao seu valor correspondente no mundo real.

Volatilidade: a frequência na qual um dado varia ao longo do tempo.

Tempestividade: mede o quão atualizado é o dado para a realização de uma tarefa específica, uma vez que é possível existir um dado atual, mas já considerado obsoleto para uma tarefa específica.

Relevância: o grau de utilidade de um dado para uma tarefa específica.

Quantidade Apropriada de Dados: a quantidade ou volume dos dados apropriado no contexto de uma tarefa específica.

Acessibilidade: o grau de facilidade para acessar os dados e o grau de disponibilização dos dados.

Interpretabilidade: o grau de facilidade de compreensão dos dados pelos usuários.

Duplicação: a porcentagem de valores ou entidades duplicados de maneira indesejada em uma base de dados.

Segurança: o acesso aos dados é restrito aos usuários autorizados.

Reputação: o grau de confiança dos usuários em relação aos dados e suas fontes.

Na próxima seção, é detalhado um dos problemas mais desafiadores da área de Qualidade de Dados, o qual está relacionado à dimensão de qualidade de dados Duplicação.

2.2 Deduplicação de Dados

Um dos problemas mais desafiadores na área de qualidade de dados é denominado Deduplicação de Dados e consiste na identificação de dados (registros, objetos, etc) que se referem a uma mesma entidade no mundo real em uma base de dados. Este problema foi inicialmente investigado por estatísticos na década de 50 e é também chamado de resolução de entidades, reconciliação de entidades, correspondência de entidades, vinculação de entidades, dentre outros [12].

O problema de deduplicação de dados é especialmente desafiador por uma série de motivos: i) exige a escolha de algoritmos, funções de similaridade e limiares que, na prática, são bastante complexos de serem definidos apropriadamente; ii) seja n o tamanho de uma base de dados, a deduplicação de dados possui complexidade assintótica $O(n^2)$, uma vez que potencialmente cada entidade de uma base de dados precisa ser comparada com cada outra entidade da mesma base; iii) a falta de padronização nos esquemas dos dados pode dificultar significativamente a correta identificação das entidades duplicatas; iv) o tamanho das bases de dados pode tornar inviável a resolução serial (i.e., utilizando um único nó computacional) do problema; e v) o processo de deduplicação de dados pode necessitar ser frequentemente re-executado caso os dados sejam contínua e/ou significativamente modificados ao longo do tempo. Diante destes desafios, a literatura relacionada [12; 13; 21; 43] investigou diversas técnicas e abordagens visando diminuir o tempo necessário para a resolução do problema e aumentar a eficácia associada às saídas dos algoritmos de deduplicação de dados, ou seja, a quantidade de duplicatas corretamente identificadas por estes algoritmos.

O problema de deduplicação de dados pode ser formalmente definido da seguinte ma-

neira:

Definição 2.2.1 (Deduplicação de Dados) *Seja $D = \{r_1, r_2, \ldots, r_{|D|}\}$ o conjunto de entidades na base de dados D. O processo de deduplicação de dados visa processar D no intuito de gerar um conjunto $D_{dup} \subseteq D \times D$, tal que, para todo par $(r_i, r_j) \in D_{dup}$: $i \neq j$ e r_i e r_j representam a mesma entidade no mundo real.*

Utilizando uma estratégia básica (ou seja, comparando cada entidade da base de dados com todas as demais entidades), o número de comparações realizadas por uma tarefa de deduplicação de dados é igual a $\frac{(|D|\times(|D|-1))}{2}$.

Na Figura 2.1 (adaptada de [13]), são apresentadas as principais etapas relacionadas ao processo de deduplicação de dados. Inicialmente, é realizado um processo de padronização e limpeza nos dados das bases de dados, visando principalmente aumentar a qualidade dos dados e unificar os esquemas das bases de dados. Em seguida, é usualmente utilizada uma técnica de indexação, a qual visa diminuir a quantidade de comparações entre entidades realizada pelo processo de deduplicação. Na prática, esta etapa tem como objetivo diminuir a quantidade de comparações a serem realizadas ao comparar apenas entidades que compartilham características[1] em comum. Após a indexação, é realizada a etapa de comparações entre entidades. Esta etapa é usualmente realizada por meio da utilização de funções de similaridade [12] (e.g., Jaccard, Distância de Edição e JaroWinkler) e limiares pré-definidos, ou seja, valores utilizados para atestar a duplicação de entidades ou a existência de uma semelhança significativa entre entidades a partir do resultado das funções de similaridade. Prosseguindo, é realizada a etapa de classificação, a qual utiliza os resultados da etapa anterior para classificar pares de entidades em: não duplicatas, duplicatas ou potenciais duplicatas. A etapa de classificação pode ser realizada por meio da utilização ou combinação das seguintes técnicas: i) comparação direta com um limiar de similaridade [12]; ii) utilização de técnicas de agrupamento automático [30]; iii) utilização de regras de classificação de entidades [86]; ou iv) utilização de algoritmos de aprendizado de máquina [72].

Após a execução do processo de deduplicação de entidades, a lista de pares de entidades classificados como potenciais duplicatas precisa passar por um processo denominado revisão manual. Este processo é realizado a partir da intervenção de uma especialista no domínio dos

[1] partes de valores de atributos ou de esquemas

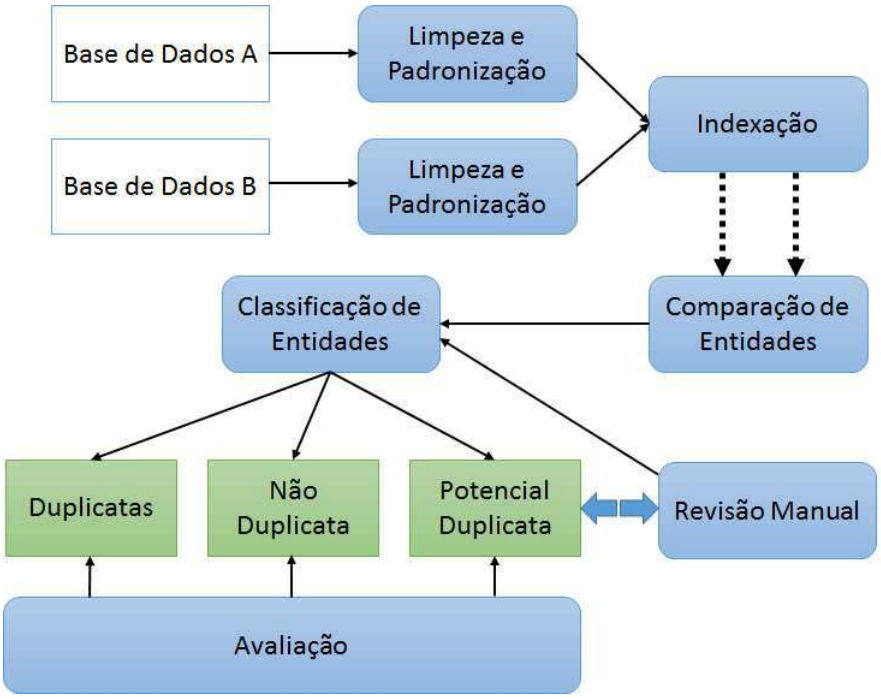

Figura 2.1: Etapas do processo de deduplicação de dados (adaptada de [15]).

dados, visando avaliar cada par dessa lista no intuito de reclassificá-lo como não duplicata ou duplicata. Na prática, por depender do custo e tempo de trabalho de um ser humano, este processo é bastante lento e custoso. Por esta razão, os algoritmos de deduplicação de dados devem tentar minimizar a quantidade de pares de entidades classificadas como potenciais duplicatas.

2.2.1 Indexação

Como explicado na seção anterior, ao lidar com bases de dados muito volumosas, o número de comparações a serem realizadas em um processo de deduplicação de dados pode tornar o tempo de execução necessário para a completude da tarefa bastante alto, podendo chegar a dias ou semanas de execução. Por este motivo, são comumente utilizadas técnicas de indexação [12; 13] para limitar a quantidade de comparações entre as entidades da base de dados.

Por exemplo, uma das técnicas mais conhecidas de indexação é denominada blocagem padrão [13]. Esta técnica se baseia na associação de uma chave de bloco para cada entidade na base de dados. Exemplos clássicos de chave de bloco são: o valor de um atributo cate-

górico da entidade, as n primeiras letras do valor de algum atributo ou o Soundex [12] do valor de um atributo. Desse modo, a blocagem padrão consiste em separar as entidades em blocos disjuntos, de acordo com o valor correspondente da chave de bloco de cada entidade, e limitar a comparação entre entidades que pertencem ao mesmo bloco. Esta ideia pode ser formalizada da seguinte maneira:

Definição 2.2.2 (Blocagem Padrão) *Sejam $D = \{r_1, r_2, \ldots, r_{|D|}\}$ um conjunto de entidades na base de dados D, k uma função para o cálculo da chave de bloco de uma entidade e $k(r)$ a chave de bloco de uma entidade $r \in D$. O objetivo do processo de blocagem padrão é gerar um conjunto de blocos $B = \{b_1, b_2, \ldots, b_{|B|}\}$, tal que cada $b \in B$ é um conjunto de entidades pertences a D. Seja $block(r) \in B$ o bloco de uma entidade $r \in D$, então: $\forall r \in D \exists b \in B(r \in b) \ \wedge \ \forall r_1, r_2 \in D(k(r_1) = k(r_2) \Rightarrow block(r_1) = block(r_2)) \ \wedge \ \bigcap_{b \in B} b = \emptyset$.*

Caso as entidades sejam divididas de maneira uniforme entre os $|B|$ blocos, a técnica de blocagem padrão reduz a complexidade de um problema de deduplicação de dados para $\frac{|D|}{2} \times (\frac{|D|}{|B|} - 1)$ comparações entre entidades [13]. Apesar da aparente simplicidade, a técnica de blocagem padrão apresentou uma eficácia superior à diversas outras técnicas de indexação nos experimentos realizados por um recente survey [13] no estado da arte. Exemplos de outras técnicas de indexação disponíveis no estado da arte são a janela deslizante [21] e o agrupamento canopy [12].

2.2.2 Técnicas de Classificação de Entidades Duplicadas

Considerando o fluxograma mostrado na Figura 2.1, uma das etapas (Classificação de Entidades) realizadas pelo processo de deduplicação de dados consiste em classificar os pares de entidades em duplicatas, não duplicatas ou potenciais duplicatas. Para a realização desta etapa, diferentes técnicas de classificação podem ser utilizadas.

As técnicas de classificação baseadas em comparação com limiar utilizam intervalos entre limiares específicos para classificar os pares de entidades de acordo com a similaridade computada entre as mesmas. Um possível intervalo de limiares a ser utilizado por uma técnica de classificação deste tipo é mostrado na Eq. (2.1).

$$classificacao(r_1, r_2) = \begin{cases} duplicata(r_1, r_2) & \textbf{se } sim(r_1, r_2) \geq 0.85 \\ potencial\_duplicata(r_1, r_2) & \textbf{se } 0.7 < sim(r_1, r_2) < 0.85 \\ nao\_duplicata(r_1, r_2) & \textbf{se } sim(r_1, r_2) \leq 0.7 \end{cases} \tag{2.1}$$

As técnicas de classificação baseadas em regras [86] utilizam regras pré-definidas, usualmente especificadas por especialistas no domínio dos dados, para classificar as entidades. Por exemplo, sejam r_1 e r_2 entidades em uma base de dados, uma possível regra para classificação de entidades é: se $(sim(name(r_1), name(r_2)) \geq 0.8 \wedge zip(r_1) = zip(r_2) \wedge phone(r_1) = phone(r_2))$ então $duplicata(r_1, r_2)$.

Técnicas de classificação baseadas em agrupamento automático usualmente mapeiam as entidades de uma base de dados para um grafo de similaridade não direcionado, no qual os vértices representam entidades e os pesos das arestas são computados como a similaridade entre as entidades, e aplicam técnicas de agrupamento automático [88] de entidades sobre este grafo. Ao fim deste processo, cada agrupamento representa os pares de entidades duplicadas.

Por fim, técnicas de classificação baseadas em aprendizado de máquina [56] utilizam uma base de treino inicial, contendo pares de entidades consideradas duplicadas e não duplicadas, e com base nestes exemplos utiliza técnicas de aprendizado de máquina para classificar novos pares de entidades da mesma base de dados.

2.2.3 Relevância do Problema de Deduplicação de Dados

O problema de deduplicação de dados foi bastante investigado nas últimas décadas tanto em nível acadêmico quanto na indústria de software. Isto porque, a nova escala de dados processada atualmente pelos sistemas demandou novas técnicas e algoritmos para resolver o problema manipulando bases de dados muito mais volumosas. Além disso, é reconhecido que a existência de entidades duplicadas em uma base de dados pode acarretar em uma série de problemas em diferentes contextos reais [12].

No contexto de pesquisa de censos populacionais, a deduplicação de dados tem uma série de aplicações distintas e importantes: i) fazer a junção, eliminando-se as duplicatas, de dados

provenientes de censos passados com dados de um novo censo; ii) eliminar as eventuais duplicatas provenientes da junção de dados de censos realizados por diferentes empresas; iii) eliminar as eventuais duplicatas provenientes da junção de dados de censos realizados por diferentes países. Todas estas tarefas tem o potencial de reduzir drasticamente o custo associado à análise de grandes volumes de dados provenientes de censos, mas dependem da aplicação de um processo eficiente e eficaz de deduplicação de dados [12].

Por sua vez, o domínio da saúde foi um dos pioneiros a utilizar de maneira efetiva por muitas décadas o processo de deduplicação de dados [12]. Similarmente ao que ocorre com os censos, a deduplicação de dados permite uma efetiva junção de dados de diferentes fontes e facilita a junção de dados coletados anteriormente com novos dados coletados. Outro exemplo se dá em aplicações de segurança [12], nas quais são combinadas técnicas de deduplicação, biometria, processamento de linguagem natural e reconhecimento automático de imagens no intuito de detectar padrões entre vários bancos de dados contendo informações de indivíduos suspeitos de praticar e/ou organizar atos terroristas.

Por fim, técnicas de deduplicação são também essenciais em outros contextos práticos, tais como a prevenção e detecção de fraudes, na validação de endereços de clientes e na indexação de dados provenientes de bancos de dados bibliográficos.

2.3 Classificação Coletiva para Deduplicaçação de Dados

Além das técnicas de classificação de entidades duplicadas baseadas em limiar, aprendizado de máquina e regras pré-definidas (discutidas na Seção 2.2.2), uma outra estratégia existente para classificação de entidades duplicadas é denominada classificação coletiva [12]. A ideia básica dessa abordagem consiste em classificar entidades duplicadas com base na similaridade entre grupos de entidades, ao invés de considerar apenas a similaridade entre pares de entidades. A técnica de classificação coletiva será mais detalhadamente abordada neste capítulo, uma vez que este trabalho se propõe a investigar novas abordagens de solução para o problema de deduplicação de dados no contexto de classificação coletiva.

O processo geral realizado pela classificação coletiva é mostrado na Figura 2.2. A primeira etapa desta técnica consiste em realizar uma junção de similaridade (tarefa que visa identificar os pares de entidades em uma base de dados com similaridade acima de um

valor de limiar pré estabelecido), preferencialmente utilizando um algoritmo eficiente [16; 83] para este fim, no intuito de gerar um grafo de similaridade representando as entidades da base de dados. Em seguida, é aplicada uma técnica de agrupamento automático de entidades sobre o grafo de similaridade gerado. Ao fim deste processo, cada agrupamento produzido representa as duplicatas identificadas, ou seja, o resultado do agrupamento representa a saída do processo de deduplicação. Em recentes avaliações conduzidas pelo estado da arte [12; 31; 86], a técnica de classificação coletiva apresentou resultados de eficácia bastante elevados. Os detalhes da notação adotada para a representação de grafos de similaridade e agrupamentos no contexto de classificação coletiva são apresentados na próxima seção.

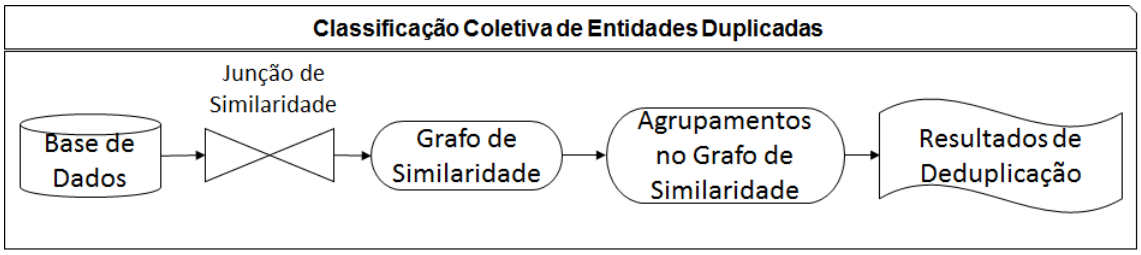

Figura 2.2: Etapas do processo de classificação coletiva de entidades duplicadas.

2.3.1 Grafo de Similaridade

Nesta seção, é apresentada a notação adotada para representar as entidades em uma base de dados na forma de um grafo de similaridade, assim como os agrupamentos gerados a partir de um processo de classificação coletiva. Seja $D = \{r_1, r_2, \ldots, r_{|D|}\}$ um conjunto de entidades. Após realizar o processo de junção de similaridade sobre as entidades em D, é criado um grafo de similaridade $G(V, E)$ no qual cada entidade em D é mapeada para um vértice v em V e é adicionada uma aresta $e(v, v')$ em E para cada par de vértices distintos (v, v'), tal que $v, v' \in V$ e o valor de similaridade $sim(v, v')$ seja maior ou igual a um limiar θ.

Um algoritmo de agrupamento automático tem como objetivo agrupar os vértices em V em um conjunto de agrupamentos disjuntos, ou seja, $\mathcal{L}_G = \{C_1, C_2, \ldots, C_{|\mathcal{L}_G|}\}$, tal que $\forall\langle C \in \mathcal{L}_G\rangle(C = \{v_1, v_2, \ldots\ v_{|C|}\})$, $\forall\langle v \in V\rangle\exists\langle C \in \mathcal{L}_G\rangle(v \in C)$ e $\bigcap_{C\in\mathcal{L}_G} C = \emptyset$. O processo realizado por um algoritmo de agrupamento automático [33] é usualmente guiado pela minimização de uma função objetivo, aplicação de técnicas de grafo, ordenação de

pesos de arestas ou graus dos vértices, distância entre vértices e centróides ou medóides, ou técnicas de fluxo.

No contexto do problema de classificação coletiva de entidades duplicadas, é essencial que o algoritmo de agrupamento automático apresente as seguintes características: i) produza como saída um conjunto de agrupamentos disjuntos, pois a geração de agrupamentos com sobreposição é claramente um resultado indesejado no contexto do problema; e ii) seja classificado como um algoritmo não supervisionado, ou seja, não requeira o conhecimento *a priori* da quantidade de agrupamentos a serem produzidos, uma vez que esta informação não está disponível em um problema de deduplicação de dados. Na próxima seção, é apresentada a utilização da técnica de classificação coletiva no contexto do problema de deduplicação incremental de dados.

2.3.2 Deduplicação Incremental de Dados

Em um cenário em que frequentes alterações afetam as bases de dados, os resultados de processos de deduplicação anteriores tornam-se rapidamente obsoletos à medida em que os dados são alterados ao longo do tempo. Sendo assim, ao lidar com bases de dados volumosas, torna-se bastante ineficiente e custoso a re-execução do processo de deduplicação de dados como um todo após cada alteração realizada na base de dados. No intuito de minimizar este problema, podem ser aplicadas técnicas incrementais de deduplicação de dados, as quais visam realizar o processo de deduplicação apenas sobre a porção da base de dados que foi afetada pelas alterações na mesma.

Recentes trabalhos [30; 15; 86] do estado da arte apresentaram diversas contribuições no contexto do problema de Deduplicação Incremental de Dados (DID) utilizando classificação coletiva e propuseram diversas abordagens de solução para este problema. Na Figura 2.3, é apresentado o fluxo de tarefas relacionado ao processo de deduplicação incremental no contexto de classificação coletiva. Este processo é realizado cada vez que alterações afetam os dados contidos na base de dados. Inicialmente, os incrementos (ou seja, operações de inserção, remoção e atualização) nos dados são aplicados à base de dados. Em seguida, estas modificações são refletidas no grafo de similaridade que representa a base de dados. Prosseguindo, é aplicada uma estratégia para selecionar um subconjunto dos agrupamentos que foram afetados pelos incrementos e este subconjunto é processado por um algoritmo

de DID para ser atualizado considerando as alterações provenientes dos incrementos nos dados. Após o término desta tarefa, os resultados do processo de deduplicação, associados aos agrupamentos gerados, são também atualizados.

Alternativamente, uma técnica de indexação, tal como blocagem padrão ou janela deslizante, pode ser aplicada à base de dados antes da operação de junção de similaridade realizada pelo processo de classificação coletiva. Por exemplo, caso seja aplicada uma técnica de blocagem, um processo independente de classificação coletiva (Figura 2.2) e DID (Figura 2.3) é realizado para cada bloco gerado pela técnica de indexação aplicada. Por conseguinte, os incrementos nos dados devem ser sempre direcionados aos blocos correspondentes das entidades afetadas pelas modificações.

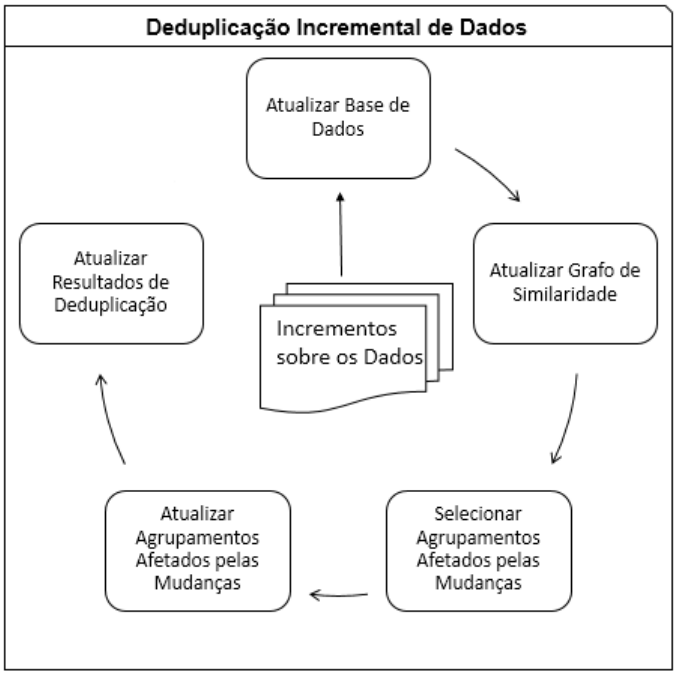

Figura 2.3: Etapas do processo de deduplicação incremental de dados no contexto de classificação coletiva.

Um conjunto de incrementos na base de dados D é denotado por ΔD. Uma vez que estes incrementos também afetam o grafo de similaridade G (proveniente da operação de junção de similaridade sobre D), o conjunto de incrementos sobre G é denotado por ΔG. Seja $\{\Delta G_0, \Delta G_2, \ldots, \Delta G_{n-1}\}$ o conjunto de n incrementos sobre o grafo de similaridade G. Cada $\Delta G \in \{\Delta G_0, \Delta G_2, \ldots, \Delta G_{n-1}\}$ é composto por um conjunto de operações individuais, denotado por $\Delta G = \{\delta_1, \delta_2, \ldots, \delta_{|\Delta G|}\}$. Por sua vez, cada $\delta \in \Delta G$ representa uma

operação individual de inserção, remoção ou atualização de entidade.

Na Figura 2.4, é apresentado um exemplo de incrementos sobre um grafo de similaridade. Nesta figura, é mostrado um grafo de similaridade $G = (V, E)$, tal que $V = \{v_1, v_2, \ldots, v_{23}\}$ e $\theta = 0.7$. São também mostrados os incrementos nos dados de G representados por $\Delta G = \{\delta_1, \delta_2, \delta_3\}$. Por sua vez, na Figura 2.5 é apresentado o resultado dos incrementos (ΔG) nos dados sobre o grafo de similaridade (G) mostrado na Figura 2.4. A partir do estágio exibido na Figura 2.5, os desafios associados a um método de DID consistem na realização das seguintes tarefas: i) utilizar um método apropriado para selecionar um subconjunto dos agrupamentos em $\mathcal{L}_G$ que foram afetados por ΔG; ii) atualizar os agrupamentos selecionados na etapa anterior no intuito de gerar agrupamentos de alta qualidade; iii) utilizar a evidência proveniente dos incrementos nos dados para corrigir erros produzidos em agrupamentos anteriores; e iv) executar este processo mais rápido do que um algoritmo de deduplicação que processe todos os agrupamentos em $\mathcal{L}_G$.

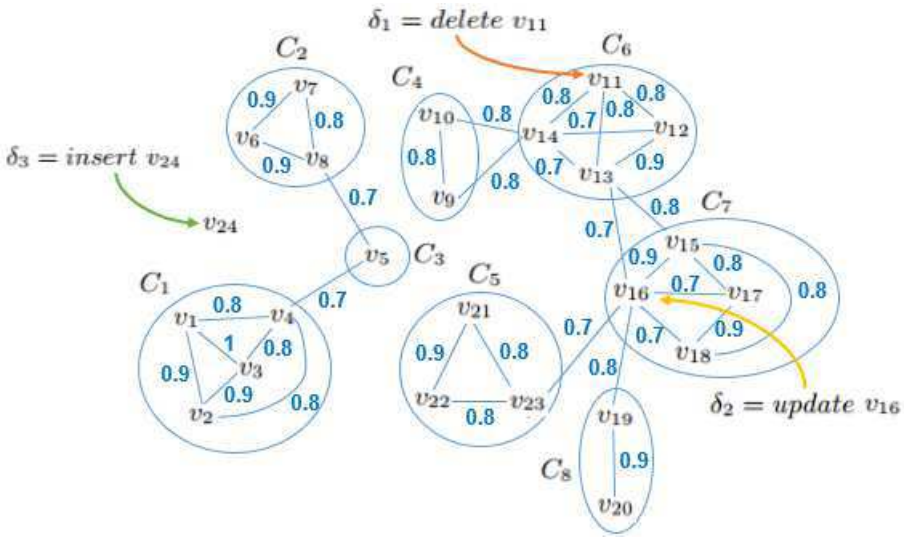

Figura 2.4: Um grafo de similaridade G e um conjunto de incrementos de dados a serem realizados sobre G.

Na prática, uma possível abordagem para guiar as tarefas ii e iii (anteriormente mencionadas) consiste em minimizar uma função objetivo (o) (ou seja, uma penalidade), a qual recebe como entrada os agrupamentos em $\mathcal{L}_G$. No intuito de avaliar $o(\mathcal{L}_G)$, diferentes abordagens [6, 18] podem ser empregadas. Por exemplo, a função objetivo denominada *Correlation Clustering* (CC) é mostrada na Eq. (2.2). Esta equação agrega a soma dos pesos das

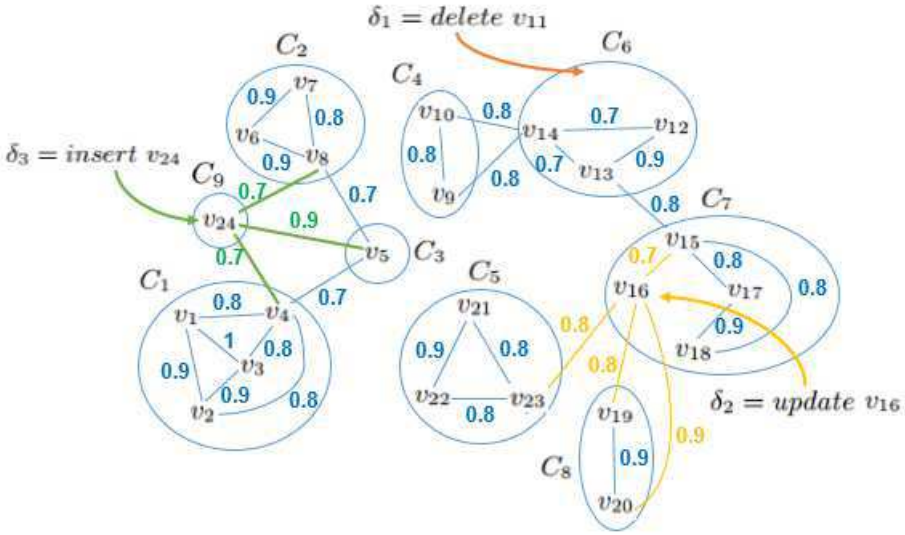

Figura 2.5: Resultado da aplicação de um conjunto de incrementos de dados sobre o grafo de similaridade G.

arestas *inter cluster* (i.e., que conectam vértices pertencentes a agrupamentos distintos) com a soma dos complementos dos pesos das arestas *intra cluster* (i.e., que conectam vértices pertencentes ao mesmo agrupamento).

$$CC(\mathcal{L}_G) = \sum_{C \in \mathcal{L}_G, r, r' \in C} (1 - sim(r, r')) \; + \sum_{C, C' \in \mathcal{L}_G, C \neq C', r \in C, r' \in C'} sim(r, r') \tag{2.2}$$

Uma das tarefas relacionadas ao processo de DID consiste em selecionar um subconjunto de agrupamentos a ser reprocessado por um método de DID, dado um conjunto ΔG de incrementos sobre G. Neste trabalho, este conjunto é denominado de *Componente de Cobertura*. O método utilizado para selecionar o componente de cobertura é denotado por $\dot{T}$ e o conjunto resultante é denotado por $\dot{T}(\Delta G)$. Em [30], os autores propõem três métodos diferentes para selecionar um componente de cobertura: Componente Conectado (Connected Component, denotado por $T(\Delta G)$), Componente Estritamente Conectado (Monotone Connected Component, denotado por $\hat{T}(\Delta G)$) e Componente Diretamente Conectado (Directly Connected Component, denotado por $\bar{T}(\Delta G)$). Dentre estes métodos, $T(\Delta G)$ é o mais abrangente (ou seja, tende a selecionar um subgrafo maior do que os outros dois métodos), ao passo que $\bar{T}(\Delta G)$ é o método mais restritivo (ou seja, tende a selecionar um subgrafo menor do que os outros dois métodos). Uma vez que, visando limitar a quantidade de agrupamentos selecionados, a abordagem do estado da arte (algoritmo $Greedy$)

emprega o método $\bar{T}(\Delta G)$, as heurísticas propostas neste trabalho (Capítulo 6) também empregam este método. A definição [30] do método Componente Diretamente Conectado ($\bar{T}(\Delta G)$) é apresentada a seguir.

Definição 2.3.1 (Componente Diretamente Conectado) *Sejam G um grafo de similaridade, $\mathcal{L}_G$ um agrupamento em G e ΔG um conjunto de incrementos sobre G. O componente diretamente conectado de ΔG, denotado por $\bar{T}(\Delta G)$, é definido da seguinte maneira:*

- *Para cada vértice inserido $v \in \Delta G$, $\bar{T}(\Delta G)$ contém v e os agrupamentos conectados (i.e., agrupamentos que estão conectados a v por pelo menos uma aresta) a v em $\mathcal{L}_G$;*
- *Para cada vértice removido $v \in \Delta G$, $\bar{T}(\Delta G)$ contém o agrupamento de v em $\mathcal{L}_G$, mas não contém nem v nem as arestas de v;*
- *Para cada aresta $e \in \Delta G$ que teve seu peso aumentado, se e é uma aresta entre os agrupamentos $C_1, C_2 \in \mathcal{L}_G$, $\bar{T}(\Delta G)$ contém C_1 e C_2;*
- *Para cada aresta $e \in \Delta G$ que teve seu peso diminuído, se e é uma aresta entre os vértices de um mesmo agrupamento $C \in \mathcal{L}_G$, $\bar{T}(\Delta G)$ contém C.*

Seguindo [30], as seguintes definições relacionadas ao problema de DID no contexto de classificação coletiva são apresentadas.

Definição 2.3.2 (Deduplicação Incremental) *Sejam D um conjunto de entidades, ΔD um conjunto de incrementos sobre D e $\mathcal{L}_D$ um agrupamento das entidades em D. A deduplicação incremental visa agrupar as entidades em $D + \Delta D$ (i.e., o resultado da aplicação de ΔD sobre D) baseando-se em $\mathcal{L}_D$. O método de deduplicação incremental é denotado por f e os resultados da deduplicação incremental por $f(D, \Delta D, \mathcal{L}_D)$.*

Quando a penalidade associada à $o(f(D, \Delta D, \mathcal{L}_D))$ é mínima, este resultado é chamado de deduplicação incremental ótima.

Definição 2.3.3 (Deduplicação Incremental Ótima) *Seja $\mathcal{L}_G^{opt}$ o agrupamento ótimo sobre G. Um método de deduplicação incremental f é considerado ótimo se, para cada G, ΔG, e $\mathcal{L}_G^{opt}$, o resultado f(G, ΔG, $\mathcal{L}_G$) é o agrupamento ótimo sobre $G + \Delta G$, i.e, $\mathcal{L}_{G+\Delta G}^{opt}$.*

Além de tentar minimizar uma função objetivo, outro objetivo crucial de um método de DID está relacionado com sua eficiência. É também esperado que o processo realizado por um método de DID f seja realizado muito mais rápido do que um método de deduplicação não incremental (denotado por F). Portanto, é possível resumir os objetivos de um método de DID da seguinte maneira:

Objetivos de Eficácia: i) para um número qualquer de processos de DID sobre G, f deve maximizar: $o(F(G, \Delta G, \mathcal{L}_G)) - o(f(G, \Delta G, \mathcal{L}_G))$; e ii) o método de DID f deve ser capaz de corrigir erros existentes em agrupamentos anteriores utilizando evidências provenientes dos incrementos nos dados.

Objetivo de Eficiência: $f(G, \Delta G, \mathcal{L}_G)$ deve ser completado muito mais rápido do que $F(G, \Delta \mathrm{G})$ (se $|G| \gg |\Delta G|$), ou seja, caso o tamanho do incremento seja muito menor do que o tamanho da base de dados, é esperado que um algoritmo incremental seja muito mais eficiente do que um algoritmo que processe a base de dados como um todo.

A principal abordagem do estado da arte para o problema de DID (algoritmo $Greedy$ [30]) é apresentada no Algoritmo 1. O algoritmo $Greedy$ inicialmente empilha cada agrupamento em $\bar{T}(\Delta G)$ em uma pilha de agrupamentos denotada por $\mathbf{Q}^c$ (linhas 2-4). Em seguida, i) um agrupamento C é desempilhado de $\mathbf{Q}^c$ e o algoritmo tenta realizar operações de MERGE (linha 9), SPLIT (linha 11) ou MOVE (linha 13) utilizando C, desde que estas operações produzam uma melhoria na qualidade dos agrupamentos, ou seja, diminuam o valor calculado por uma função objetivo; e ii) caso uma destas operações seja realizada, então C e (na maioria dos casos) os agrupamentos afetados pela operação executada são empilhados em $\mathbf{Q}^c$. Este processo é repetido até que $\mathbf{Q}^c$ esteja vazia. Detalhes relacionados à semântica das operações realizadas pelo algoritmo $Greedy$ são apresentados no Apêndice B.

2.3.3 Algoritmos de Agrupamento Automático em Grafos

Nesta seção, são apresentados três algoritmos de agrupamento automático em grafos [2; 34; 33] bastante eficientes que são considerados estado da arte. Estes algoritmos de agrupamento automático foram anteriormente avaliados em [33], processando o agrupamento de bases de dados como um todo, e apresentaram resultados bastante satisfatórios, considerando tanto eficiência quanto eficácia. Neste trabalho, estes algoritmos são utilizados no contexto de DID para: i) substituir a utilização de uma abordagem baseada em minimização de função de

Algoritmo 1: Algoritmo Greedy

Input : $G(V, E)$: grafo de similaridade original

ΔG: um conjunto de incrementos sobre G

$\mathcal{L}_G$: um agrupamento em G

Output: novo agrupamento em $\mathcal{L}_{G+\Delta G}$

```
1  begin
2      Q^c ← ∅
3      G' ← T̄(ΔG)
4      Put each cluster in G' to Q^c
5      while Q^c ≠ ∅ do
6          dequeue C ∈ Q^c
7          changed ← false
8          // operations return true if they change the
               clustering
9          changed ← MERGE(C, G + ΔG, L_G, Q^c)
10         if (not changed) then
11             changed ← SPLIT(C, G + ΔG, L_G, Q^c)
12         if (not changed) then
13             changed ← MOVE(C, G + ΔG, L_G, Q^c)
14     return L_G
```

penalidade; ou ii) re-agrupar os vértices de um componente de cobertura antes da aplicação de uma abordagem de DID baseada em minimização de função de penalidade.

O algoritmo *CENTER* [34] primeiramente cria uma lista de pares de vértices, ordenados de maneira decrescente pelo valor de similaridade entre os vértices. O algoritmo então realiza o agrupamento dos vértices utilizando uma única passagem pela lista gerada. Na primeira vez em que um vértice u é escaneado, o mesmo é marcado como o centro do agrupamento. Então, todo vértice v que é similar a u (i.e., existe uma aresta (u, v) no grafo de similaridade) é atribuído ao agrupamento do vértice u e não é considerado novamente [33].

Por sua vez, o algoritmo *MERGE-CENTER* [33] é uma extensão do algoritmo *CENTER* que

realiza a união de quaisquer dois agrupamentos C_1 e C_2, durante a execução do algoritmo CENTER, sempre que existir um vértice w, tal que w está conectado aos vértices u e v, u é o centro de C_1 e v é o centro de C_2. Na Figura 2.6 (extraída de [27]) são apresentados exemplos de funcionamento dos algoritmos $Center$ e $Merge\text{-}Center$.

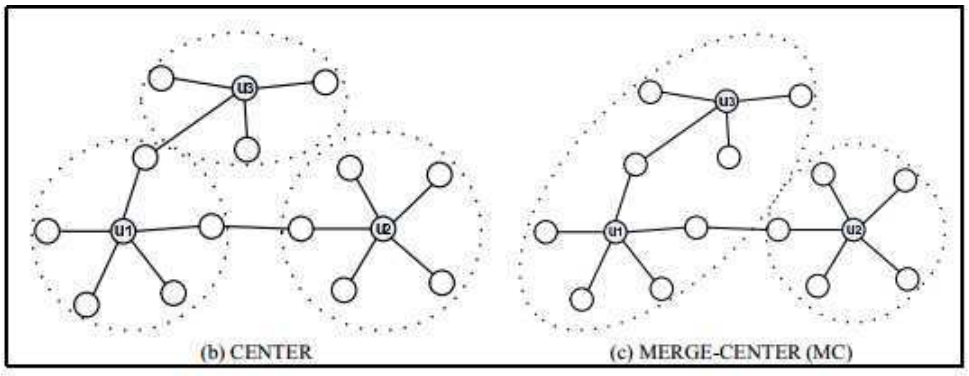

Figura 2.6: Exemplo de funcionamento dos algoritmos (a) Center e (b) Merge-Center (extraída de [33]).

Por fim, o algoritmo STAR [2] visa gerar subgrafos densos em forma de estrela. Este algoritmo realiza os seguintes passos [33]:

a. Cada vértice $v \in G(V, E)$ é inicialmente não marcado;

b. Calcule o grau do de cada vértice $v \in V$;

c. O vértice não marcado com o maior grau é considerado o centro da estrela, e é construído um agrupamento a partir do centro da estrela e os vértices conectados ao centro. Em seguida, cada vértice é marcado no agrupamento recém construído.

d. Repita o passo **c** até que todos os vértices estejam marcados.

Note que o algoritmo STAR pode gerar agrupamentos com vértices sobrepostos. Uma vez que esta é uma propriedade indesejável para um método de DID, neste trabalho é empregada uma ligeira modificação do passo **c** do algoritmo STAR. Ao invés de construir um agrupamento a partir do centro da estrela e os vértices conectados ao centro, o agrupamento é construído a partir do centro e os vértices conectados ao centro que ainda não foram marcados. Na Figura 2.7 é ilustrado um exemplo de agrupamento em forma de estrela produzido pelo algoritmo $Star$.

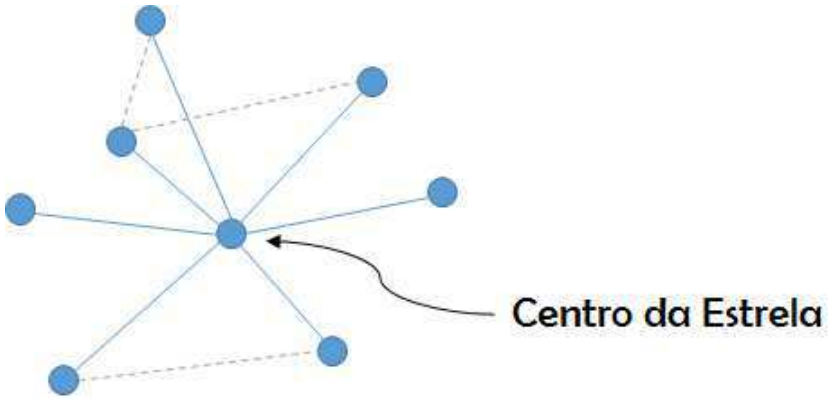

Figura 2.7: Exemplo de agrupamento em forma de estrela produzido pelo algoritmo Star [2].

Prosseguindo a explicação dos conceitos e terminologias, nas seções seguintes são apresentados os conceitos de Software como Serviço, Computação em Nuvem e Aprendizado de Máquina.

2.4 Software como Serviço

O Serviço de Monitoramento de Qualidade de Dados proposto neste trabalho se enquadra na categoria de *Software as a Service* (SaaS), a qual engloba uma abordagem na qual os clientes podem utilizar serviços e aplicações disponíveis na Internet. Neste cenário, as aplicações são armazenadas em nuvem, ou seja, sua localização física e detalhes técnicos sobre as tecnologias utilizadas nas aplicações são transparentes para os clientes do serviço. Este modelo de negócio acarreta em diversas vantagens, dentre as quais é importante destacar: a utilização de serviços sobre demanda, a diminuição das necessidades de utilização de um hardware robusto pelos clientes, a possibilidade de abstrair detalhes técnicos dos clientes do serviço e a facilidade na manutenção, evolução e atualizações no serviço disponibilizado.

Produtos e serviços disponíveis na categoria de SaaS incluem uma vasta área de aplicacões, tais como: relacionamento com o cliente, gerenciamento de recursos humanos, armazenamento de dados, aplicações de finança e cadeia de suprimentos. Funcionalidades utilizadas na categoria de SaaS não exigem custos de instalação, diminuem a demanda de aquisição de hardware nas empresas, podem ser acessadas de forma mais flexível por qualquer dispositivo com acesso a Internet e são cobradas de acordo com o tipo e a qualidade do serviço prestado. Por estes motivos, a utilização de SaaS tornou-se uma tendência para a

disponibilização de serviços sobre a web por empresas líderes na área de tecnologia da informação. Google, Twitter, Facebook e Flickr são exemplos de empresas que disponibilizam serviços via SaaS. Na próxima seção, o paradigma de Computação em Nuvem, usualmente empregado na implementação de *Software as a Service*, é explicado de maneira mais detalhada.

2.5 Computação em Nuvem

Atualmente, é bastante comum que a disponibilização de software como serviço utilize tecnologias baseadas em computação em nuvem. A computação em nuvem recentemente emergiu como uma plataforma de computação com foco em alta confiabilidade, ubiquidade e disponibilidade [3], principalmente no intuito de disponibilizar soluções computacionais sob demanda [73] e simplificar os processos relacionados ao provisionamento de infraestruturas computacionais, utilização de hardware sob demanda e desenvolvimento de software. Estas vantagens estão relacionadas à capacidade de escalabilidade de recursos computacionais associada às tecnologias de computação em nuvem. Dessa maneira, provedores de serviço podem ampliar (em inglês, *scale up*) ou reduzir (em inglês, *scale down*) os recursos computacionais de acordo com as flutuações nas demandas ao longo do tempo. Desse modo, o modelo de negócio baseado em computação em nuvem está bastante relacionado ao conceito de pagamento por utilização (em inglês, *pay per use*) de recursos. Uma vez que as empresas buscam sempre reduzir o custo total de propriedade de componentes de TI, a computação em nuvem provê uma mudança importante na forma como as infraestruturas são montadas, configuradas e gerenciadas.

Na prática, ao utilizar serviços em nuvem, gerentes de negócio podem usufruir das vantagens relacionadas à economia de custos provenientes da manutenção da infraestrutura de tecnologia da informação e das tarefas de negócio que são realizadas por um provedor de serviço em nuvem [3]. Atualmente, grandes empresas do mercado de software (tais como Amazon e Google) disponibilizam serviços baseados em computação em nuvem e investem em tecnologias relacionadas. Além disso, a computação em nuvem revolucionou o modelo tradicional do mercado de software a partir da ideia de provisionamento de recursos, a qual permite que recursos computacionais virtualizados possam ser criados, ampliados, reduzidos

e descartados sob demanda. Esta capacidade torna a computação em nuvem bastante promissora para o suporte no processamento de bases de dados Big Data, uma vez que para atingir o potencial de aplicações Big Data é necessário adotar abordagens capazes de processar uma elevada quantidade de dados em tempo hábil para o problema em questão. Desse modo, é possível alocar recursos virtualizados sob demanda para o processamento de Big Data dependendo das demandas relacionadas às restrições de tempo sobre o processamento dos dados. Por fim, é possível reduzir os custos de processamento e, por conseguinte, oferecer serviços em nuvem melhores, mais baratos e mais confiáveis [3].

A arquitetura em alto nível relacionada aos tipos de serviços prestados por modelos de negócio baseados em computação em nuvem é mostrada na Figura 2.8. A arquitetura apresenta quatro níveis organizados em uma pilha: camada de cliente, camada de serviço, camada de plataforma e camada de infraestrutura. Com base nesta organização, provedores de serviço implantam serviços na camada de serviço, em um ambiente de computação em nuvem, os quais podem ser consumidos por clientes ou outras aplicações na camada de cliente. Por sua vez, as regras para o consumo do serviço são formalizadas por meio de um contrato de níveis de serviço (em inglês, *Service Level Agreement - SLA*), o qual representa um contrato eletrônico que visa especificar as garantias do serviço prestado pelo provedor. Mais especificamente, o SLA consiste em uma coleção contratual de cláusulas entre um provedor de serviço e o cliente do serviço na qual são detalhadas a natureza, qualidade e o escopo do serviço a ser contratado [77]. Por fim, um SLA pode também especificar a porcentagem de violações de qualidade de serviço que são toleradas, entre um intervalo de tempo, sem que o provedor receba uma penalidade financeira [24]. Desse modo, algoritmos de provisionamento de recursos em ambientes de computação em nuvem são usualmente utilizados no intuito de minimizar custos do serviço ao alocar a quantidade mínima de recursos computacionais necessária para atender as demandas especificadas no SLA do cliente, evitando assim, que penalidades sejam aplicadas ao provedor do serviço.

Ao consumir serviços no nível de SaaS, os clientes não possuem controle sobre as configurações de hardware e software que suportam a implementação do serviço. Desse modo, a interação entre os clientes e os provedores de serviço se limita à interação com a interface e ao envio de valores de parâmetros de entrada. Alternativamente, clientes podem também contratar serviços no segundo nível da pilha mostrada na Figura 2.8, o qual é denominado

plataforma como serviço (em inglês, *Platform as a Service - PaaS*), e representa uma plataforma de desenvolvimento de serviços em nuvem que pode ser utilizada pelos clientes. Esta plataforma usualmente incorpora e disponibiliza arcabouços, ferramentas e ambientes de desenvolvimento e teste de software, gerenciamento de configurações e abstração de recursos no nível de hardware.

Os clientes podem também contratar recursos ao nível de hardware utilizando o nível de infraestrutura (em inglês, *Infrastructure as a Service - IaaS*). Para tal, a virtualização é uma técnica bastante utilizada visando compor e decompor recursos físicos, dependendo da flutuação nas demandas dos clientes [18]. Independente do nível da pilha que é consumido pelos clientes, os recursos consumidos (físicos ou virtuais) são usualmente disponibilizados a partir de preços pré-fixados e utilizando o modelo de negócio *pay per use*.

Figura 2.8: Pilha de serviços baseados em computação em nuvem.

2.6 Classificação em Aprendizado de Máquina

O Aprendizado de Máquina [88] é uma sub-área da Inteligência Artificial que investiga algoritmos e técnicas matemáticas e estatísticas para adquirir conhecimento, habilidades e organizar o conhecimento existente de forma automática. O aprendizado é extraído diretamente dos dados, baseado em exemplos passados, sem que haja uma programação explícita. Isto é concretizado por meio de algoritmos e técnicas que permitem ao computador aprender e melhorar seu desempenho em uma determinada tarefa por meio da observação, análise e

generalização de dados.

Uma formulação clássica da abordagem de aprendizado supervisionado é o problema de classificação. A tarefa de classificação consiste em produzir um modelo, a partir de um conjunto de exemplos de treinamento, que mapeia novos exemplos para um valor pertencente a um conjunto de classes. Este tipo de abordagem é utilizado para problemas preditivos, ou seja, tarefas em que são realizadas induções nos dados a fim de gerar predições.

A tarefa de classificação pode ser definida como binária (existência de apenas duas classes) ou multiclasse (mais de duas classes são consideradas). Um classificador utiliza como base um algoritmo de classificação que utiliza exemplos provenientes de uma base inicial de treinamento com o intuito de classificar (ou seja, atribuir um valor pertencente a uma classe) à novas instâncias cujos valores são pertencentes ao mesmo domínio dos exemplos da base de treino. Desse modo, o algoritmo baseia-se em uma base de treino, denominada neste trabalho como TD (*Training Data*), cujos exemplos possuem suas respectivas classes conhecidas. A formalização do problema pode ser realizada com base na notação de vetor [56] ou de sistema de decisão [82].

Notação baseada em Vetor

Seja $TD = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\}$ o conjunto inicial de treinamento, tal que $\mathbf{x_i}$ é um vetor de atributos do i-ésimo exemplo e y_i é um rótulo de classe pertencente ao conjunto $\mathbf{Y} = \{y_1, y_2, \ldots, y_k\}$ contendo k classes. Um caso especial de classificação ocorre quando $k = 2$, o qual representa um problema de classificação binária. Os valores do vetor $\mathbf{x}$ podem pertencer a um domínio numérico, nominal ou categórico, mas na prática domínios numéricos são mais comuns. O domínio dos valores do vetor $\mathbf{x}$ representam uma importante característica do problema, uma vez que os algoritmos de classificação tratam de maneiras distintas os domínios de valores.

Um conjunto de exemplos de treinamento é definido de forma $TD \subseteq \mathbf{X} \times \mathbf{Y}$, tal que $\mathbf{X}$ representa o espaço de atributos dos exemplos contidos nos dados de treinamento e $\mathbf{Y}$ é um conjunto finito e discreto que representa as classes de classificação. Desse modo, um classificador pode ser modelado como um função $g : \mathbf{X} \rightarrow \mathbf{Y}$, a qual é utilizada para classificar instâncias no domínio $\mathbf{X}$ que não estão contidas na base de treinamento inicial.

Notação baseada em Sistema de Decisão

Um problema de classificação pode também ser modelado por meio de um sistema de decisão [82] na forma: $(TD, \mathcal{A} \cup \{d\})$, tal que TD representa um conjunto de exemplos descritos pelo conjunto $\mathcal{A}$ de atributos condicionais e um atributo de decisão d. O valor de um atributo $a \in \mathcal{A}$ associado a uma instância $x \in TD$ é denotado por $a(x)$. Por sua vez, o valor $d(x)$ pertence a um conjunto discreto e finito. Desse modo, um método de classificação (ou classificador) visa predizer a classe $d(t)$ de uma nova instância t, com base no conhecimento da base de treino TD, ou seja, o conjunto $\bigcup_{a \in \mathcal{A}} a(t)$ é fornecido e $d(t)$ é estimado com base no valores em TD.

2.7 Considerações Finais

Neste capítulo, foi apresentado um entendimento geral acerca dos principais tópicos pertencentes ao domínio da presente pesquisa. Inicialmente, foram apresentados os conceitos mais relevantes empregados na área de Qualidade de Dados e mostrados exemplos de dimensões de qualidade de dados. Em seguida, foi discutida a relevância e a definição do problema de deduplicação de dados, assim como as principais técnicas empregadas para lidar com este problema.

Prosseguindo, foram discutidos os pontos de interseção entre a área de Qualidade de Dados e problemas envolvendo Big Data. Para lidar com problemas relacionados à interseção destas duas áreas, é usualmente empregado o paradigma de computação em nuvem, o qual se baseia na alocação dinâmica de recursos computacionais e na cobrança financeira aos clientes de acordo com a frequência e tempo de uso dos recursos alocados. A computação é nuvem é, por sua vez, o paradigma base para a disponibilização de serviços na Internet na categoria de SaaS. Além disso, foram apresentados e formalizados os problemas de classificação coletiva de entidades duplicatas e deduplicação incremental de dados. Por fim, foi apresentada a definição do problema de classificação utilizando aprendizado de máquina.

Os conceitos, formalizações e algoritmos apresentados neste capítulo são utilizados como base para a proposição das contribuições apresentadas neste trabalho, as quais envolvem principalmente: i) a proposição de uma arquitetura para um SMQD; ii) a proposição da estrutura de um SLA de qualidade de dados; iii) a proposição de algoritmos para alocação

dinâmica de recursos computacionais no contexto de um SMQD; iv) a proposição de novas métricas e heurísticas relacionadas ao problema de deduplicação incremental de dados no contexto de classificação coletiva; e v) a proposição de heurísticas para controlar o tamanho dos blocos gerados na indexação de bases de dados para a resolução do problema de deduplicação de dados. No capítulo seguinte, são apresentados os trabalhos do estado da arte relacionados às contribuições propostas neste trabalho.

Capítulo 3

Trabalhos Relacionados

Neste capítulo é apresentada a revisão bibliográfica acerca dos principais trabalhos relacionados encontrados na literatura e suas respectivas propostas. Essencialmente, os trabalhos estão divididos em três grupos: i) trabalhos que investigam provisionamento de recursos computacionais em ambientes de computação em nuvem trabalhos; ii) trabalhos que propõem algoritmos incrementais de qualidade de dados; e iii) trabalhos que tratam o problema de controle do tamanho de blocos produzidos na etapa de indexação no contexto de deduplicação de dados.

3.1 Provisionamento de Recursos em Computação em Nuvem

A Computação em Nuvem se tornou recentemente uma área de pesquisa bastante ativa. Alguns trabalhos focaram em desenvolver modelos utilizados em computação em grades [68] e redes fechadas [45] visando maximizar a utilização de recursos em ambientes deste tipo. Em [68], os autores representam recursos computacionais e os requisitos dos serviços utilizando uma matriz de recursos e requisitos quantificados. Estas abordagens [45; 68] são diferentes deste trabalho porque visam maximizar a utilização de recursos no contexto do processamento de cargas de trabalho estáticas e pré-definidas, as quais são significativamente diferentes das cargas de trabalho dinâmicas que são processadas no contexto de um SMQD. Por sua vez, em [90] os autores propõem algoritmos que são projetados para

garantir que provedores de SaaS possam gerenciar mudanças ocorridas em requisitos de clientes, mapear estes requisitos em valores de parâmetros no nível de infraestrutura e lidar com a heterogeneidade de configuração de máquinas virtuais. No entanto, estes algoritmos são úteis em contextos específicos de requisitos submetidos para um sistema de CRM ou ERP implantados em um ambiente de computação em nuvem.

Algoritmos de provisionamento para computação em nuvem foram também investigados no contexto de banco de dados [1; 8; 46; 92]. O principal objetivo dos autores em [92] foi otimizar a alocação de recursos no contexto de cargas de trabalho representadas como um conjunto de consultas de banco de dados. Por sua vez, os autores em [1] propuseram uma abordagem para garantir que consultas que escalam de maneira satisfatória em pequenas bases de dados continuem a escalar de maneira aceitável quando as bases de dados se tornem maiores e, desse modo, continuem respeitando requisitos especificados em um SLO (*Service Level Objective*). Em [8], os autores empregam técnicas utilizadas para adaptar o processamento de consultas, em ambientes de computação em nuvem, sobre volumes de dados massivos que são armazenados de maneira distribuída. Por fim, um serviço é proposto por [46] para garantir requisitos de eficiência e privacidade de consultas de similaridade. Note que todas estas abordagens [1; 8; 92] são relacionadas ao processamento e otimização de consultas e, por este motivo, não são adequadas para o provisionamento de recursos computacionais no contexto de um SMQD.

Técnicas de aprendizado de máquina também foram anteriormente investigadas [10; 38; 47] no contexto de computação em nuvem. No entanto, as abordagens propostas neste trabalho são diferentes das abordagens apresentadas em [10; 38; 47] porque neste trabalho são avaliadas técnicas de aprendizado de máquina diferentes (*SVM*, *random forests* e *kNN*) e é empregada uma carga de trabalho representada como um conjunto de entradas quantificadas associadas à tarefas de qualidade de dados.

Outros trabalhos [37; 71; 85] empregaram técnicas de inteligência artificial para lidar com problemas como avaliação de serviços, requisitos de segurança e minimização de consumo de energia. Em [37], os autores tratam o problema de seleção e composição de serviços, com base em valores de *QoS* (qualidade de serviço esperada), empregando um algoritmo discreto baseado na técnica de colônia de abelhas artificiais. Por sua vez, os autores em [85] utilizam um algoritmo baseado em inteligência de enxames para provisionar recursos em

um ambiente de computação em nuvem levando em consideração restrições de segurança. Além disso, uma heurística iterativa não determinística (denominada Evolução Simulada) é proposta pelos autores em [71] para lidar com o problema de alocação de VMs tentando minimizar o consumo de energia.

Em [94], são elencadas diversas técnicas que podem ser exploradas para tratar o problema de provisionar dinamicamente a quantidade de instâncias de uma aplicação com base em modelos específicos de predição, tais como teoria das filas [80], teoria do controle [40] e aprendizado de máquina [7] baseada em estatística. Em [78], técnicas de aprendizado de máquina são aplicadas visando classificar tráfegos em redes para predizer possíveis ataques de usuários maliciosos. Os autores de [7] investigam o problema de provisionamento dinâmico de $data\ centers$ utilizando técnicas de aprendizado de máquina baseadas em estatística. Por sua vez, um levantamento sobre diversas técnicas para tratar o provisionamento de recursos é realizado pelos autores em [36], no qual é apresentada a utilização de aprendizado de máquina para modelar tempos de resposta de sistemas [27] e o tempo de resposta de tarefas executadas em lote [74].

O presente trabalho é similar aos trabalhos propostos pelos autores em [7; 27; 74; 78; 80; 94], uma vez que muitos algoritmos propostos neste trabalho aplicam técnicas de aprendizado de máquina para lidar com problemas no contexto de computação em nuvem. No entanto, neste trabalho é investigado um problema diferente (redução de custos da tarefa de deduplicação de dados) dos problemas investigados pelos autores em [7; 27; 74; 78; 80; 94]. Outrossim, neste trabalho é proposta uma solução para a alocação de VMs baseada no processamento de valores de tempos de restrição e em entradas de algoritmos de qualidade de dados visando a minimização de custos de infraestrutura, ao invés de visar objetivos como restrições de segurança ou a minimização de consumo de energia.

3.2 Algoritmos para Deduplicação Incremental de Dados

Alguns trabalhos [15; 30; 86] foram pioneiros na investigação do problema de deduplicação incremental de dados baseada em classificação coletiva. Em [86], os autores propõem uma abordagem para DID baseada em uma condição denominada *general incremental* que, se satisfeita por um algoritmo de agrupamento, facilita a minimização de uma função de pena-

lidade associada aos agrupamentos gerados pelos algoritmos de DID. Por sua vez, os autores em [30] propuseram três métodos para a geração de agrupamentos ótimos (ou seja, cuja penalidade associada é mínima) e uma heurística (denominada $Greedy$) para o problema de DID. No Experimento III, conduzido neste trabalho, as heurísticas para DID propostas, juntamente com três algoritmos (*Star*, *Center* e *Merge-Center*) de agrupamento automático disponíveis no estado da arte, foram comparados com o algoritmo $Greedy$. A principal diferença entre a abordagem de DID proposta neste trabalho (Capítulo 6) e o algoritmo $Greedy$ está relacionada com o pré-processamento realizado pelas heurísticas denominadas filtros de componentes de cobertura, as quais visam limitar a quantidade de agrupamentos processados no processo de DID ao selecionarem os agrupamentos que possuem mais chance de serem modificados.

Por fim, uma abordagem para o agrupamento incremental de entidades é proposta pelos autores em [15], a qual promove um esquema de votação para estimar o agrupamento ideal para novas entidades inseridas em um grafo de similaridade e atualizar os agrupamentos de maneira recursiva. Este algoritmo apresenta a desvantagem relacionada à possibilidade de processar recursivamente o grafo de similaridade como um todo, enquanto as heurísticas propostas neste trabalho (Capítulo 6) visam tornar o processo de classificação coletiva no contexto de DID mais eficiente, ao limitar a quantidade de agrupamentos processados por um algoritmo de DID ou empregar um algoritmo de agrupamento automático de baixa complexidade. Na Tabela 3.1, é apresentada uma comparação entre diversas características dos trabalhos para DID do estado da arte e da abordagem proposta no Capítulo 6 (Nascimento et. al 2017).

O problema de DID no contexto de classificação coletiva também está relacionado à problemas investigados recentemente no estado da arte, tais como a deduplicação de dados utilizando esquemas de votação [32], deduplicação de dados em tempo real [61] e deduplicação de dados em tempo de consulta [69]. Em [32], os autores investigam o problema de reduzir custos associados à execução da tarefa de deduplicação de dados cujos resultados são baseados em votos de usuários, assumindo que alguns votos de entrada não são confiáveis ou são menos confiáveis. Para tal, são empregadas estratégias que assumem uma premissa similar à empregada pelo algoritmo $Greedy$, ou seja, que as informações provenientes das novas entidades processadas devem ser utilizadas para atualizar e melhorar os resultados de

Tabela 3.1: Comparativo entre abordagens para deduplicação incremental de dados no contexto de classificação coletiva.

Abordagem	**Utiliza função de penalidade**	**Limita o tamanho do subgrafo a ser processado**	**Filtra o conjunto de agrupamentos a ser processado**
Nascimento et. al 2017	Sim	Sim	Sim
Gruenheid et. al 2014	Sim	Sim	Não
Costa et. al 2010	Não	Não	Não

deduplicação de dados [32]. Por sua vez, uma abordagem utilizada para exploração iterativa, consulta e deduplicação de dados geo-referenciadas distribuídas ao longo de diversos provedores de dados é proposta em [61]. A abordagem proposta estende de maneira incremental os resultados de deduplicação ao longo do processamento de novas entidades.

Finalmente, em [69] é investigado o problema de deduplicação de dados em tempo de consulta, no contexto de integração de dados na web, empregando abordagens iterativas e incrementais baseadas em dados armazenados em cache. Uma vez que algumas heurísticas propostas neste trabalho (Capítulo 6) visam otimizar o processo de DID, estas ideias podem ser futuramente exploradas no contexto de outros problemas [32; 61; 69] também relacionados à deduplicação de dados.

3.3 Algoritmos Utilizados no Controle do Tamanho de Blocos de Entidades

A etapa de indexação de uma tarefa de deduplicação de dados pode empregar uma grande variedade de técnicas, dentre as quais: *suffix array* [17], *canopy clustering* [14], q-grams [29] e abordagens híbridas [20]. Estas técnicas podem ainda ser otimizadas por meio da adoção de abordagens de paralelização [28; 52; 53; 54; 55]. Após a execução da etapa de

indexação, é possível aplicar algoritmos para podar os blocos de entidades visando produzir blocos de tamanhos configurável. O controle no tamanho dos blocos produzidos é importante para limitar a quantidade de comparações entre entidades e, assim, permitir a aplicação de técnicas de classificação complexas e/ou viabilizar a utilização de técnicas para deduplicação de dados em tempo real.

Muitos trabalhos publicados no estado da arte [13; 22; 63; 65] exploram estas técnicas, tanto teórica quanto experimentalmente, e discutem diversas relações de custo-benefício (envolvendo tempo de execução, eficácia de blocagem e resultados de qualidade) produzidos por estes métodos. Por exemplo, em [13] são apresentadas e avaliadas 12 técnicas de blocagem e é mostrado que as técnicas *standard blocking*, *adaptive sorted neighborhood* e *canopy clustering* produziram os melhores resultados na avaliação conduzida. Ainda que estas técnicas tenham produzido resultados promissores, as mesmas não permitem controle sobre o tamanho dos blocos produzidos.

Outros trabalhos [26; 50; 67; 95] propuseram algoritmos que visam controlar o tamanho de agrupamentos gerados por técnicas de agrupamento automático. Para tal, estes trabalhos exploram programação linear inteira [95], heurísticas [26] para modificar o algoritmo k-means [93], o algoritmo húngaro [9] para tratar a etapa de alocação do algoritmo k-means balanceado [50] e o algoritmo Levenberg–Marquardt [51] para também modificar [67] o algoritmo k-means. As abordagens propostas no Capítulo 7 são similares aos trabalhos propostos em [26; 50; 67; 95] uma vez que este trabalho explora técnicas para controlar o tamanho de blocos. No entanto, a avaliação conduzida neste trabalho visa avaliar as abordagens propostas no contexto de múltiplas chaves de blocos que são empregadas na etapa de indexação de um problema de deduplicação de dados, ao invés de tratar o problema de controlar o tamanho dos agrupamentos produzidos por algoritmos de agrupamento automático. Além disso, as abordagens propostas pelos autores em [26; 50; 67; 95] requerem a computação de similaridade entre todos os pares de entidades na base de dados, ao passo que as abordagens propostas no Capítulo 7 visam evitar todas as comparações entre as entidades ao podar os blocos de uma coleção de blocagens recebida como entrada.

Em [62], os autores propõem uma abordagem de blocagem agnóstica, denominada *meta blocking*, a qual trata o problema de poda de uma coleção de blocagens (Definição 7.3.4) explorando a coocorrência das entidades nos blocos e empregando diversos esquemas de

pontuação. A abordagem proposta em [62] produz um *Undirected Blocking Graph* (UBG) e poda o grafo gerado empregando abordagens baseadas no cálculo de números de nós vizinhos e no peso das arestas. Por fim, o grafo podado é processado em uma etapa denominada $Blocking\ Collecting$ no intuito de gerar o resultado final de blocagem. Uma abordagem de *meta blocking* supervisionado é também explorada no trabalho proposto em [64], no qual é definido o problema de *meta blocking* supervisionada empregando um grafo de blocagem generalizado, exemplos de treinamento e as técnicas de poda propostas em [62]. As abordagens propostas no Capítulo 7 são similares aos trabalhos propostos em [62; 64] porque este trabalho também trata o problema de poda de uma coleção de blocagens. No entanto, as abordagens propostas em [62; 64] não permitem controlar o tamanho dos blocos produzidos, ao passo que as abordagens propostas neste trabalho permitem garantir que todos os blocos produzidos apresentem tamanhos que respeitem um intervalo configurável. Além disso, as abordagens propostas no Capítulo 7 não requerem uma base de treinamento, ou seja, são algoritmos não supervisionados.

O trabalho proposto em [87] utiliza uma abordagem que usa exemplos de treinamento e realiza operações de união e divisão de blocos para indexar bases de dados. As abordagens propostas no Capítulo 7 são diferentes do trabalho proposto em [87] porque as heurísticas propostas neste trabalho são não supervisionadas e permitem o controle sobre o tamanho dos blocos produzidos a partir de parâmetros configuráveis. Finalmente, em [25] os autores propuseram uma abordagem para controlar o tamanho de blocos com base em dois algoritmos recursivos ($SimilarityBasedClustering$ e $SizeBasedClustering$) que: i) empregam funções de chave de bloco iterativamente para dividir blocos grandes; ii) unem blocos pequenos com base na similaridade entre valores de chave de bloco; e iii) utilizam uma função de penalidade para determinar se dois blocos devem ser unidos. As heurísticas propostas no Capítulo 7 são similares ao trabalho proposto em [25] porque também visam podar blocos recebidos como entrada para controlar dos tamanho de blocos produzidos. No entanto, a abordagem proposta em [25] não é adequada para podar uma coleção de blocagens que é inicialmente indexada por múltiplas funções de chave de bloco (Definição 7.3.6) porque os algoritmos $SimilarityBasedClustering$ e $SizeBasedClustering$ funcionam indexando a base de dados com a aplicação de uma única função de chave de bloco (as demais funções de chave de bloco são empregadas iterativamente visando dividir blocos grandes). Por outro

lado, as heurísticas propostas neste trabalho são capazes de processar uma base de dados indexada por múltiplas funções de chave de bloco explorando a coocorrência das entidades para dividir blocos grandes. Outrossim, neste trabalho a união de blocos é realizada com base no tamanho da interseção dos blocos (ao invés de utilizar a similaridade entre valores de chave de bloco). Por fim, neste trabalho foram também propostas heurísticas para remover entidades de blocos ($lECP$, $lBCP$ e $LBSP$) e excluir blocos ($lBCE$), ao passo que em [25] os autores não tratam estes problemas.

Na Tabela 3.2, é apresentada uma comparação entre diversas características dos trabalhos para controle do tamanho de blocos do estado da arte e da abordagem proposta no Capítulo 7 (Nascimento et. al 2017).

Tabela 3.2: Comparativo entre abordagens para controle do tamanho de blocos.

Abordagem	**Garante tamanho mínimo dos blocos**	**Garante tamanho máximo dos blocos**	**Explora coocorrência**	**Múltiplas chaves de bloco na blocagem Inicial**	**Calcula a similaridade entre pares de blocos**	**Requer a comparação entre todas as entidades**
Nascimento et. al 2017	sim	sim	sim	sim	não	não
Papadakis et. al 2014	não	não	sim	sim	não	não
Fisher et. al 2015	sim	sim	não	não	sim	não
Ganganath et. al 2014	sim	sim	não	não	sim	sim

Capítulo 4

Arquitetura de um Serviço de Monitoramento de Qualidade de Dados (SMQD)

Visando a concretização do objetivo geral do trabalho, o qual consiste na proposição de abordagens para a redução de custos associados à execução de algoritmos de deduplicação de addos no contexto de um SMQD, neste capítulo são propostas uma arquitetura em alto nível para um SMQD e uma estrutura para definição de SLAs no contexto de qualidade de dados, contexto no qual serão propostas as abordagens investigadas neste trabalho visando reduzir custos da deduplicação de dados.

Um dos principais objetivos da arquitetura proposta consiste em englobar componentes que visam diminuir os custos financeiros associados ao funcionamento de um serviço para monitoramento contínuo de qualidade de bases de dados volumosas (e, consequentemente, reduzir os custos repassados para clientes de um serviço para este fim). Para tal, a arquitetura deve: i) englobar estratégias para alocar a mínima quantidade de recursos computacionais necessária para atender os requisitos definidos pelos clientes; ii) evitar o reprocessamento de comparações entre entidades já comparadas anteriormente por meio da adoção de algoritmos incrementais de deduplicação de dados; e iii) permitir controlar a quantidade de comparações entre entidades a ser realizada no processo de deduplicação de dados.

4.1 SLA de Qualidade de Dados

Como discutido na Seção 2.5, um SLA representa um contrato eletrônico que visa descrever as garantias do serviço prestado pelo provedor, especificando principalmente a qualidade esperada em relação às funcionalidades oferecidas pelo serviço. Na prática, medir a conformidade de serviços com características pré-definidas foi uma estratégia por muito tempo utilizada em contextos como redes de computadores e administração de servidores [49]. Por exemplo, é geralmente simples de ser verificado se os dados estão trafegando de maneira lenta em uma rede ou se um servidor encontra-se sem funcionamento.

No entanto, a diferença entre o gerenciamento de níveis de qualidade de serviços e o gerenciamento de níveis de qualidade de dados ocorre na perceptível variabilidade em definir um nível aceitável de qualidade de dados [49] e como representar estes níveis de aceitação relacionados à qualidade de dados. Em outras palavras, na prática é um desafio representar um SLA de Qualidade de Dados (em inglês, *Data Quality Service Level Agreement* - DQSLA). Portanto, é importante adotar uma estrutura padronizada para representar um DQSLA englobando tanto os requisitos de qualidade de serviços envolvendo qualidade de dados, especificados por um cliente, quanto as garantias definidas pelo provedor de um serviço de monitoramento de qualidade de dados [59]. Dessa forma, é possível representar objetivos de negócios, no contexto de qualidade de dados, definidos em alto nível por meio de parâmetros envolvendo a qualidade esperada dos dados. Ao listar expectativas críticas, métodos de avaliação e parâmetros específicos, os gerentes de negócio podem associar a governança dos dados com níveis de sucesso em atividades de negócio [49].

Definições de SLA existentes no estado da arte visam englobar aspectos específicos do contexto no qual os contratos são utilizados. Por exemplo, os autores em [76] utilizam um SLA no contexto dos problemas de escalonamento e execução de consultas em um ambiente de nuvem e, por este motivo, consideram no SLA valores como os tempos de escalonamento, de despacho, de espera em pilha, de execução e apresentação dos resultados da consulta (dentre outros). Por sua vez, os autores de [66] consideram parâmetros como vazão do canal de comunicação, capacidade de processamento das tarefas, tamanho das tarefas, número de sessões, dentre outros. Similarmente, o principal objetivo da representação de uma abordagem específica para um SLA de qualidade de dados consiste em definir um nível de acordo de

serviço que englobe aspectos específicos do contexto do monitoramento contínuo da qualidade de dados armazenados em nuvem realizado por um SMQD, tais como: a dimensão de qualidade de dados, a quantidade de alterações na base de dados que irá disparar novas avaliações das bases de dados e os valores dos parâmetros dos algoritmos de qualidade de dados (uma vez que tais valores podem influenciar fortemente o tempo de execução das tarefas).

Assim, a representação de um DQSLA é proposta utilizando uma 9-tupla contendo os seguintes elementos: $\langle \mathcal{D}, DQ_{dim}, M_{rules}, \Delta ts, |\Delta \mathcal{D}|_{thr}, T_{res}, \mathcal{R}, \mathcal{P}, Init \rangle$, tal que:

$\mathcal{D}$ é a base de dados a ser monitorada;

DQ_{dim} é uma dimensão de qualidade de dados;

M_{rules} define os detalhes (algoritmos e parâmetros) que serão utilizados na avaliação da base de dados $\mathcal{D}$;

Δts é um intervalo de tempo no qual o DQSLA é válido;

$|\Delta \mathcal{D}|_{thr}$ representa a quantidade de mudanças na base de dados ($\mathcal{D}$) que irá disparar uma nova execução no serviço para avaliar a qualidade de $\mathcal{D}$, levando em consideração os detalhes especificados no parâmetro M_{rules};

T_{res} é a eficiência esperada associada a uma avaliação da base de dados $\mathcal{D}$. Na prática, este parâmetro representa uma restrição de tempo associada à execução de um algoritmo de qualidade de dados que irá processar a base de dados $\mathcal{D}$;

$\mathcal{R}$ é o método (em tempo real e/ou histórico) adotado para reportar os resultados subsequentes da avaliação da base de dados ($\mathcal{D}$) na interface disponibilizada pelo serviço;

$\mathcal{P}$ especifica as penalidades que serão aplicadas ao provedor do serviço nos casos em que a restrição de tempo (T_{res}) não for atendida pelo serviço;

$Init$ é um valor lógico (verdadeiro ou falso) que indica a necessidade de realizar uma avaliação inicial da base de dados monitorada;

Alguns dos valores de parâmetros de um DQSLA podem ser especificados utilizando regras BNF (*Backus Naur Form*) pré-definidas, por exemplo:

[$\mathcal{R}$] := Tempo Real | Histórico | Tempo Real $\wedge$ Histórico

[DQ_{dim}] := Acurácia | Completude | Volatilidade | Duplicação | Consistência | ...

Usando estas regras, no Exemplo 4.1.1 é apresentado um exemplo de DQSLA utilizando a dimensão $Duplication$.

Exemplo 4.1.1 (Exemplo de DQSLA) *Dado o DQSLA_13*$\langle$*101, Duplicação,* $Dedup_{rules}$,

(2015-11-09 09:20, 2015-11-11 10:20), 6000, 15 min, Tempo Real, $\mathcal{P}$, True⟩, sua semântica deve ser interpretada da seguinte maneira: a base de dados (101) deve ser continuamente monitorada para detecção de entidades duplicadas durante o intervalo de tempo (2015-11-09 09:20, 2015-11-11 10:20). Cada avaliação da base de dados deve ser realizada de acordo com os valores do parâmetro $Dedup_{rules}$ e não deve durar mais de 15 minutos para ser executada. Além da avaliação inicial da base de dados ($Init$ =True), a base de dados (101) deve ser re-avaliada após cada 6000 modificações em seu conteúdo. Além disso, os resultados da avaliação da base de dados monitorada devem ser continuamente reportados na interface do serviço ($\mathcal{R}$ = Tempo Real). Por fim, é também especificada uma penalidade ($\mathcal{P}$) para o provedor do serviço caso o processo de avaliação dure mais do que 15 minutos (uma possível abordagem para definir os valores do parâmetro $\mathcal{P}$ é discutida na Seção 5.1).

Por sua vez, as regras para a definição de valores associados ao parâmetro M_{rules} de um DQSLA também podem ser especificadas por meio de uma gramática BNF. Exemplos desta especificação podem ser verificados em [58]. Por fim, note que é possível representar um DQSLA utilizando um documento XML e validar a sintaxe dos valores dos parâmetros, por meio da utilização de regras BNF previamente especificadas, utilizando validadores de esquemas XML.

4.2 Arquitetura Proposta

Nesta seção, é proposta uma arquitetura em alto nível para um Serviço de Monitoramento de Qualidade de Dados (SMQD). Como mencionado na Seção 2.5, a adoção do paradigma de computação em nuvem para a disponibilização de serviços traz uma série de benefícios e, por este motivo, a arquitetura proposta é baseada em conceitos deste paradigma. Na prática, um SMQD pode precisar lidar com situações como tempos de restrições bastante restritos (especificados por meio de DQSLAs) e o processamento de bases de dados volumosas. Por este motivo, a arquitetura proposta incorpora uma série de estratégias visando a redução de custos associados ao funcionamento do serviço, tais como a adoção de provisionamento dinâmico de recursos para a execução de algoritmos de qualidade de dados.

A elasticidade na alocação de recursos computacionais é uma característica fundamental no processamento de dados armazenados em nuvem. No contexto de um SMQD, o serviço

deve adotar estratégias para processar automaticamente o conteúdo de um DQSLA e estimar uma quantidade ideal de recursos para executar os algoritmos especificados respeitando as restrições de tempo associadas aos mesmos. Na prática, o volume das bases de dados que precisam ser monitoradas e a complexidade dos algoritmos de qualidade de dados que precisam ser executados podem variar significativamente ao longo do tempo. Por conseguinte, a quantidade de recursos computacionais que deve ser alocada pelo serviço pode também variar notoriamente ao longo do tempo. Os algoritmos de provisionamento de recursos computacionais propostos neste trabalho para tratar este problema são detalhados no Capítulo 5.

No contexto de um SMQD, a elasticidade na alocação de recursos computacionais é importante por uma série de motivos: i) permitir que os requisitos de níveis de serviço, especificados pelos clientes usando um DQSLA, sejam atendidos (o que evita a aplicação de penalidades para o serviço); e ii) otimizar a utilização de recursos, o que acarreta na diminuição de custos de infraestrutura para a execução do SMQD e, consequentemente, na redução dos custos repassados aos clientes.

A arquitetura em alto nível proposta para o SMQD é mostrada na Figura 4.1, na qual são apresentados os principais módulos, relacionamentos entre os módulos e fluxos de dados e tarefas realizadas pelo serviço. O cliente do SMQD interage com o serviço das seguintes maneiras principais: i) inserindo novos DQSLAs e/ou atualizando DQSLAs existentes; e ii) visualizando os resultados do monitoramento da qualidade dos dados em tempo real ou por meio de consultas analíticas. Os passos representados na arquitetura ilustram a sequência de atividades realizadas pelo SMQD para o processamento automático de DQSLAs e monitoramento da qualidade das bases de dados.

No passo 1, um cliente, o qual inicialmente possui uma ou mais bases de dados armazenadas em nuvem, submete os parâmetros de entrada do DQSLA por meio de um formulário em uma interface web ou realizando o envio de um arquivo XML representando os parâmetros do DQSLA. No passo 2, o DQSLA recebido é validado pelo módulo *SLA Validator* utilizando um esquema XML, selecionado no passo 3, de acordo com os valores dos parâmetros DQ_{dim} e M_{rules} do DQSLA recebido. No passo 4, se a estrutura e o conteúdo do DQSLA forem validados de maneira correta por meio do esquema XML selecionado, o DQSLA é mapeado (passo 5) e armazenado (passo 6) como um conjunto de parâmetros que

serão principalmente utilizados pelos algoritmos de qualidade de dados.

Em seguida, se o valor do parâmetro $Init$ do DQSLA foi definido como $true$, no passo 7 o módulo *Provisioning Planner* processa e agrega informações provenientes do algoritmo de qualidade de dados a ser executado, dos metadados (e.g., tamanho e distribuição dos dados na base de dados a ser monitorada) e (opcionalmente) de uma base de treinamento para estimar uma infraestrutura de máquinas virtuais adequada para executar um algoritmo de qualidade de dados e respeitar a restrição de tempo especificada no DQSLA, ou seja, gerando um tempo de execução do algoritmo de qualidade de dados que é menor ou igual ao tempo de restrição (T_{res}). Como saída da etapa 7, uma configuração de *cluster* de máquinas virtuais é estimada. A especificação da configuração de *cluster* é utilizada pelos módulos *Business Model* (para futuramente calcular custos a serem repassados para o cliente) e *Resource Configurator*, o qual é responsável por utilizar tecnologias de computação em nuvem para alocar um *cluster* de máquinas virtuais e executar os algoritmos de qualidade de dados de maneira distribuída.

No passo 13, após a execução do algoritmo de qualidade de dados ser completada, o módulo *Execution Summarizer* agrega e sumariza os dados da execução no intuito de popular um repositório contendo dados históricos de execução de algoritmos, permitindo assim futuras consultas analíticas. Além disso, o serviço pode também i) popular uma base de treinamento (passo 14) que pode ser futuramente utilizada pelo módulo *Provisioning Planner*; e ii) apresentar os resultados da execução em uma interface gráfica (passo 15) para acompanhamento em tempo real pelos clientes do serviço.

Outro possível fluxo de tarefas (mostrado em linhas tracejadas na arquitetura) realizado pelo SMQD é disparado quando a quantidade de alterações (inserções, atualizações ou remoções) na base de dados monitorada excede o valor do parâmetro $|\Delta D|_{thr}$ especificado pelo cliente no DQSLA. Quando este cenário é detectado por meio de um monitoramento constante dos *logs* de alterações na base de dados monitorada, o qual é realizado pelo módulo *Online Metadata Monitoring*, este fato é notificado ao módulo *Provisioning Planner* que, por sua vez, é responsável por estimar uma nova configuração de *cluster* de máquinas virtuais para executar um algoritmo de qualidade de dados, como foi anteriormente explicado nesta seção.

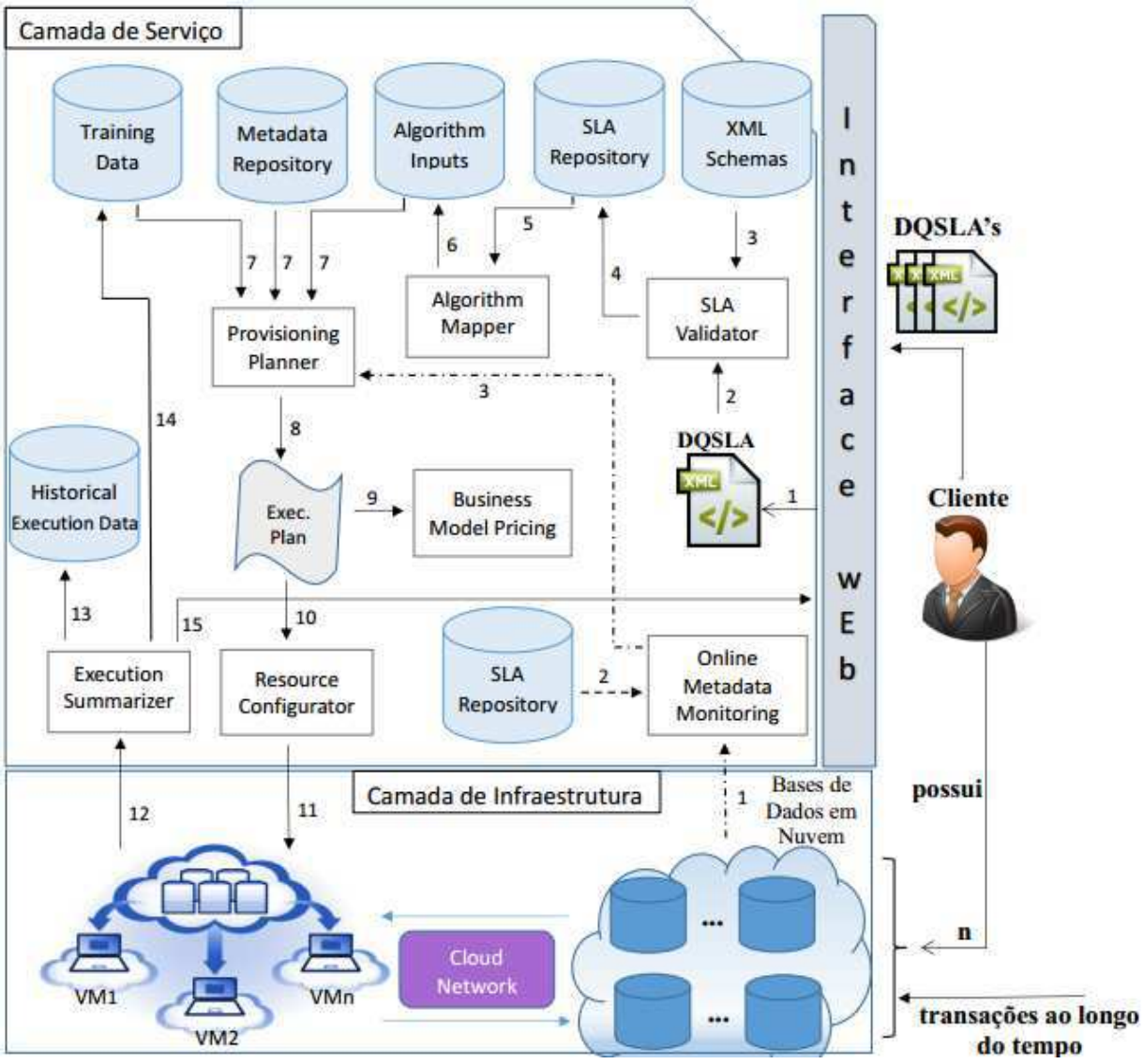

Figura 4.1: Arquitetura proposta para o serviço de monitoramento de qualidade de dados.

4.3 Considerações Finais

Neste capítulo, foram propostas uma arquitetura em alto nível para um SMQD e uma estrutura para a representação de SLAs de qualidade de dados. A arquitetura proposta apresenta diversos módulos visando a diminuição de custos associados à disponibilização de um SMQD, por exemplo: i) *Provisioning Planner*: visa reduzir custos associados à execução de algoritmos de qualidade de dados a partir da alocação dinâmica de recursos computacionais (usando algoritmos de provisionamento); ii) *Business Model Pricing*: provê modelos para o cálculo de custos de infraestrutura e de penalidades que são utilizados pelo módulo *Provisioning Planner*; iii) *Algorithm Mapper*: visa selecionar algoritmos de qualidade no intuito de reduzir os custos de execução ao empregar, por exemplo, heurísticas que controlam o tamanho dos blocos gerados pela etapa de indexação no contexto de um problema de deduplicação de dados; e iv) *Online Metadata Monitoring*: visa monitorar mudanças ocorridas nos dados dos bancos de dados, permitindo assim, a execução de algoritmos incrementais de qualidade de dados. As contribuições deste trabalho estão relacionadas principalmente com as funcionalidades dos módulos *Provisioning Planner* e *Business Model Pricing*.

Os artefatos produzidos neste capítulo são utilizados como base para a proposição de abordagens para reduzir custos associados à deduplicação de dados no contexto de um SMQD. Mais especificamente, estratégias para provisionamento dinâmico de máquinas virtuais, para deduplicação incremental de dados e para controle do tamanho de blocos produzidos na fase de indexação de um problema de deduplicação de dados (ver Figura 2.1). No capítulo seguinte, são propostos vários algoritmos de provisionamento de recursos computacionais, os quais estão relacionados com o módulo *Provisioning Planner* da arquitetura proposta (Figura 4.1).

Capítulo 5

Provisionamento Dinâmico de Recursos Computacionais em um SMQD

Neste capítulo, são propostos: i) um modelo de custo para um SMQD, o qual visa calcular custos de infraestrutura para executar algoritmos de qualidade de dados e custos de penalidade (associados ao não cumprimento de requisitos de SLA) infringidos ao serviço; e ii) um conjunto de algoritmos de provisionamento de recursos computacionais baseados em heurísticas e aprendizado de máquina. O modelo de custo e os algoritmos propostos estão relacionados aos módulos *Business Model Pricing* e *Provisioning Planner*, respectivamente, os quais estão presentes na arquitetura do SMQD proposta (Figura 4.1).

A principal motivação para a proposição dos algoritmos de provisionamento apresentados neste capítulo é permitir avaliar se abordagens baseadas em aprendizado de máquina e heurísticas são eficazes para reduzir custos da execução de tarefas de deduplicação de dados em nuvem, levando em consideração cargas de trabalho específicas do contexto de um SMQD, as quais são caracterizadas como um conjunto de entradas para tarefas de deduplicação de dados. Além disso, o modelo de custo e algoritmos propostos neste capítulo estão intrinsecamente relacionados, pois o modelo é utilizado para calcular a eficácia dos algoritmos de provisionamento de recursos propostos para reduzir os custos de um SMQD.

5.1 Modelo de Custo

No contexto de um SMQD, os recursos computacionais alocados pelo serviço são modelados como um conjunto de máquinas virtuais utilizadas para a execução de algoritmos de qualidade de dados. Na prática, o módulo *Provisioning Planner*, na arquitetura proposta para o SMQD, é responsável por estimar um *cluster* de máquinas virtuais (em inglês, *Virtual Machine* - VM) para executar um algoritmo de qualidade de dados. A especificação deste *cluster* é denominada *Classe de Configuração* e é composta por um par $\langle$*#VM*, $VM\_Configuration\rangle$, o qual denota a quantidade e a configuração das máquinas virtuais, respectivamente.

Sejam $N = \{n_1, n_2, \ldots, n_m\}$ a quantidade de máquinas virtuais disponíveis e que podem ser alocadas para a execução de um algoritmo de qualidade de dados, $\gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_k\}$ um conjunto cujos elementos representam as configurações de máquinas virtuais disponíveis e $\mathcal{C}l$ uma classe de configuração representada por um par $\langle n_i, \gamma_j\rangle$, tal que $n_i \in N$ e $\gamma_j \in \gamma$. Assim, $\mathcal{C}_{conf} = (N \times \gamma) = \{\mathcal{C}l_1, \mathcal{C}l_2, \ldots, \mathcal{C}l_{m\times k}\}$ representa o conjunto de todas as classes de configuração possíveis que podem ser escolhidas por um algoritmo de provisionamento de recursos computacionais no contexto de um SMQD.

Sejam $Proc(\mathcal{C}l_i)$ e $\mathcal{C}l_i.n$ a capacidade de processamento e o número de nós (VMs) da classe $\mathcal{C}l_i$, respectivamente. Para facilitar a proposição de algoritmos de provisionamento de recursos computacionais, $\mathcal{C}_{conf}$ é utilizado como um conjunto parcialmente ordenado pelas seguintes regras:

- $\mathbf{R_1}$) $\forall \mathcal{C}l_i, \mathcal{C}l_j \in \mathcal{C}_{conf}[(i \geq j) \Rightarrow Proc(\mathcal{C}l_i) \geq Proc(\mathcal{C}l_j)]$, i.e., quanto maior o índice da classe de configuração do *cluster*, maior é sua capacidade de processamento;

- $\mathbf{R_2}$) $\forall \mathcal{C}l_i, \mathcal{C}l_j \in \mathcal{C}_{conf}[((i < j) \wedge (Proc(\mathcal{C}l_i) = Proc(\mathcal{C}l_j)) \Rightarrow \mathcal{C}l_i.n < \mathcal{C}l_j.n]$, i.e., se duas classes de configuração de *cluster* diferentes possuem o mesma capacidade de processamento (pois o valor resultante da multiplicação entre a quantidade de VMs e vCPUs alocados é o mesmo), a classe de configuração com menor índice aloca uma quantidade menor de máquinas virtuais.

Na Figura 5.1, é ilustrada a relação entre os índices das classes de configuração e as respectivas capacidades de processamento produzidas. Como mostrado na Figura 5.1, quanto

maior o valor do índice de uma classe de configuração, maior será o poder de processamento do $cluster$ de máquinas virtuais associado à classe de configuração.

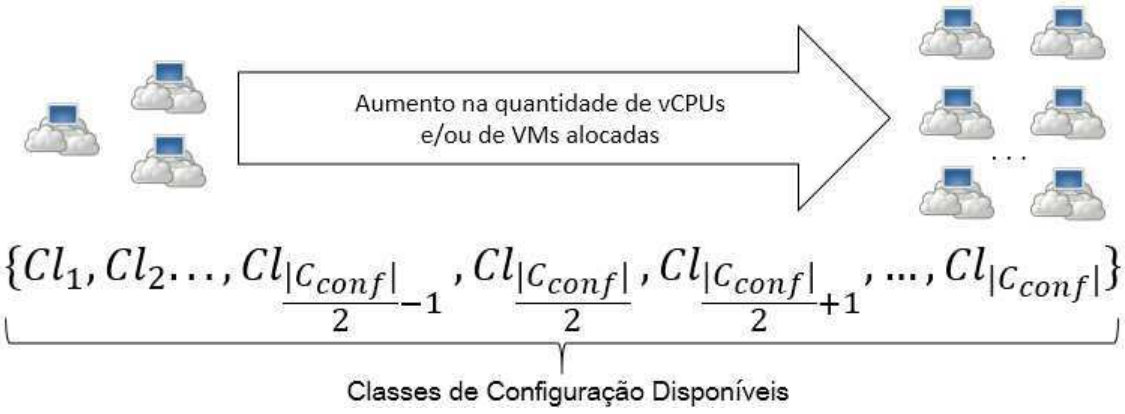

Figura 5.1: Relação entre classes de configuração e os recursos computacionais alocados.

Para a execução de algoritmos de deduplicação de dados no contexto de um SMQD, uma configuração homogênea para as máquinas virtuais é preferível, pois esta opção tende a diminuir a chance de ocorrer um problema denominado desbalanceamento de carga [44]. Este problema pode ocorrer (dentre outros motivos) em decorrência de uma máquina específica (com uma configuração inferior às demais) executar uma sub-tarefa bastante custosa e, consequentemente, adiar consideravelmente a completude da tarefa de deduplicação de dados (no Apêndice D.1 são apresentados mais detalhes sobre o problema de desbalanceamento de carga).

5.1.1 Custos do Serviço

Seja $C\langle t_i, t_f\rangle$ o conjunto de clientes do SMQD durante o intervalo de tempo (t_i, t_f). Durante este intervalo de tempo, cada cliente $c \in C\langle t_i, t_f\rangle$ pode criar um ou mais DQSLAs, os quais são representados pelo conjunto $S_c\langle t_i, t_f\rangle$. Por sua vez, cada DQSLA $s \in S_c\langle t_i, t_f\rangle$ pode disparar um conjunto $\mathcal{T}_s\langle t_i, t_f\rangle$ de tarefas de qualidade de dados durante o intervalo de tempo (t_i, t_f).

A execução de cada tarefa $\tau \in \mathcal{T}_s\langle t_i, t_f\rangle$ possui um custo de infraestrutura que é influenciado pelo tempo de execução (*ExecTime*), tempo de inicialização (*InitTime*) e pelos custos de alocação da infraestrutura (*Price*) associada às máquinhas virtuais alocadas. Seja *Alloc*(τ) o conjunto das máquinas virtuais alocadas para a execução da tarefa de qualidade de dados

τ. Cada $VM_v \in$ *Alloc*(τ) possui um identificador (v) único e sequencial. Seja $\gamma_i \in \gamma$ uma configuração de máquinha virtual de identificador v ($VM_v^{\gamma_i}$), então cada $VM_v^{\gamma_i} \in$ *Alloc*(τ) possui a mesma configuração γ_i e o mesmo custo de alocação *Price*($VM_v^{\gamma_i}$). O custo (\$) de infraestrutura de uma tarefa de qualidade de dados τ, empregando uma classe de configuração denotada por $\mathcal{C}l(\tau)$, é representado por $VMC\$(\tau, \mathcal{C}l(\tau))$, como definido na Eq. (5.1).

$$VMC\$(\tau, \mathcal{C}l(\tau)) = \sum_{v=1}^{|Alloc(\tau)|} \Big(ExecTime(VM_v^{\gamma_i}) + InitTime(VM_v^{\gamma_i}) \Big) \times Price(VM_v^{\gamma_i}) \tag{5.1}$$

O custo total ($C\$(\tau, \mathcal{C}l(\tau))$) associado à execução de uma tarefa de qualidade de dados τ é composto pelos custos de infraestrutura ($VMC\$(\tau, \mathcal{C}l(\tau))$) somados aos custos de penalidade, como mostrado na Eq. (5.2).

$$C\$(\tau, \mathcal{C}l(\tau)) = VMC\$(\tau) + PenaltyC\$(\tau) \tag{5.2}$$

Cada tarefa de qualidade de dados $\tau \in \mathcal{T}_s\langle t_i, t_f\rangle$ executada por um SMQD pode estar sujeita à uma restrição de tempo $T_{res}(\tau)$ que é especificada no DQSLA $s \in S_c\langle t_i, t_f\rangle$ que disparou a execução de τ. Um possível modelo [91] para representar e calcular os custos de penalidade se baseia em uma função linear, como mostrado na Eq. (5.3), tal que α representa uma penalidade fixa, β é a taxa de penalidade e DT (*delay time*) é a diferença entre o tempo de execução ($T_{exec}(\tau)$) e o tempo de restrição ($T_{res}(\tau)$) da tarefa τ. No contexto de um SMQD, os valores da penalidade fixa ($\alpha(\tau)$) e da taxa de penalidade ($\beta(\tau)$) associadas à tarefa τ são especificados no DQSLA $s \in S_c\langle t_i, t_f\rangle$ que disparou a tarefa τ, baseados no valor do parâmetro $\mathcal{P}$. Portanto, é possível definir o custo de penalidade $PenaltyC\$(\tau)$, associado à execução da tarefa τ, como mostrado na Eq. (5.4).

$$SLAPenalty = \alpha + \beta \times DT \tag{5.3}$$

$$PenaltyC\$(\tau, \mathcal{C}l(\tau)) =$$
$$\begin{cases} \alpha(\tau) + \beta(\tau) \times DT(T_{exec}(\tau), T_{res}(\tau)) & \textbf{se } DT(T_{exec}(\tau), T_{res}(\tau)) > 0 \\ 0 & \text{caso contrário} \end{cases} \tag{5.4}$$

5.2 Algoritmos para Provisionamento de Recursos Computacionais

No intuito de evitar a adoção de estratégias baseadas na sub provisão (em inglês, *under provisioning*) ou super provisão (em inglês, *over provisioning*) para a execução de tarefas de qualidade de dados ao longo do tempo em um SMQD, é possível adotar algoritmos de provisionamento de recursos computacionais para estimar classes de configuração (ou seja, *clusters* de VMs) ideais para a execução de tarefas de qualidade de dados.

Sejam τ uma tarefa de qualidade de dados e $prov\_alg$ um algoritmo de provisionamento. No contexto de um SMQD, um algoritmo de provisionamento é modelado como uma função que mapeia o conjunto de características que quantificam uma tarefa de qualidade de dados em uma classe de configuração utilizada para executar a tarefa de maneira distribuída. Desse modo, seja $\mathcal{T}$ um conjunto de tarefas de qualidade de dados, então $prov\_alg$ é modelado como uma função na forma $prov\_alg : \mathcal{T} \rightarrow \mathcal{C}_{conf}$.

Seja $prov\_alg(\tau) \in \mathcal{C}_{conf}$ uma classe de configuração estimada pelo algoritmo de provisionamento $prov\_alg$ para a execução da tarefa τ. Então, o custo total ($ServC\$^{C\langle t_i,t_f\rangle}_{prov\_alg}$) associado a um SMQD para atender $|C\langle t_i, t_f\rangle|$ clientes utilizando o algoritmo de provisionamento $prov\_alg$, durante o intervalo de tempo (t_i, t_f), é a soma dos custos totais relacionados à execução de cada tarefa de qualidade de dados disparada neste período, como definido na Eq. (5.5).

$$ServC\$^{C\langle t_i,t_f\rangle}_{prov\_alg} = \sum_{c \in C\langle t_i,t_f\rangle} \sum_{s \in S_c\langle t_i,t_f\rangle} \sum_{\tau \in \mathcal{T}_s\langle t_i,t_f\rangle} C\$(\tau, prov\_alg(\tau)) \tag{5.5}$$

5.2.1 Formalização do Problema

Nesta seção, é formalizado o problema de provisionamento de recursos computacionais no contexto de um SMQD. Dada uma tarefa de qualidade de dados τ, o problema consiste em estimar uma classe de configuração $\mathcal{C}l(\tau)$ que é capaz de minimizar a diferença entre o tempo de restrição ($T_{res}(\tau)$) definido no DQSLA que disparou a tarefa τ e o tempo de execução ($T_{exec}^{\mathcal{C}l(\tau)}(\tau)$) da tarefa, como apresentado na Tabela 5.1. Na Figura 5.2 são apresentados exemplos de tempos de execução provenientes da adoção de sub provisão, super provisão e alocação ótima de recursos computacionais para a execução de uma tarefa de qualidade de dados associada a uma restrição de tempo.

Tabela 5.1: Problema de provisionamento de recursos computacionais no contexto de um SMQD.

Dada	uma tarefa de qualidade de dados τ
Estimar	$\mathcal{C}l_{opt}(\tau)$
Entre	$\mathcal{C}_{conf} = \{\mathcal{C}l_1, \mathcal{C}l_2, \ldots, \mathcal{C}l_{m \times k}\}$
para Minimizar	$T_{res}(\tau) - T_{exec}^{\mathcal{C}l_{opt}}(\tau)$
Sujeito a	$(T_{res}(\tau) - T_{exec}^{\mathcal{C}l_{opt}}(\tau)) \geq 0$

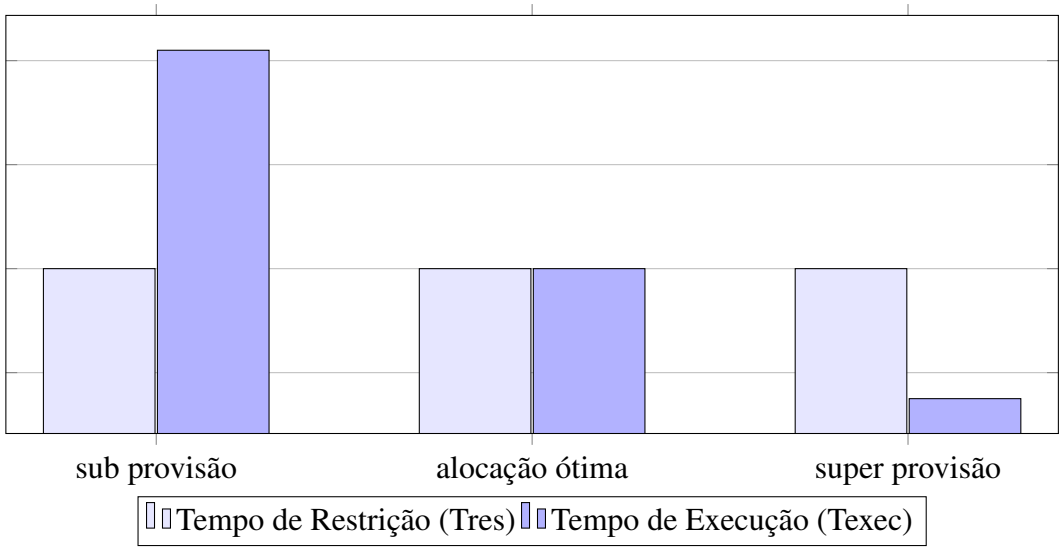

Figura 5.2: Diferenças entre o tempo de execução e o tempo de restrição produzidos pela sub provisão, alocação ótima e super provisão de recursos.

Na Figura 5.3 são ilustradas as regiões dos índices de classe de configuração que resultam em sub provisão e super provisão de recursos computacionais para a execução de uma tarefa de qualidade de dados τ. Esta intuição é formalizada por meio do Lema 5.2.1 e do Lema 5.2.2. Em seguida, com base nestes lemas, são também propostos um teorema e um corolário que são utilizados para guiar a proposição de algoritmos para lidar com o problema de provisionamento de recursos computacionais no contexto de um SMQD. As provas dos lemas e teoremas apresentados nesta seção são apresentados no Apêndice D.4.

Lema 5.2.1 *Sejam* $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_{|\mathcal{T}|}\}$ *um conjunto de tarefas de qualidade de dados,* $\mathcal{C}_{conf} = \{\mathcal{C}l_1, \mathcal{C}l_2, \cdots, \mathcal{C}l_{m\times k}\}$ *um conjunto das classes de configuração disponíveis e* alg' *um algoritmo de provisionamento. Então:*
$\exists\tau \in \mathcal{T}(alg'(\tau) \in (\mathcal{C}_{conf} \backslash \mathcal{C}l_{opt}(\tau)) \ \wedge \ T_{exec}^{alg'(\tau)}(\tau) < T_{res}(\tau)) \Rightarrow$
$\sum_{\tau\in\mathcal{T}}(C\$(\tau, alg'(\tau))) > \sum_{\tau\in\mathcal{T}}(C\$(\tau, \mathcal{C}l_{opt}(\tau)))$.

Lema 5.2.2 *Sejam* $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_{|\mathcal{T}|}\}$ *um conjunto de tarefas de qualidade de dados,* $\mathcal{C}_{conf} = \{()\mathcal{C}l_1, \mathcal{C}l_2, \cdots, \mathcal{C}l_{m\times k}\}$ *um conjunto das classes de configuração disponíveis e* alg' *um algoritmo de provisionamento. Então:*
$\exists\tau \in \mathcal{T}(alg'(\tau) \in (\mathcal{C}_{conf} \backslash \mathcal{C}l_{opt}(\tau)) \ \wedge \ T_{exec}^{alg'(\tau)}(\tau) > T_{res}(\tau)) \Rightarrow$
$\sum_{\tau\in\mathcal{T}}(C\$(\tau, alg'(\tau))) > \sum_{\tau\in\mathcal{T}}(C\$(\tau, \mathcal{C}l_{opt}(\tau)))$.

Teorema 5.2.1 *Sejam* $\mathcal{T}\langle t_i, t_f\rangle = \{\tau_1, \tau_2, \cdots, \tau_{|\mathcal{T}\langle t_i,t_f\rangle|}\}$ *um conjunto de tarefas de qualidade de dados disparadas por clientes representados por* $C\langle t_i, t_f\rangle$, $\mathcal{C}_{conf} = \{\mathcal{C}l_1, \mathcal{C}l_2, \cdots, \mathcal{C}l_{m\times k}\}$ *um conjunto das classes de configuração disponíveis, e* alg' *e* alg_{opt} *algoritmos de provisionamento. Então:*
$[\forall\tau \in \mathcal{T}\langle t_i, t_f\rangle(alg_{opt}(\tau) = \mathcal{C}l_{opt}(\tau)) \ \wedge \ \exists\tau \in \mathcal{T}\langle t_i, t_f\rangle(alg'(\tau) \in \mathcal{C}_{conf}\backslash\mathcal{C}l_{opt}(\tau))] \Rightarrow$
$ServC\$_{alg'}^{C\langle t_i,t_f\rangle} > ServC\$_{alg_{opt}}^{C\langle t_i,t_f\rangle}$.

Corolário 5.2.1 *Dado um conjunto* $\mathcal{T}$ *de tarefas de qualidade de dados, a utilização de um algoritmo de provisionamento* alg_{opt}*, tal que* $\forall\tau \in \mathcal{T}(alg_{opt}(\tau) = \mathcal{C}l_{opt}(\tau))$*, minimiza o custo total de um SMQD.*

Em resumo, no intuito de executar a tarefa de qualidade de dados τ, ao alocar um *cluster* de máquinas virtuais denotado por $\mathcal{C}l_{opt}(\tau)$, como mostrado na Tabela 5.1, um SMQD irá alocar a quantidade mínima (ideal) de recursos computacionais necessária visando respeitar

o tempo de restrição associado à tarefa τ. Por conseguinte, a utilização de um algoritmo de provisionamento ótimo acarreta na minimização dos custos de infraestrutura e de penalidades ao provedor do serviço, assim como na máxima quantidade de recursos disponíveis, o que permite ao provedor do serviço atender a um número máximo de clientes simultaneamente.

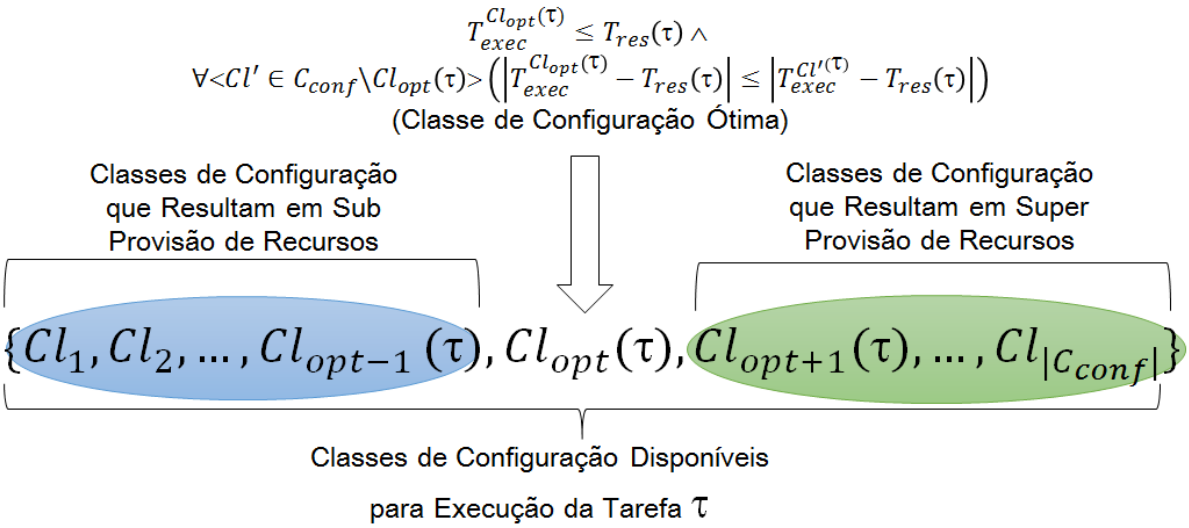

Figura 5.3: Ilustração das regiões de classes de configuração: sub provisão, alocação ótima e super provisão de recursos computacionais para a execução da tarefa τ.

5.2.2 Algoritmos de Provisionamento baseados em Aprendizado de Máquina

No intuito de desenvolver um algoritmo otimizado (alg_{opt}) de provisionamento de recursos computacionais no contexto de um SMQD, como descrito no Corolário 5.2.1, são propostos algoritmos de provisionamento baseados em heurísticas e técnicas de aprendizado de máquina. No segundo caso, o problema é modelado como um sistema de decisão para tarefas de deduplicação de dados.

Definição 5.2.1 (Sistema de Decisão para Tarefas de Deduplicação de Dados) *Um sistema de decisão*[1] *para tarefas de deduplicação de dados é modelado como* $DS_{dedup} = (TD, \mathcal{A}_{dedup} \cup \{Cl\})$*, tal que* Cl *representa um atributo de decisão (uma classe de configuração pertencente ao conjunto* $\mathcal{C}_{conf}$*) e* $\mathcal{A}_{dedup}$ *é um conjunto formado pelos atributos {*Λ*,* η*,*

[1] adaptada da definição de Sistema de Decisão em [82]

T_{exec}}, sendo que para cada $t \in TD$: $\Lambda(t)$ é a quantidade estimada de comparações entre entidades a ser realizada pela tarefa t, $\eta(t)$ é o custo (tempo de execução) de comparar duas entidades na base de dados e $T_{exec}(t)$ é o tempo de execução da tarefa de deduplicação de dados t utilizando um cluster de máquinas virtuais denotado pela classe de configuração $\mathcal{C}l(t) \in \mathcal{C}_{conf}$.

Desse modo, para uma nova instância de tarefa de deduplicação de dados $\tau\langle\Lambda(\tau)$, $\eta(\tau)$, $T_{res}(\tau)\rangle$, um algoritmo de provisionamento visa estimar $\mathcal{C}l_{opt}(\tau) \in \mathcal{C}_{conf}$, com base nos valores das instâncias na base de treinamento $t\langle\Lambda(t)$, $\eta(t)$, $T_{res}(t)\rangle$, tal que $t \in TD$. No Exemplo 5.2.1, é apresento um cenário de classificação para o processo de estimativa de classe de configuração para execução uma tarefa de deduplicação de dados.

Exemplo 5.2.1 *Seja* $TD'\langle\Lambda, \eta, T_{exec}\rangle = \{\langle 1000, 0.1, 200, \mathcal{C}l_{10}\rangle$, $\langle 600, 0.2, 400, \mathcal{C}l_4\rangle$, $\langle 300, 0.2, 600, \mathcal{C}l_3\rangle$, $\langle 2000, 0.1, 400, \mathcal{C}l_{20}\rangle, \langle 4000, 0.1, 300, \mathcal{C}l_{32}\rangle\}$ *uma base de treinamento de um Sistema de Decisão para Tarefas de Deduplicação de Dados (ver Definição 5.2.1). Seja* $\tau'\langle\Lambda, \eta, T_{res}\rangle$ *uma tarefa de deduplicação de dados representada pela tupla* $\langle 350, 0.2, 700\rangle$. *Considerando o algoritmo de aprendizado de máquina* $1NN$, *ao processar a base de treinamento* TD' *no intuito de estimar uma classe de configuração para executar a tarefa* τ', *é obtido o seguinte resultado:* $1NN(TD', \tau') = \mathcal{C}l_3$, *uma vez que* $\langle 300, 0.2, 600, \mathcal{C}l_3\rangle$ *pertencente a* TD' *é a tupla mais similar a* τ'. *Desse modo, um cluster de máquinas virtuais, associado a uma classe de configuração definida por* $\mathcal{C}l_3$, *seria utilizado para executar* τ'.

Existem dois cenários distintos em que um algoritmo de provisionamento de recursos computacionais no contexto de um SMQD é aplicado: i) o algoritmo de provisionamento precisa estimar uma classe de configuração para a execução de uma tarefa de deduplicação de dados cujo DQSLA já disparou uma ou mais tarefas anteriormente e os dados referentes aos tempos de execução passados estão disponíveis; e ii) o algoritmo de provisionamento precisa estimar uma classe de configuração para a execução de uma tarefa de deduplicação de dados cujo DQSLA não disparou nenhuma tarefa anteriormente. Na prática, o cenário *i* representa um problema mais simples, pois os dados das execuções anteriores do mesmo DQSLA facilitam consideravelmente a descoberta da classe de configuração ideal para executar a nova tarefa disparada pelo mesmo DQSLA. Por este motivo, neste trabalho foram

desenvolvidas estratégias distintas para lidar com cada um destes cenários (*i* e *ii*) separadamente. Na Seção 5.2.3, são apresentadas as estratégias propostas para lidar com o cenário *i*. Por sua vez, na Seção 5.2.4, são apresentadas as estratégias propostas para lidar com o cenário *ii*.

5.2.3 Algoritmos de Provisionamento para Processamento de Tarefas Disparadas por um Único DQSLA

Para lidar com o problema de provisionamento de recursos computacionais no cenário em que tarefas de deduplicação de dados são subsequentemente disparadas por um mesmo DQSLA, é proposta uma estratégia baseada em uma metaheurística, ou seja, um algoritmo que emprega várias heurísticas para a resolução de um problema. A metaheurística proposta emprega uma técnica de busca denominada *Hill Climbing* [48], a qual se baseia em uma solução inicial para algum problema e realiza ajustes subsequentes sobre a solução, considerando que é possível saber qual a "direção" da solução ideal dada a qualidade da solução atual, no intuito de encontrar a solução ótima para o problema.

O pseudocódigo do algoritmo *Hill Climbing Provisioning* é representado no Algoritmo 2. Inicialmente (linha 2), o algoritmo verifica se o SMQD já realizou alguma execução anterior disparada pelo mesmo DQSLA que gerou a execução da tarefa τ. Em caso positivo (linhas 4 a 6), o algoritmo faz uma chamada intercambiável ($InitialConfigurationClass$), i.e., uma chamada que pode ser substituída por diferentes heurísticas para gerar diferentes algoritmos, visando estimar uma classe de configuração inicial para executar a tarefa τ (linha 4), aloca um *cluster* de máquinas virtuais baseado na classe estimada para executar τ (linha 5) e atualiza a base de treinamento (linha 6) utilizando os dados provenientes do plano de execução alocado (esta atualização visa tornar a base de treinamento mais completa ao longo do tempo, o que tende a contribuir para que os algoritmos de provisionamento possam gerar estimativas de classes de configuração mais efetivas). Em caso negativo (linha 8), caso o tempo de execução ($T^P_{exec}(\tau)$) da tarefa previamente executada pelo mesmo DQSLA que disparou a tarefa τ não seja uma solução ideal (ou seja, a expressão $((T_{res}(\tau) - T^P_{exec}(\tau)) > thr$ **ou** $T^P_{exec}(\tau) > T_{res}(\tau))$ é avaliada como verdadeira), então o algoritmo cria uma classe de configuração ajustada ($\mathcal{C}l_T$), utilizando a chamada intercambiável $TweakSolution$ (linha

11), com base na classe de configuração existente.

Em seguida, um *cluster* de máquinas virtuais baseado na classe de configuração ajustada é utilizado para executar τ (linha 12) e a base de treinamento é atualizada (linha 13) utilizando os dados provenientes da execução de τ empregando a classe de configuração ajustada. Então, é verificado se o tempo de execução da tarefa τ aproximou-se mais do tempo de restrição associado à tarefa ($T_{res}(\tau)$), em comparação com o tempo de execução da tarefa previamente disparada pelo mesmo DQSLA que disparou a tarefa τ. Em outras palavras, é verificado se a expressão ($|T_{exec}(\tau) - T_{res}(\tau)| < |T^{P}_{exec}(\tau) - T_{res}(\tau)|$) é avaliada como verdadeira (linha 14). Em caso positivo, é verificado se tanto o tempo de execução da tarefa anterior (associada ao mesmo DQSLA que disparou a tarefa τ) quanto o tempo de execução da tarefa τ são menores que o valor de $T_{res}(\tau)$ (linha 15), ou se a execução da tarefa τ passou a não gerar penalidades para o serviço em comparação com a execução da tarefa anteriormente disparada pelo mesmo DQSLA (linha 17). Se algum desses casos acontecer, a classe de configuração da tarefa τ passa a ser a classe de configuração ajustada ($\mathcal{C}l_T$, na linha 16 ou 18). Um exemplo do funcionamento da metaheurística *Hill Climbing Provisioning* é apresentado na Figura 5.4, na qual são ilustrados os três passos realizados pelo metaheurística: i) estimativa de uma classe de configuração inicial (i.e., $\mathcal{C}l_4$); ii) a solução inicial estimada é avaliada; e iii) a solução avaliada é adaptada para a classe $\mathcal{C}l_{10}$ na próxima ocorrência de disparo de execução do DQSLA. No Apêndice D.2 é apresentada uma prova de convergência da metaheurística *Hill Climbing Provisioning*.

Note que é possível modificar as chamadas intercambiáveis (linha 4 e linha 11) da metaheurística proposta no intuito de gerar algoritmos de provisionamento diferentes baseados na técnica *Hill Climbing*. Desse modo, a seguir são propostos diferentes algoritmos para a implementação das chamadas intercambiáveis da metaheurística proposta.

Heurísticas para Ajuste de Classe de Configuração

O pseudocódigo da heurística *Sliced Tweak Solution* é mostrado no Algoritmo 3. Esta heurística realiza um ajuste aleatório (para realizar as tarefas de *scaling up* ou *scaling down*) na classe de configuração de entrada (Cl_i), dependendo da diferença entre os valores de $T_{exec}(\tau)$ e $T_{res}(\tau)$. Além disso, a heurística também utiliza duas variáveis persistentes (*InferiorIndexLimit*(τ) e *SuperiorIndexLimit*(τ)) para evitar um ajuste desnecessariamente maior

Algoritmo 2: Hill Climbing Provisioning

input : τ: uma tarefa de deduplicação de dados

thr: um limiar (positivo) mínimo utilizado para avaliar alocações classificadas como super provisão

L: o índice da última classe de configuração disponível ($L = |\mathcal{C}_{conf}|$)

TD: a base de treinamento

```
1  begin
2      if (Cl(τ) = Nil) then
3          // interchangeable call
4          Cl(τ) ⟵ InitialConfigurationClass(τ, L)
5          T_exec(τ) ⟵ allocate(τ, Cl_I)
6          TD ⟵ TD ∪ ⟨Λ(τ), η(τ), T_exec(τ), Cl(τ)⟩
7      // T^P_exec(τ) é o tempo de execução associado à tarefa
           anterior que foi disparada pelo mesmo DQSLA que
           disparou a tarefa τ
8      else if ((T_res(τ) − T^P_exec(τ)) > thr or T^P_exec(τ) > T_res(τ)) then
9          // interchangeable call
10         // T é ajustado para algum valor entre i+1 e |C_conf|
               (se T^{Cl_i(τ)}_exec > T_res(τ)) ou entre 1 e i−1 (se
               (T_res(τ) − T^{Cl_i(τ)}_exec) > thr)
11         Cl_T ⟵ TweakSolution(Cl(τ), τ, L)
12         T_exec(τ) ⟵ allocate(τ, Cl_T)
13         TD ⟵ TD ∪ ⟨Λ(τ), η(τ), T_exec(τ), Cl_T⟩
14         if (|T_exec(τ) − T_res(τ)| < |T^P_exec(τ) − T_res(τ)|) then
15             if ((T^P_exec(τ) < T_res(τ)) and (T_exec(τ) < T_res(τ))) then
16                 Cl(τ) ⟵ Cl_T
17             else if (T^P_exec(τ) > T_res(τ)) then
18                 Cl(τ) ⟵ Cl_T
19     else
20         allocate(τ, Cl(τ))
```

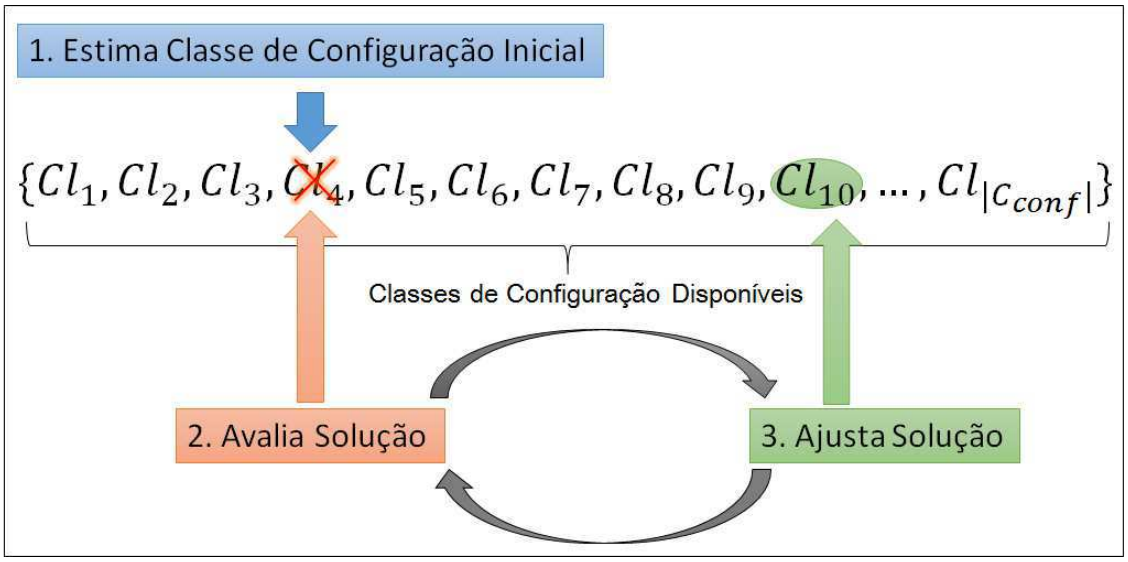

Figura 5.4: Ilustração do funcionamento do algoritmo *Hill Climbing Provisioning*.

que o necessário sobre a classe de configuração recebida como entrada (Cl_i).

Um exemplo de funcionamento da heurística *Sliced Tweak Solution* é apresentado na Figura 5.5, na qual é ilustrado que, após a classe de configuração ser ajustada de $\mathcal{C}l_{10}$ para $\mathcal{C}l_5$ (ajuste inicial), caso seja necessária uma próxima adaptação na classe de configuração para executar uma tarefa disparada por um mesmo DQSLA, a classe de configuração $\mathcal{C}l_{10}$ irá se tornar o limite superior para realizar tal ajuste.

Os demais algoritmos propostos ajuste de classe de configuração são apresentados na Seção C.1 (Apêndice C). Por sua vez, os algoritmos para estimativa de classe de configuração inicial são apresentados na Seção C.2 (Apêndice C).

5.2.4 Algoritmos de Provisionamento para Processamento de Tarefas Disparadas Inicialmente por Múltiplos DQSLAs

Nesta seção, é apresentada uma heurística para tratar o problema de processar uma carga de trabalho contendo um conjunto de tarefas de deduplicação de dados, tal que existe uma restrição de tempo associada a cada tarefa e todas as tarefas da carga de trabalho são disparadas por DQSLAs diferentes. Esta segunda característica da carga de trabalho torna o problema particularmente desafiador, uma vez que dados sobre a execução de tarefas disparadas anteriormente por um mesmo DQSLA não estão disponíveis. Por este motivo, para processar uma

Algoritmo 3: Sliced Tweak Solution

input : Cl_i: uma classe de configuração de entrada

τ: uma tarefa de deduplicação de dados

L: o índice da última classe de configuração disponível ($L = |\mathcal{C}_{conf}|$)

output: Cl_o: uma classe de configuração de saída

1 **persistent** *InferiorIndexLimit(τ)*

2 **persistent** *SuperiorIndexLimit(τ)*

3 **begin**

4 **if** $(InferiorIndexLimit(\tau) = Nil)$ **then**

5 $InferiorIndexLimit(\tau) \longleftarrow 1$

6 **if** $(SuperiorIndexLimit(\tau) = Nil)$ **then**

7 $SuperiorIndexLimit(\tau) \longleftarrow L$

8 **if** $(T_{exec}(\tau) > T_{res}(\tau))$ **then**

9 `// scaling up`

10 $Cl_R \longleftarrow randomClass(i+1, SuperiorIndexLimit(\tau))$

11 $InferiorIndexLimit(\tau) \longleftarrow i+1$

12 **return** Cl_R

13 **else if** $(T_{exec}(\tau) < T_{res}(\tau))$ **then**

14 `// scaling down`

15 $Cl_R \longleftarrow randomClass(InferiorIndexLimit(\tau), i-1)$

16 $SuperiorIndexLimit(\tau) \longleftarrow i-1$

17 **return** Cl_R

18 **return** Cl_i

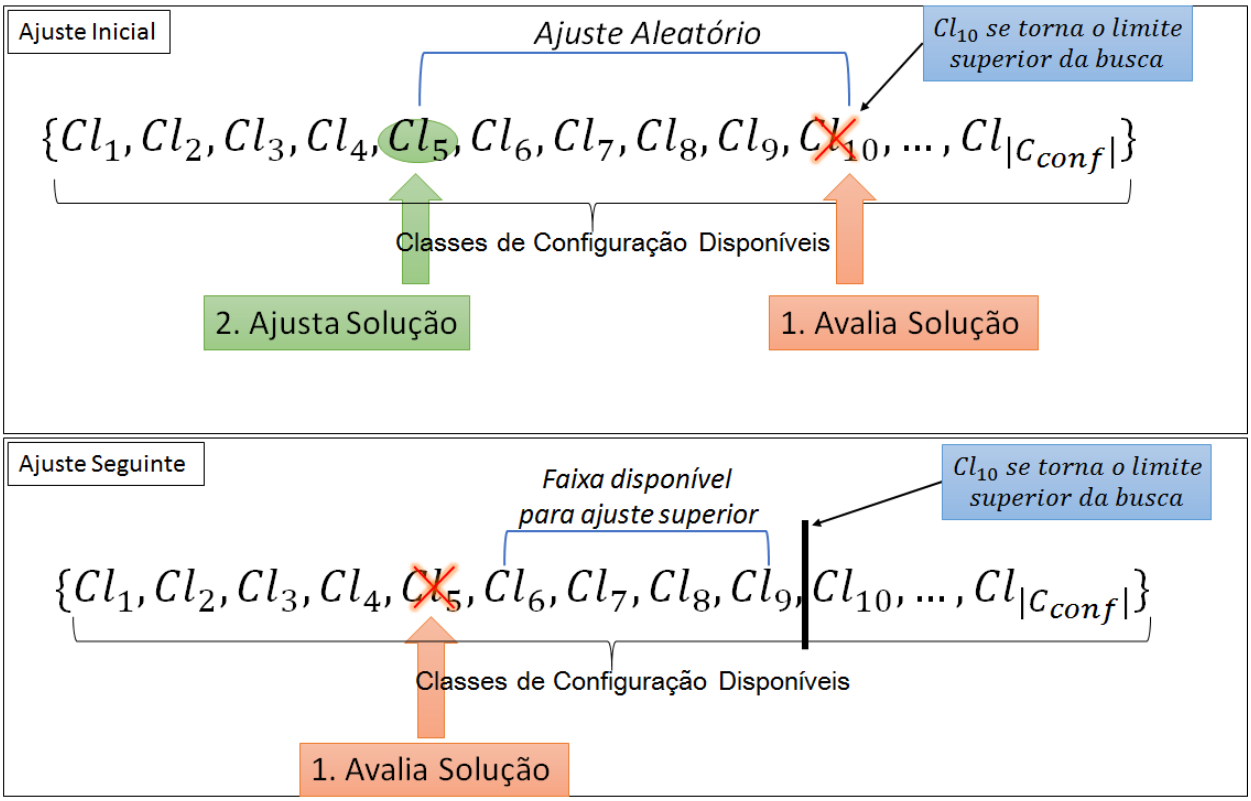

Figura 5.5: Ilustração do funcionamento do algoritmo *Sliced Tweak Solution*.

nova tarefa de deduplicação de dados disparada por um DQSLA, os algoritmos de provisionamento de recursos computacionais devem se basear apenas em i) uma base de treinamento inicialmente disponível; e/ou ii) informações dos tempos de execução e das características provenientes de tarefas de deduplicação de dados disparadas por outros DQSLAs.

Heurística *Best Performing Allocation*

A heurística *Best Performing Allocation*, representada no Algoritmo 4, visa escolher um algoritmo de provisionamento de recursos computacionais (dentre os algoritmos disponíveis no parâmetro $algSet$) com base na quantidade de vezes em que os algoritmos de provisionamento produziram alocações classificadas como sub provisão ou super provisão, ou seja, alocações cujos resultados produziram resultados significantemente diferentes dos resultados ótimos (de acordo com a Tabela 5.1). Para tal, a heurística *Best Performing Allocation* inicialmente atribui um valor de contagem (armazenado no mapa $algScore$) igual a zero para cada algoritmo de provisionamento (linhas 2-4), seleciona o algoritmo ($chosenAlg$) que possui o maior valor de contador (linha 6) e utiliza o algoritmo de provisionamento escolhido

para estimar (linha 7) uma classe de configuração inicial $\mathcal{C}l(\tau)$ para executar (linha 8) a tarefa τ. Após a execução da tarefa τ ser completada, a base de treinamento é atualizada (linha 9) com os dados da execução. Em seguida, o contador associado ao algoritmo $chosenAlg$ pode ser penalizado (ou seja, diminuído de 1) caso sua estimativa de classe de configuração tenha produzido alocações classificadas como sub provisão (linha 10) ou super provisão (linha 12).

Prosseguindo, se a classe de configuração estimada por $chosenAlg$ produziu um resultado ótimo ou próximo de ótimo (ou seja, o valor da expressão $T_{res}(\tau) \geq T_{exec}(\tau) \geq (T_{res}(\tau) - overThr)$ é avaliado como verdadeiro), então o contador associado a $chosenAlg$ pode ser incrementado em 1, dependendo do valor do parâmetro $optInc$. Note que ambos os valores $\{0, 1\}$ que podem ser associados ao parâmetro $optInc$ podem acarretar em vantagens e desvantagens. Se o valor 1 é atribuído ao parâmetro $optInc$, então o algoritmo *Best Performing Allocation* pode empregar, por períodos maiores de tempo, algoritmos de provisionamento que produziram anteriormente resultados ótimos (ou próximos do ótimo). No entanto, quando estes algoritmos passarem a produzir alocações não ótimas, uma vez que os contadores associados a estes algoritmos estarão com valores altos, demorará um tempo maior até que estes algoritmos sejam substituídos por outros algoritmos de provisionamentos que possuírem os respectivos contadores com valores menores.

Por outro lado, se for atribuído o valor 0 ao parâmetro $optInc$ (visando minimizar o problema causado pela atribuição do valor 1 ao mesmo parâmetro), o algoritmo *Best Performing Allocation* irá empregar por períodos muito mais curtos os algoritmos de provisionamento que produzem alocações ótimas. Este fato é particularmente problemático quando um algoritmo de provisionamento específico (alg_z) é a opção de algoritmo de provisionamento ideal para uma certa sequência de tarefas de deduplicação de dados. Se este for o caso, então independentemente da quantidade de vezes que alg_z produza alocações ótimas, caso este algoritmo produza uma única alocação não ótima, todos os outros algoritmos no conjunto $(algSet \backslash alg_z)$ poderão ser empregados antes que alg_z seja escolhido novamente. Assim, uma vez que os algoritmos de provisionamento em $(algSet \backslash alg_z)$ tendem a produzir alocações não ótimas para este caso particular, então no pior caso serão necessárias $|algSet|-1$ alocações não ótimas para que alg_z seja escolhido novamente. Portanto, é importante avaliar empiricamente ambas as opções $\{optInc = 0, optInc = 1\}$ para o parâmetro $optInc$ por meio de investigações experimentais.

Algoritmo 4: Best Performing Allocation

input : τ: uma tarefa de deduplicação de dados

$algSet$: um conjunto de algoritmos de aprendizado de máquina

$algScore$: uma mapa contendo contadores associados aos algoritmos

$overThr$: um limiar (positivo) utilizado para avaliar alocações classificadas como super provisão

$(TD, \mathcal{A}_{dedup} \cup \{\mathcal{C}l\})$: um sistema de decisão para tarefas de deduplicação de dados

$optInc$: um parâmetro {0, 1} para incrementar os contadores associados aos algoritmos nos casos de alocação ótima

```
1  begin
2      foreach mlAlg in algSet do
3          if (algScore[mlAlg] = Nil) then
4              algScore[mlAlg] ← 0
5      // a random choice is made in case of ties
6      chosenAlg ← arg max_{mlAlg ∈ algSet} (algScore[mlAlg])
7      Cl(τ) ← chosenAlg((TD, A_dedup ∪ {Cl}), τ)
8      allocate(τ, Cl(τ))
9      TD ← TD ∪ ⟨Λ(τ), η(τ), T_exec(τ), Cl(τ)⟩
10     if ((T_exec(τ) > T_res(τ))) then
11         algScore[chosenAlg] ← algScore[chosenAlg] − 1
12     else if (T_res(τ) − T_exec(τ)) > overThr then
13         algScore[chosenAlg] ← algScore[chosenAlg] − 1
14     else
15         algScore[chosenAlg] ← algScore[chosenAlg] + optInc
```

Um exemplo de funcionamento do algoritmo *Best Performing Allocation* é apresentado na Figura 5.6, na qual é apresentado um conjunto de algoritmos de provisionamento ($\{alg_1, alg_2, alg_3, alg_4, alg_5\}$) e suas respectivas pontuações iniciais. Em seguida, é mostrado o fluxo de funcionamento da heurística: i) é selecionado o algoritmo contendo a maior pontuação (i.e., alg_2); ii) a solução estimada por alg_2 é avaliada (i.e., é avaliado se a configuração de $cluster$ estimada por alg_2 produziu super provisão, sub provisão ou alocação ótima de recursos); e iii) **a.** caso a configuração tenha produzido super provisão ou sub provisão de recursos, a pontuação associada ao algoritmo alg_2 é atualizada para $alg_2.score - 1$ ou **b.** caso a configuração tenha produzido alocação ótima de recursos, a pontuação associada ao algoritmo alg_2 é atualizada para $alg_2.score + optInc$, tal que $optInc = 0$ ou 1.

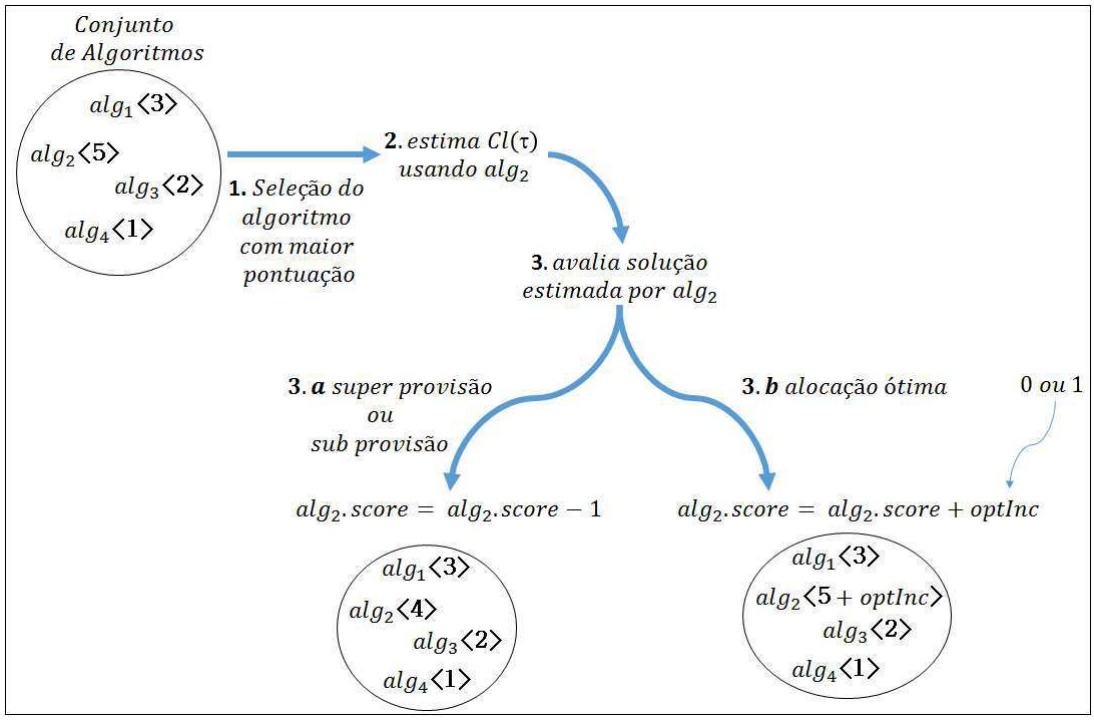

Figura 5.6: Ilustração do funcionamento do algoritmo *Best Performing Allocation*.

As demais heurísticas propostas para seleção de algoritmos de provisionamento são apresentadas na Seção C.3 (Apêndice C)

Abordagens Básicas baseadas em Super Provisão

Neste trabalho, são também empregadas duas abordagens básicas para estimar classes de configuração para a execução de tarefas de deduplicação de dados, ambas baseadas em super

provisão. A primeira abordagem básica, denominada $Over_{100}$, simplesmente retorna a classe de configuração que possui o maior índice dentre as classes de configuração disponíveis no SMQD. Esta ideia é formalizada da seguinte maneira:

$$Over_{100}((TD, \mathcal{A}_{dedup} \cup \{\mathcal{C}l\}, \tau) = \mathcal{C}l_{|\mathcal{C}_{conf}|} \tag{5.6}$$

Por sua vez, a segunda abordagem básica ($Over_{75}$) adota uma estratégia baseada em super provisão que sempre emprega uma classe de configuração cujo índice corresponde a 75% do índice empregado pela abordagem $Over_{100}$. Em outras palavras, a abordagem $Over_{75}$ aloca um *cluster* de máquinas virtuais com uma capacidade de processamento ligeiramente inferior (25% a menos) em relação ao *cluster* empregado pela abordagem $Over_{100}$. Esta ideia é formalizada da seguinte maneira:

$$Over_{75}((TD, \mathcal{A}_{dedup} \cup \{\mathcal{C}l\}, \tau) = \mathcal{C}l_{\lceil 0.75 \times |\mathcal{C}_{conf}| \rceil} \tag{5.7}$$

Neste trabalho, abordagens baseadas em sub provisão não são empregadas, uma vez que na prática estas abordagens gerariam penalidades muito significativas para um SMQD (ver Eq. (5.4)).

5.3 Considerações Finais

Neste capítulo, foi proposto um modelo de custo para um SMQD e apresentada a formalização do problema de alocação de recursos computacionais para a execução de tarefas de qualidade de dados associadas à restrições de tempo. Foram também propostos vários algoritmos para provisionamento dinâmico de recursos computacionais no contexto de um SMQD. Os algoritmos propostos são baseados em heurísticas e técnicas de aprendizado de máquina. A ideia geral dos algoritmos baseados em heurísticas consiste em adaptar a classe de configuração utilizada para executar uma tarefa de qualidade de dados, com base em resultados de outras tarefas de qualidade de dados disparadas pelo mesmo DQSLA. Por sua vez, os algoritmos baseados em aprendizado de máquina utilizam dados provenientes de uma base de treinamento para tentar estimar a classe de configuração ótima (ou seja, que

minimiza os custos de execução) para a execução de uma tarefa de qualidade de dados. A avaliação dos algoritmos propostos neste capítulo é apresentada no Capítulo 8.

No capítulo seguinte, são propostas novas métricas, algoritmos e heurísticas para o contexto de deduplicação incremental de dados, o qual consiste em outra abordagem que pode ser adotada por um SMQD para a redução de custos provenientes da execução de tarefas de qualidade de dados.

Capítulo 6

Estratégias para Deduplicação Incremental de Dados

Como explicado no Capítulo 2, o processo de Deduplicação Incremental de Dados (DID) no contexto de classificação coletiva visa atualizar apenas um subconjunto dos agrupamentos que são afetados por mudanças nos dados ao longo do tempo e, dessa forma, reduzir os custos da deduplicação de dados ao evitar reprocessar os agrupamentos de todas as entidades na base de dados.

Neste capítulo, são propostas e descritas definições e métricas no contexto de deduplicação incremental de dados utilizando classificação coletiva. As métricas propostas visam contemplar aspectos não explorados e/ou minimizar desvantagens relacionados às métricas existentes no estado da arte. Por exemplo, as métricas visam calcular o grau de estabilidade nos resultados de eficácia e eficiência produzidos por algoritmos de DID. Em seguida, são propostas novas heurísticas para a contexto do problema investigado, as quais visam executar o processo de DID de maneira mais eficiente. Por fim, é proposta uma metaheurística utilizada para gerar diferentes métodos de deduplicação incremental de dados no contexto de classificação coletiva. As abordagens propostas neste capítulo podem ser incorporadas a um SMQD, utilizando dados provenientes do módulo *Online Metadata Monitoring* mostrado na arquitetura proposta na Figura 4.1, visando reduzir custos associados à execução de tarefas de deduplicação de dados utilizando uma abordagem incremental para avaliar a qualidade dos dados. A avaliação das métricas e algoritmos propostos neste capítulo é apresentada no Capítulo 8.

6.1 Métricas para Avaliação de Algoritmos Incrementais de Deduplicação de Dados

No contexto do problema de Deduplicação Incremental de Dados (DID), é desejável que um algoritmo de DID produza resultados aceitáveis, tanto de eficiência (tempo de execução) quanto de eficácia (precisão e cobertura dos resultados), ao longo do tempo. Por esta razão, nesta seção são propostas definições e métricas relacionadas aos resultados de eficiência e eficácia que são produzidos pelos algoritmos de DID.

Para facilitar a proposição de métricas e heurísticas apresentadas nesta seção, as seguintes notações são adotadas. Seja G_k um grafo de similaridade, f um método de DID e $\mathcal{L}_{G_k}$ um agrupamento em G_k. Então:

- $f(G_k, \mathcal{L}_{G_k}, \Delta G_k) = \mathcal{L}_{G_{k+1}}$, ou seja, ao utilizar o método de DID f para atualizar o conjunto de agrupamentos $\mathcal{L}_{G_k}$, após o processamento do conjunto de atualizações ΔG_k, é produzido o conjunto de agrupamentos $\mathcal{L}_{G_{k+1}}$;
- $f(G,\mathcal{L}_G, \Delta G) = \mathcal{L}_{G+\Delta G}$, ou seja, ao utilizar o método de DID f para atualizar o conjunto de agrupamentos $\mathcal{L}_G$, após o processamento do conjunto de atualizações ΔG, é produzido o conjunto de agrupamentos $\mathcal{L}_{G+\Delta G}$.

Métricas de Eficácia

Em [33], os autores propuseram duas maneiras distintas para calcular precisão (Pr) e cobertura (Rc) dos resultados dos agrupamentos de grafos de similaridade: i) medindo a diferença entre os agrupamentos resultantes e seus respectivos agrupamentos mais próximos no agrupamento de referência (gabarito do resultado); e ii) contando a quantidade de duplicatas identificadas. Uma vez que, na prática, a segunda estratégia é mais precisa (pois evita que mais de um agrupamento produzido por um algoritmo seja comparado com o mesmo agrupamento no conjunto de agrupamentos de referência e, assim, seja gerada uma avaliação imprecisa de eficácia), a mesma foi adotada neste trabalho. Além disso, dado que não foi proposta uma métrica relacionada à cobertura de agrupamentos em [33], neste trabalho é também proposta uma métrica para este fim.

Seja $\mathbf{D}$ uma base de dados a ser continuamente deduplicada, G seu grafo de similaridade correspondente e $\mathcal{L}_G = \{C_1, C_2, \ldots, C_{|\mathcal{L}_G|}\}$ o conjunto de agrupamentos sobre G. Seja $\mathcal{G} = \{g_1, g_2, \ldots, g_{|\mathcal{G}|}\}$ o conjunto de agrupamentos de referência (gabarito), tal que $\bigcap_{g \in \mathcal{G}} g = \emptyset$ e $\forall \langle C \in \mathcal{L}_G \rangle \forall \langle v \in C \rangle \exists \langle g \in \mathcal{G} \rangle (v \in g)$. O agrupamento de referência do vértice v é denotado por $g(v)$, ou seja, $v \in g(v)$ e $g(v) \in \mathcal{G}$.

Seguindo a abordagem proposta por [33], a precisão dos agrupamentos (CPr) pode ser calculada como a precisão média de todos os agrupamentos contidos em $\mathcal{L}_G$ que possuem pelo menos dois vértices. Por sua vez, o cálculo de CPr para um agrupamento $C \in \mathcal{L}_G$, denotado por CPr_C, é definido na Eq. (6.1).

$$CPr_C = \frac{|(v_1, v_2) \in (C \times C) \mid v_1 \neq v_2 \wedge g(v_1) = g(v_2)|}{|C| \times (|C| - 1)} \tag{6.1}$$

Seja $a \in \mathbf{N}^*$ e $B = \{b_1, b_2, \ldots, b_{|B|}\}$, tal que $\forall \langle b \in B \rangle (b \in \mathbf{N}^*)$, $|B| > 1$ e $\sum_{b \in B} b = a$. Então: $\forall \langle a \in \mathbf{N}^* \rangle (a(a-1) > \sum_{b \in B} b(b-1))$ (prova no Apêndice D.4). Uma vez que essa inequação é sempre verdade, então a Eq. (6.1) penaliza o valor resultante de CPr_C cada vez que um agrupamento $C \in \mathcal{L}_G$ contiver um par de vértices não duplicados, ou seja, $\exists \langle (v_1, v_2) \in (C \times C) \rangle (v_1 \neq v2 \wedge g(v_1) \neq g(v_2))$.

De maneira similar a CPr, neste trabalho a avaliação da cobertura dos agrupamentos (CRc) é calculada dividindo a quantidade de pares duplicados corretamente identificados em todos os agrupamentos pela quantidade esperada de pares duplicados corretamente identificados em todos os agrupamentos, como mostrado na Eq. (6.2).

$$CRc = \sum_{C \in \mathcal{L}_G, |C| > 1} \frac{|(v_1, v_2) \in (C \times C) \mid v_1 \neq v_2 \wedge g(v_1) = g(v_2)|}{\sum_{g \in \mathcal{G}} |g| \times (|g| - 1)} \tag{6.2}$$

Desse modo, o cálculo da média harmônica entre precisão e cobertura dos agrupamentos gerado pode ser calculado como: $Fmeasure = \frac{2 \times CPr * CRc}{CPr + CRc}$.

Exemplo 6.1.1 *Considerando a Figura 6.1(a), na qual símbolos iguais significam entidades duplicadas, os agrupamentos resultantes produzem* $CPr = (\frac{20}{5\times4} + \frac{12}{4\times3} + \frac{12}{4\times3} + \frac{6}{3\times2} + \frac{2}{2\times1} + \frac{2}{1\times1}) \times \frac{1}{6} = 1$, $CRc = \frac{5\times4+4\times3+4\times3+3\times2+2\times1+2\times1}{6\times5+7\times6+7\times6} = \frac{20+12+12+6+2+2}{30+42+42} = \frac{54}{114} = 0,47$

e $F\text{-}measure = 2 \times \frac{1\times0,47}{1+0,47} = 0,64$. *Por sua vez, considerando a Figura 6.1(b), os agrupamentos resultantes produzem* $CPr = (\frac{4\times3+2\times1+3\times2+3\times2+3\times2+3\times2}{18\times17} + \frac{3\times2}{3\times2} + \frac{4\times3}{4\times3} + \frac{2\times1}{2\times1} + \frac{2\times1}{2\times1} + \frac{2\times1}{2\times1}) \times \frac{1}{6} = (0,12 + 1 + 1 + 1 + 1 + 1) \times \frac{1}{6} = \frac{0,12+5}{6} = 0,87$, $CRc = \frac{4\times3+2\times1+3\times2+3\times2+3\times2+3\times2+4\times3+2\times1+2\times1+2\times1}{18\times17+3\times2+4\times3+2\times1+2\times1+2\times1} = 0,19$ *e* $F\text{-}measure = 2 \times \frac{0,87\times0,19}{0,87+0,19} = 0,31$.

Note que é possível produzir resultados de agrupamentos nos quais $|\mathcal{L}_G| \neq |\mathcal{G}|$ e ainda gerar $CPr = 1$. Os agrupamentos mostrados na Figura 6.1(a) ilustram esta situação. No intuito de gerar valores resultantes de precisão elevados para resultados deste tipo, considerando que os resultados da Figura 6.1(a) representam agrupamentos de baixa qualidade, os autores em [33] também propuseram uma métrica denominada precisão de agrupamentos penalizada ($PCPr$), computada como $PCPr = \frac{|\mathcal{L}_G|}{|\mathcal{G}|} \times CPr$ (se $|\mathcal{G}| > |\mathcal{L}_G|$) ou $PCPr = \frac{|\mathcal{G}|}{|\mathcal{L}_G|} \times CPr$ (se $|\mathcal{L}_G| \geq |\mathcal{G}|$), a qual visa penalizar CPr quando $|\mathcal{L}_G| \neq |\mathcal{G}|$.

Exemplo 6.1.2 *Considerando a Figura 6.1(a), os agrupamentos resultantes produzem* $CPr = 1$ *(ver Exemplo 6.1.1) e* $PCPr = CPr \times \frac{|\mathcal{G}|}{|\mathcal{L}_G|} = 1 \times \frac{3}{6} = 0,5$.

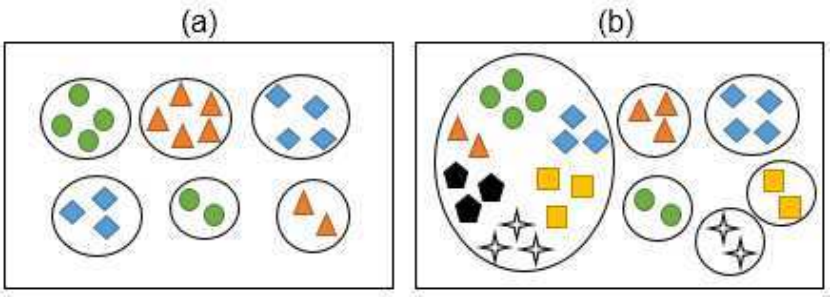

Figura 6.1: (a) Resultados de agrupamentos em que $CPr = 1$, ainda que $|\mathcal{L}_G| \neq |\mathcal{G}|$. (b) Resultados de agrupamentos em que $PCPr$ é alto $(0,87)$, porém a precisão dos agrupamentos é bastante baixa.

Por outro lado, note que a métrica $PCPr$ ainda apresenta uma consequência indesejada. Na prática, é possível gerar um valor elevado da métrica $PCPr$ (quando $|\mathcal{L}_G| \approx |\mathcal{G}|$), ainda que a precisão dos agrupamentos resultantes não seja alta. Este cenário é ilustrado na Figura 6.1(b), na qual existe um agrupamento grande com precisão baixa ($CPr_C = 0,12$) e uma quantidade considerável de agrupamentos menores produzindo $CPr_C = 1$. Neste exemplo,

$CPr = PCPr = 0,87$ (porque $|\mathcal{L}_G| = |\mathcal{G}|$), o que representa um valor de precisão bastante elevado, considerando os resultados de baixa qualidade gerados pelos agrupamentos mostrados na Figura 6.1(b). Além disso, quanto maior a quantidade de agrupamentos pequenos que apresentem $CPr_C \approx 1$, maior será o valor da precisão ($PCPr$) resultante (quando $|\mathcal{L}_G| \approx |\mathcal{G}|$). No intuito de eliminar a consequência indesejada associada à métrica $PCPr$, neste trabalho é proposta outra métrica de precisão de agrupamentos, denominada precisão de agrupamentos ponderada ($wCPr$), a qual computa a média ponderada da precisão dos agrupamentos levando em consideração o tamanho dos agrupamentos, como mostrado na Eq. (6.3).

$$wCPr = \frac{\sum_{C \in \mathcal{L}_G, |C|>1} |C| \times CPr_C}{\sum_{C' \in \mathcal{L}_G, C'>1} |C'|} \tag{6.3}$$

Exemplo 6.1.3 *Considerando a Figura 6.1(b), os agrupamentos resultantes produzem* $CPr = PCpr = (\frac{12+6+6+6+6+2}{18 \times 17} + 1 + 1 + 1 + 1 + 1) \times \frac{1}{6} = \frac{0,12+5}{6} = 0,87$. *Por sua vez, os agrupamentos resultantes produzem* $wCPr = \frac{18}{31} \times \frac{12+6+6+6+6+2}{18 \times 17} + \frac{3}{31} \times 1 + \frac{4}{31} \times 1 + \frac{2}{31} \times 1 + \frac{2}{31} \times 1 + \frac{2}{31} \times 1 = 0,55$.

Como mencionado anteriormente, é importante que um método de DID produza resultados de eficácia satisfatórios ao longo do tempo. Por esta razão, é formulada a definição de eficácia pré-definida, ou seja, a habilidade de um método de DID gerar resultados de precisão (Pr) e cobertura (Rc) pré-definidos (para simplificação, as métricas CPr e CRc são denotadas por Pr e Rc, respectivamente).

Definição 6.1.1 (Eficácia Pré-definida) *Sejam* $\mathcal{L}_G$ *um agrupamento em* G, *e* p' *e* r' *números reais, tal que* $0 < p' \leq 1$ *e* $0 < r' \leq 1$. *Um método de DID* f *satisfaz a propriedade eficácia pré-definida se, para qualquer deduplicação incremental, baseada em* $\mathcal{L}_G$ *e* ΔG, *o resultado f(*G, ΔG, $\mathcal{L}_G$*) produzir* $Pr(\mathcal{L}_{G+\Delta G}) > p'$ *e* $Rc(\mathcal{L}_{G+\Delta G}) > r'$.

Existem dois fatores relacionados à definição de Eficácia Pré-definida que são importantes de serem destacados: i) a definição é baseada em um agrupamento de entrada $\mathcal{L}_G$, ao invés de $\mathcal{L}_G^{opt}$ (ou seja, não pressupõe um método ótimo de DID); e ii) na prática, a efetividade de

$f(G, \Delta G, \mathcal{L}_G)$ pode ser fortemente influenciada pela qualidade dos agrupamentos em $\mathcal{L}_G$. Desse modo, é importante adotar uma estratégia para medir a eficácia de $f(G, \Delta G, \mathcal{L}_G)$, dada a eficácia do agrupamento de entrada $\mathcal{L}_G$. Para tal, é proposta a noção de estabilidade de eficácia relacionada a um método de DID.

Definição 6.1.2 (Estabilidade de Eficácia) *Seja $\mathcal{L}_G$ um agrupamento em G. Um método de DID f satisfaz estabilidade de eficácia se para qualquer ΔG, o resultado f(G, ΔG, $\mathcal{L}_G$) produz $Pr(\mathcal{L}_{G+\Delta G}) \geq Pr(\mathcal{L}_G)$ e $Rc(\mathcal{L}_{G+\Delta G}) \geq Rc(\mathcal{L}_G)$.*

Claramente, a estabilidade de eficácia relacionada a um método f de DID é uma propriedade desejável. No entanto, é bastante difícil provar tal propriedade para um método de DID, uma vez que os resultados $Rc(\mathcal{L}_{G+\Delta G})$ e $Pr(\mathcal{L}_{G+\Delta G})$ podem depender de fatores tais como as características de G, o valor de θ e a efetividade da função (ou funções) de similaridade empregadas, características estas que são independentes das propriedades de um método de DID. Portanto, é importante definir uma estratégia para calcular o *grau de estabilidade de eficácia* de um método de DID. Para tal, são inicialmente definidas duas métricas: perda acumulada de precisão ($accPrLoss$) e perda acumulada de cobertura ($accRcLoss$), as quais são utilizadas para calcular o grau de estabilidade de eficácia de um método de DID.

A perda acumulada de precisão é definida na Eq. (6.4). Na prática, esta métrica visa calcular, para um grafo de similaridade G_0, um agrupamento inicial $\mathcal{L}_{G_0}$ e um conjunto $\{\Delta G_0, \ldots, \Delta G_{n-1}\}$ de incrementos, a perda de precisão acumulada que é gerada pela aplicação do método de DID f para processar $\{\Delta G_0, \ldots, \Delta G_{n-1}\}$. Por sua vez, a perda acumulada de cobertura é definida na Eq. (6.5). Esta métrica visa calcular a perda acumulada de cobertura que é produzida ao utilizar o método f para processar $\{\Delta G_0, \ldots, \Delta G_{n-1}\}$.

$$accPrLoss(G_0, f, \{\Delta G_0, \Delta G_1, \ldots, \Delta G_{n-1}\}) = Min\Big(1, \sum_{k=0}^{n-1} Pr(\mathcal{L}_{G_k}) - Pr(f(G_k, \Delta_{G_k}, \mathcal{L}_{G_k}))\Big) \tag{6.4}$$

$$accRcLoss(G_0, f, \{\Delta G_0, \Delta G_1, \ldots, \Delta G_{n-1}\}) = Min\Big(1, \sum_{k=0}^{n-1} Rc(\mathcal{L}_{G_k}) - Rc(f(G_k, \Delta_{G_k}, \mathcal{L}_{G_k}))\Big) \tag{6.5}$$

Sejam G_0 um grafo de similaridade, $\mathcal{L}_G$ um conjunto de agrupamentos e $\{\Delta G_0, \Delta G_1, \ldots, \Delta G_n\}$ um conjunto de incrementos sobre G_0. O grau de estabilidade de eficácia de um método de DID é definido na Eq. (6.6). Esta métrica calcula o complemento da média harmônica entre $accPrloss$ e $accRcloss$. Desse modo, quanto maiores forem os valores das perdas acumuladas (em relação à precisão e cobertura) geradas pelo método f de DID, menor será o grau de estabilidade de eficácia deste método.

$$dgEfficacySt(G_0, f, \{\Delta G_0, \ldots, \Delta G_{n-1}\}, \mathcal{L}_{G_0}) = \left(1 - \frac{2 \times (accRcLoss \times accPrLoss)}{accRcLoss + accPrLoss}\right) \tag{6.6}$$

Métricas de Eficiência

O objetivo de eficiência de um método de DID, na forma como é apresentado no estado da arte [30] (ver Seção 2.3), não define uma maneira de medir quão significativo deve ser o valor da expressão $T_{exec}(F(G, \Delta G, \mathcal{L}_G)) - T_{exec}(f(G, \Delta G, \mathcal{L}_G))$, dada a diferença entre $|G|$ e a quantidade e complexidade dos agrupamentos que são afetados por ΔG. Além disso, note que esta proporção é difícil de ser quantificada porque não é viável avaliar o valor de $F(G, \Delta G, \mathcal{L}_G)$, para cada incremento ΔG, no intuito de calcular o grau de estabilidade de eficiência de um método de DID. Portanto, é importante investigar outra estratégia, a qual não dependa da avaliação de $T_{exec}(F(G, \Delta G, \mathcal{L}_G))$, no intuito de medir o grau de estabilidade de eficiência de um método de DID.

No intuito de desenvolver métricas de eficiência específicas para DID, é necessário adotar uma estratégia para medir $|\dot{T}(\Delta G)|$, ou seja, quão complexo é empregar um método de DID sobre o componente de cobertura $\dot{T}(\Delta G)$ no intuito de atualizar seus agrupamentos ($\mathcal{L}_{\dot{T}(\Delta G)}$) após aplicar ΔG sobre G. Claramente, a complexidade associada ao processamento de $\dot{T}(\Delta G)$ depende da quantidade de vértices e arestas em $\dot{T}(\Delta G)$. Desse modo, o seguinte raciocínio é utilizado para quantificar a complexidade de $\dot{T}(\Delta G)$: quanto maior a quantidade de arestas em $\dot{T}(\Delta G)$, mais complexo se torna a tarefa de atualizar os agrupamentos em $\dot{T}(\Delta G)$. Portanto, esta complexidade é medida como $|E(\mathcal{L}_{\dot{T}(\Delta G)})|$, tal que

$E(\mathcal{L}_{\dot{T}(\Delta G)})$ é o conjunto de arestas em $\dot{T}(\Delta G)$.

Outra propriedade desejável de um método de DID, denominada *Estabilidade de Eficácia*, é utilizada para guiar a proposição de uma estratégia adequada para medir o *grau de estabilidade de eficiência* no contexto de DID.

Definição 6.1.3 (Estabilidade de Eficiência) *Sejam G_a e G_b grafos de similaridade, e ΔG_a e ΔG_b incrementos sobre G_a e G_b, respectivamente. Um método de DID f satisfaz estabilidade de eficiência se, para quaisquer G_a, G_b, ΔG_a e ΔG_b, tal que $|\dot{T}(\Delta G_a)| < |\dot{T}(\Delta G_b)|$, a aplicação de f produz: $T_{exec}(f(G_a, \Delta G_a, \mathcal{L}_{G_a}) < T_{exec}(f(G_b, \Delta G_b, \mathcal{L}_{G_b})$.*

Com base nesta definição, é possível formalizar a métrica eficiência relativa ($relEfficiency$), a qual visa medir a eficiência de um processo de DID realizado por um método f em relação a outro processo de DID também executado por f, como mostrado na Eq. (6.7). A intuição desta métrica se baseia no fato de que, quanto maior a diferença entre $|\dot{T}(\Delta G_b)|$ e $|\dot{T}(\Delta G_a)|$, menor deve ser a proporção $\frac{T_{exec}(f(G_a,\Delta G_a,\mathcal{L}_{G_a}))}{T_{exec}(f(G_b,\Delta G_b,\mathcal{L}_{G_b}))}$ (caso contrário, f está claramente não atingindo o objetivo de eficiência de um método de DID). Desse modo, se o valor de $|\dot{T}(G_b)|$ - $|\dot{T}(G_a)|$ é alto, então é atribuído um alto valor (calculado como 1 - $\frac{|\dot{T}(\Delta G_a)|}{|\dot{T}(\Delta G_b)|}$) para multiplicar $\frac{T_{exec}(f(G_a,\Delta G_a,\mathcal{L}_{G_a}))}{T_{exec}(f(G_b,\Delta G_b,\mathcal{L}_{G_b}))}$ e, portanto, para diminuir ainda mais a eficiência computada. Por outro lado, se o valor da expressão $|\dot{T}(G_b)|$ - $|\dot{T}(G_a)|$ é baixo, então é atribuído um valor baixo (1 - $\frac{|\dot{T}(\Delta G_a)|}{|\dot{T}(\Delta G_b)|}$) para multiplicar a proporção de tempos de execução (porque este cenário é menos relacionado ao objetivo de eficiência de um método de DID, o qual é especialmente relevante quando $|\dot{T}(G_a)| \ll |\dot{T}(G_b)|$) e, por esta razão, a eficiência calculada é menos afetada).

$$relEfficiency(f(G_a, \Delta G_a, \mathcal{L}_{G_a}), f(G_b, \Delta G_b, \mathcal{L}_{G_b})) = \begin{cases} \left(1 - \left(1 - \frac{|\dot{T}(\Delta G_a)|}{|\dot{T}(\Delta G_b)|}\right) \times \left(\frac{T_{exec}(f(G_a,\Delta G_a,\mathcal{L}_{G_a}))}{T_{exec}(f(G_b,\Delta G_b,\mathcal{L}_{G_b}))}\right)\right) & \textbf{se } \frac{|\dot{T}(\Delta G_a)|}{|\dot{T}(\Delta G_b)|} < 1 \\ 0 & \text{caso contrário} \end{cases} \tag{6.7}$$

Assim, a métrica que visa calcular o grau de estabilidade de eficiência, mostrada na Eq. (6.8), é calculada como a média dos valores $relEfficiency$ para todos os pares ($\Delta G_a, \Delta G_b$) $\in (\Delta G \times \Delta G)$, tal que $\frac{|\dot{T}(\Delta G_a)|}{|\dot{T}(\Delta G_b)|} < 1$. Uma vez que o valor final calculado pela Eq. (6.8) é

Tabela 6.1: objetivos sumarizados de um método de DID.

Dados	$\{G_0, \mathcal{L}_{G_0}, \{\Delta G_0, \Delta G_1, \ldots, \Delta G_{n-1}\}\}$
Minimizar	$accPrLoss$, $accRcLoss$, $accExecTime$
Maximizar	$dgEfficacySt$, $dgEficiencySt$, CPr, CRc, $PCPr$, $wCPr$
Propriedades Desejáveis	*Estabilidade de Eficácia*, *Estabilidade de Eficiência*

baseado na métrica $relEfficiency$, então a habilidade de um método de DID de maximizar sua eficiência relativa é uma propriedade desejável.

$$dgEficiencySt(f(G_0, \Delta G = \{\Delta G_0, \ldots, \Delta G_{n-1}\}, \mathcal{L}_{G_0})) = \sum_{\Delta G_a, \Delta G_b \in \Delta G} \frac{relEfficiency(f(G_a, \Delta G_a, \mathcal{L}_{G_a}), f(G_b, \Delta G_b, \mathcal{L}_{G_b}))}{|(\Delta G_a, \Delta G_b)|, s.t. \frac{|\dot{T}(\Delta G_a)|}{|\dot{T}(\Delta G_b)|} < 1} \tag{6.8}$$

Por sua vez, outra métrica básica para calcular a eficiência de um método de DID é o tempo de execução acumulado ao processar $\{\Delta G_0, \Delta G_1, \ldots, \Delta G_{n-1}\}$, como apresentado na Eq. (6.9).

$$accExecTime(G_0, f, \{\Delta G_0, \Delta G_1, \ldots, \Delta G_{n-1}\}) = \sum_{k=0}^{n-1} T_{exec}(f(G_k, \Delta_{G_k}, \mathcal{L}_{G_k})) \tag{6.9}$$

Na Tabela 6.1, são sumarizados os objetivos e propriedades desejáveis de um método de DID.

6.1.1 Heurísticas para Deduplicação Incremental de Dados

Nesta seção, são propostas heurísticas no intuito de otimizar ainda mais o processo de DID, em relação aos resultados da abordagem do estado da arte (algoritmo $Greedy$, o qual realiza operações de $Merge$, $Split$ e $Move$ nos agrupamentos pertencentes ao componente de cobertura), e manter níveis aceitáveis de resultados de eficácia. Para tal, são exploradas e combinadas as seguintes estratégias:

i) a utilização de algoritmos de agrupamento automático eficientes, disponíveis no estado da arte (apresentados no Capítulo 2);

ii) a aplicação de filtros de componente de cobertura, os quais visam limitar a quantidade de agrupamentos em $\mathcal{L}_{\dot{T}(\Delta G)}$ que são processados pelo método de DID; e

iii) a combinação das estratégias i) e ii) com uma abordagem gulosa ($Greedy$ [30]).

Ainda que a utilização das estratégias i) e ii) tenda a otimizar o processo de DID, a utilização destas estratégias pode também reduzir a eficácia dos agrupamentos produzidos. Por esta razão, é importante investigar estratégias que apresentem bons resultados considerando a relação entre eficiência do método de DID e a qualidade dos agrupamentos produzidos.

Na prática, existe uma série de cenários em que pode ser vantajosa a otimização do processo de DID em troca de resultados de eficácia ligeiramente inferiores. Primeiro, pode existir uma restrição de tempo (T_{res}) bastante rígida, especificada por meio de um DQSLA, associada à execução do método de DID. Desse modo, não é viável processar todos os agrupamentos contidos em $\dot{T}(\Delta G)$ para que a restrição de tempo seja respeitada. Em outras palavras, este cenário requer o melhor resultado de eficácia possível, dada uma restrição de tempo igual a T_{res}.

Segundo, se os incrementos nos dados afetam uma grande quantidade de agrupamentos e/ou são bastante frequentes, então torna-se bastante custoso processar todos os agrupamentos afetados pelos incrementos no componente de cobertura. Por fim, pode ser necessário aplicar múltiplas chaves de bloco sobre a base de dados ou realizar múltiplas passagens utilizando a técnica de indexação janela deslizante (visando aumentar a eficácia do processo de deduplicação) e, por este motivo, é também necessário executar um processo de DID distinto para cada chave de bloco aplicada. Portanto, esforços no sentido de otimizar o processo de DID em cada um destes cenários são importantes.

Relação entre Minimização de Penalidade de Agrupamentos, Precisão e Cobertura

Na prática, utilizar um método de DID baseado em minimização de função de penalidade nem sempre produz melhores resultados de F-measure. Para ilustrar este cenário, são apresentando dois exemplos a seguir.

Exemplo 6.1.4 *Considerando a Figura 6.2(a), note que o agrupamento resultante produz* $CC = 0,1+0,2+0,1+0,8+0,8 = 2$ *(penalidade mínima), mas este resultado também gera perda de cobertura. Por sua vez, considerando a Figura 6.2(b), os agrupamentos resultantes geram* $CC = 0,1+0,1+0,2+0,2+0,2+1+1 = 2.8$*, mas este resultado produz cobertura perfeita.*

Exemplo 6.1.5 *Considerando a Figura 6.3(a), note que o agrupamento resultante produz* $CC = 0,2 + 0,2 + 0,2 + 0,85 = 1,45$ *(penalidade mínima), mas este resultado também gera perda de precisão. Por outro lado, considerando a Figura 6.3(b), os resultados dos agrupamentos produzem* $CC = 0,2 + 0,8 + 0,8 + 0,15 = 1,95$*, mas geram precisão perfeita.*

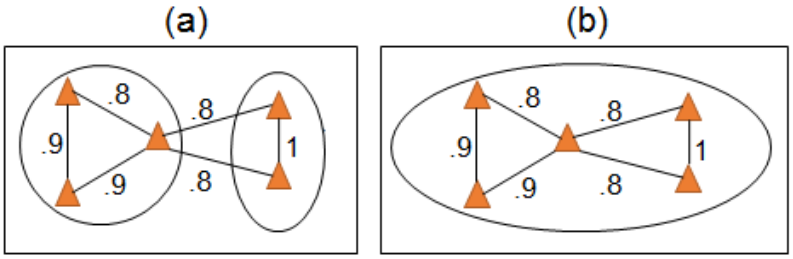

Figura 6.2: (a) Exemplo de grafo de similaridade em que a minimização da função de penalidade gera perda de cobertura. (b) Exemplo de grafo de similaridade em que um resultado não ótimo de função de penalidade produz cobertura perfeita.

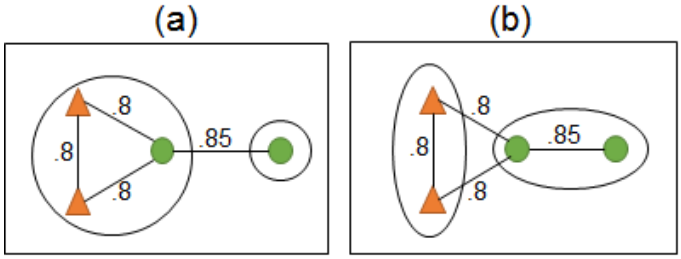

Figura 6.3: Exemplo de grafo de similaridade em que a minimização da função de penalidade gera perda de precisão. (b) Exemplo de grafo de similaridade em que um resultado não ótimo de função de penalidade produz precisão perfeita.

Considerando os Exemplos 6.2 e 6.3, é possível argumentar que a escolha de um valor adequado para o parâmetro θ (limiar utilizado para definir o valor mínimo do peso das arestas

em um grafo de similaridade) e a utilização de uma função de similaridade adequada podem eliminar (ou ao menos minimizar) as situações em que minimização de função de penalidade produzem perda de eficácia. No entanto, é difícil ajustar estes parâmetros antes da execução de um método de DID. Desse modo, é importante investigar empiricamente se as situações descritas nos Exemplos 6.2 e 6.3 ocorrem na prática, no contexto de DID.

Algoritmo Q-Greedy

Neste trabalho, é utilizado um algoritmo ligeiramente diferente do algoritmo $Greedy$ (Algoritmo 1), denominado **Q**-Greedy, o qual recebe como parâmetro de entrada uma pilha de agrupamentos a ser processada, ao invés de adicionar cada agrupamento em $\bar{T}(\Delta G)$ na pilha $\mathbf{Q}^{\mathbf{c}}$. O pseudocódigo do algoritmo **Q**-Greedy é representado no Algoritmo 5.

Filtro de Componente de Cobertura

Note que, mesmo aplicando o método mais restritivo ($\bar{T}$) para selecionar um componente de cobertura, na prática $\mathcal{L}_{\bar{T}(\Delta G)}$ pode conter uma grande quantidade de agrupamentos. Por exemplo, considerando o exemplo da Figura 2.5, $\mathcal{L}_{\bar{T}(\Delta G)} = \{C_1, C_2, C_3, C_5, C_6, C_7, C_8, C_9\}$, ou seja, apenas três incrementos nos dados afetaram oito agrupamentos.

Por esta razão, neste trabalho são propostas heurísticas para reduzir ainda mais a quantidade de agrupamentos a serem processados por um método de DID. A ideia principal destas heurísticas é limitar a aplicação do algoritmo **Q**-$Greedy$ para um subconjunto dos agrupamentos em $\mathcal{L}_{\bar{T}(\Delta G)}$. Este processo é realizado por um *Filtro de Componente de Cobertura*.

Definição 6.1.4 (Filtro de Componente de Cobertura) *Sejam G um grafo de similaridade, ΔG um conjunto de incrementos sobre G, $\dot{T}$ um método para selecionar um componente de cobertura e $\mathcal{L}_{\dot{T}(\Delta G)}$ um agrupamento em $\dot{T}(\Delta G)$. Um filtro de componente de cobertura $\mathcal{F}$ é uma heurística que visa selecionar um subconjunto $\mathcal{L}^{\mathcal{F}}_{\dot{T}(\Delta G)} \in 2^{\mathcal{L}_{\dot{T}(\Delta G)}}$, tal que $F\text{-}measure(\mathbf{Q}\text{-}Greedy(\mathcal{L}_{\lambda(\Delta G)}))$ e $(|\mathcal{L}_{\lambda(\Delta G)}| - |\mathcal{L}^{\mathcal{F}}_{\lambda(\Delta G)}|)$ são maximizados.*

Os objetivos de um filtro ($\mathcal{F}$) de componente de cobertura são formalizados na Tabela 6.2. Primeiramente, é importante notar a relação entre $|\mathcal{L}^{\mathcal{F}}_{G}|$, $T_{exec}(\mathbf{Q}\text{-}Greedy(\mathcal{L}^{\mathcal{F}}_{G}))$ e a eficácia produzida por $\mathbf{Q}\text{-}Greedy(\mathcal{L}^{\mathcal{F}}_{G})$. Desse modo, é esperado que um filtro $\mathcal{F}$ possa gerar um resultado balanceado em relação a estes três aspectos. Segundo, é essencial que sejam

empregadas estratégias simples (ou seja, eficientes) para selecionar $\mathcal{L}_G^{\mathcal{F}}$, caso contrário $\mathcal{F}$ pode não respeitar a restrição de eficiência associada aos filtros de componente de cobertura (mostrada na última linha da Tabela 6.2). Na Seção C.4 (Apêndice C) são propostos algoritmos para tratar o problema formalizado na Tabela 6.2.

6.2 Considerações Finais

Neste capítulo, foram propostos definições, métricas e algoritmos para o contexto de deduplicação incremental de dados utilizando classificação coletiva. As definições propostas serviram como base para a elaboração de métricas específicas para a avaliação de algoritmos de DID. Ao total, foram propostas sete métricas específicas para o contexto de DID, sendo duas métricas de eficiência ($dgEfficacySt$ e $accExecTime$) e cinco métricas de eficácia ($accPrLoss$, $accRcLoss$, $dgEficiencySt$, CRc e $wCPr$). Por fim, foram também propostas novas heurísticas para o contexto de DID, assim como uma metaheurística específica (denominada CFG), a qual pode ser utilizada para a geração de diferentes algoritmos de DID. As métricas propostas neste capítulo, assim como as heurísticas e a metaheurística desenvolvida, são exploradas no Capítulo 8 visando avaliar diferentes algoritmos de DID considerando tanto métricas clássicas (cobertura e precisão) quanto as métricas propostas neste capítulo.

Algoritmo 5: Q-$Greedy$

Input : $G(V, E)$: um grafo de similaridade

$\mathcal{L}_G$: um agrupamento em G

$\mathbf{Q^c}$: uma pilha de agrupamentos

Output: um novo agrupamento em $\mathcal{L}_{G+\Delta G}$

```
1 begin
2     while Q^c ≠ ∅ do
3     // procede como o algoritmo Greedy (Algoritmo 1) a
         partir da linha 6
```

Tabela 6.2: Objetivos formalizados de um filtro de componente de cobertura $\mathcal{F}$ para $\dot{T}(\Delta G)$.

Dados	$\{\Delta G, \dot{T}, \mathcal{L}_{\dot{T}(\Delta G)}\}$
Encontrar	$\mathcal{L}^{\mathcal{F}}_{\dot{T}(\Delta G)}$ utilizando $\mathcal{F}(\mathcal{L}_{\dot{T}(\Delta G)})$
Entre	$2^{\mathcal{L}_{\dot{T}(\Delta G)}}$
para Maximizar	$\lvert\mathcal{L}_{\dot{T}(\Delta G)}\rvert - \lvert\mathcal{L}^{\mathcal{F}}_{\dot{T}(\Delta G)}\rvert$
para Minimizar	$F\text{-}measure(\mathbf{Q}\text{-}Greedy(\mathcal{L}_{\dot{T}(\Delta G)}))-$ $F\text{-}measure(\mathbf{Q}\text{-}Greedy(\mathcal{L}^{\mathcal{F}}_{\dot{T}(\Delta G)}))$
Sujeito a	$(T_{exec}(\mathcal{F}(\mathcal{L}_{\dot{T}(\Delta G)})) + T_{exec}(\mathbf{Q}\text{-}$ $Greedy(\mathcal{F}(\mathcal{L}_{\dot{T}(\Delta G)})))) < T_{exec}(\mathbf{Q}\text{-}Greedy(\mathcal{L}_{\dot{T}(\Delta G)}))$

Capítulo 7

Controle do Tamanho de Blocos para Deduplicação de Dados

Como apresentado no Capítulo 2, a técnica de indexação denominada blocagem tem como objetivo reduzir a quantidade de comparações a serem realizadas entre entidades. Esta redução é realizada agrupando entidades que possuam alguma característica (definida por uma função de chave de bloco) em comum e limitando a comparação apenas entre entidades que pertencem ao mesmo bloco. A eficácia de uma etapa de blocagem é definida como a capacidade desta etapa de: i) reduzir a quantidade de comparações entre entidades; e ii) maximizar a quantidade de pares de entidades duplicadas que são inseridos no mesmo bloco (objetivo utilizado para determinar a qualidade dos blocos gerados).

Neste capítulo, são propostas e descritas estratégias, definições e métricas para enfatizar o problema de processamento de uma coleção de blocagens no intuito de controlar o tamanho dos blocos gerados por múltiplas funções de chave de bloco (ver Capítulo 2). Este problema precisa ser tratado na arquitetura proposta (Figura 4.1) uma vez que, ao controlar o tamanho dos blocos produzidos na etapa de indexação e, assim, reduzir a quantidade de comparações entre entidades. Como consequência, os custos de executar as etapas seguintes do processo de deduplicação de dados (Figura 2.1) são reduzidos. Desse modo, na prática, o controle no tamanho dos blocos reduz os custos associados à resolução do problema de deduplicação de dados e, portanto, os custos de infraestrutura produzidos por um SMQD. Por sua vez, esta redução acarreta em um menor custo para os clientes do serviço.

No contexto da arquitetura do SMQD proposta (Figura 4.1), os detalhes associados à

resolução do problema de controle de tamanho de blocos para deduplicação de dados, tais como o intervalo de tamanho dos blocos e as funções de chave de bloco adotadas, são definidos por meio do DQSLA que é enviado pelo cliente do serviço para definir os detalhes associados aos algoritmos utilizados para avaliar a qualidade da base de dados monitorada.

Na Seção 7.1, são apresentados o contexto e a relevância do problema investigado. Na Seção 7.2, é apresentado um exemplo no contexto do problema. Em seguida (Seção 7.3), são apresentadas métricas e definições relacionadas ao problema de controle de tamanho de blocos para deduplicação de dados. Prosseguindo, é proposta uma definição formal para o problema. Em seguida (Seção 7.4), é proposta uma metaheurística, denominada $MkPCBS$, a qual especifica os passos em alto nível utilizados para solucionar o problema investigado. Por fim, são propostas diversas heurísticas visando tratar as chamadas especificadas na metaheurística $MkPCBS$. A avaliação dos algoritmos propostos neste capítulo é apresentada no Capítulo 8.

7.1 Motivação

Na prática, a blocagem realizada na etapa de indexação pode também acarretar em desvantagens na solução de um problema de deduplicação de dados. Este fenômeno ocorre porque a etapa de blocagem causar um efeito colateral indesejado que é reduzir a quantidade de duplicatas identificadas, o que ocorre sempre que duas entidades duplicadas são inseridas em blocos distintos e, portanto, não são comparadas na etapa de comparação (ver Figura 2.1). Claramente, a eficácia da etapa de blocagem depende fortemente da eficácia da função de chave de bloco adotada. Na prática, é difícil escolher uma única função de chave de bloco que seja capaz de reduzir significativamente a quantidade de comparações entre entidades e ao mesmo tempo manter uma qualidade aceitável dos blocos produzidos. Esta escolha é especialmente desafiadora no contexto em que as entidades armazenadas no banco de dados apresentam problemas de qualidade de dados como dados inacurados ou faltantes [6].

No intuito de maximizar a qualidade dos blocos produzidos na etapa de indexação, é possível aplicar múltiplas [12; 54] chaves de bloco para indexar as entidades da base de dados. Desse modo, ao invés de aplicar uma única função de chave de bloco, é utilizado um conjunto de funções de chave de bloco, as quais são simultaneamente aplicadas

para indexar as entidades na base de dados. Assim, é produzido um resultado de blocagem (um conjunto de blocos) para cada função de chave de bloco empregada. A tendência é que, quanto mais funções de chave de bloco são empregadas (e, portanto, mais propriedades das entidades são exploradas na etapa de indexação), maiores são as chances de um par de entidades duplicadas coocorrer em pelo menos um bloco, o que contribui positivamente para a qualidade do resultado de blocagem. Esta intuição é confirmada por resultados experimentais de vários trabalhos do estado da arte [35; 39; 42; 54]. Por outro lado, quanto mais funções de chave de bloco são aplicadas, maior é quantidade de blocos gerados. Por sua vez, uma quantidade maior de blocos reduz a eficácia da técnica de blocagem que visa reduzir a quantidade de comparações a serem realizadas entre entidades.

Uma possível abordagem para reduzir a quantidade de comparações entre entidades gerada pela aplicação de múltiplas funções de chave de bloco para indexar as entidades de uma base de dados consiste em controlar o tamanho dos blocos produzidos. Ao empregar uma abordagem para garantir que o tamanho de todos os blocos gerados esteja entre um intervalo pré-definido, é possível configurar a quantidade (máxima) de comparações a serem realizadas entre entidades. Na prática, além de reduzir os custos na solução de um problema de deduplicação de dados, este controle do tamanho dos blocos facilita a aplicação de outras abordagens associadas ao problema de deduplicação de dados, tais como: i) a aplicação de técnicas de deduplicação de dados que preservam a privacidade [81; 84] e que requerem tamanhos mínimo e máximo dos blocos produzidos; ii) a execução de abordagens para deduplicação de dados em tempo real, as quais requerem que uma quantidade (máxima) pré-definida de comparações sejam realizadas entre entidades visando atender uma restrição de tempo associada à solução do problema; iii) a distribuição das comparações geradas por uma tarefa de deduplicação de dados é facilitada [53; 28] caso os tamanhos do blocos possam ser controlados; e iv) por fim, o controle no tamanho dos blocos produzidos torna viável a aplicação de técnicas de classificação mais sofisticadas, tais como classificação coletiva (Seção 2.3), as quais usualmente produzem resultados de alta qualidade, mas são custosas de serem aplicadas quando a base de dados processada é volumosa.

Em [25], os autores propõem uma abordagem para controlar o tamanho dos blocos pro-

duzidos a qual emprega funções de chave de bloco iterativamente no intuito de dividir blocos grandes, i.e., blocos com tamanho maior do que um limiar máximo. Por esta razão, esta abordagem não é viável para lidar com o controle de tamanho de blocos quando todas as funções de chave de bloco escolhidas são inicialmente empregadas para indexar as entidades na base de dados. Por sua vez, os autores de [62] propõem uma abordagem para podar um conjunto de blocagens, visando maximizar a eficácia do conjunto de blocos podado e ao mesmo tempo manter a qualidade dos blocos modificados, mas não lidam com o problema de controlar o tamanho dos blocos produzidos. Visando lidar com estas limitações dos trabalhos disponíveis no estado da arte, neste trabalho é proposta uma abordagem para controlar o tamanho dos blocos gerados por um conjunto de funções de chave de bloco que são simultaneamente aplicadas no intuito de indexar as entidades da base de dados.

7.2 Exemplo

Nesta seção é apresentado um exemplo visando ilustrar os seguintes conceitos: i) indexação baseada em uma única função de chave de bloco; e ii) indexação baseada em múltiplas funções de chave de bloco. Na Tabela 7.1, é apresentada a base de dados D_1, a qual contém 16 entidades que representam publicações científicas no mundo real. O atributo eID denota o id da entidade em D_1 e valores de eId que possuem a mesma cor representam entidades duplicadas.

Para ilustrar a situação em que a etapa de blocagem pode reduzir a qualidade do resultado da deduplicação de dados ao inviabilizar a comparação entre entidades duplicadas, são apresentados[1] exemplos de quatro funções de chave de bloco distintas que são empregadas separadamente para indexar as entidades na base de dados D_1.

Exemplo 7.2.1 (Indexação Baseada em uma Única Função de Chave de Bloco (Título)) *Ao indexar as entidades na base de dados D_1 utilizando uma única função de chave de bloco $f_1 = F_1(Titulo)$ (primeira letra do atributo Título), é produzido o seguinte resultado de blocagem:* $\{A\{d_1, d_5, d_7, d_8, d_{15}, d_{16}\},\ C\{d_2, d_3, d_4\},\ S\{d_6\},\ B\{d_9\},\ L\{d_{10}, d_{14}\},$

[1]Para representar a indexação utilizando uma única função de chave de bloco, é empregada a seguinte notação: $\{bk_1\{e_1, e_2, \cdots\}, bk_2\{e_3, e_4, \cdots\}, bk_3\{e_5, e_6, \cdots\}, \cdots\}$, tal que cada bk_i é um valor de chave de bloco único produzido pela função de chave de bloco empregada

Tabela 7.1: Base de dados D_1.

eID	Título	Ano	Conferência	Local	Autor1
d1	A dataquality-aware cloud service based on machine learning	2015	SAC	Salamanca	Nascimento, D.C. et. al
d2	Cloud service using Machine Learning	2015	Applied Computing	Spain	FILHO, D.C.
d3	Cloud service based on metaheuristic	2016	Applied Computing	Spain	Nascimento Filho, D
d4	Cloud service based on provisioning algorithms		SAC	Salamanca	Filho, D
d5	Adaptive sorted neighborhood blocking		SAC	Salamanca	GOMES, Demetrio
d6	Sorted neighborhood blocking for entity matching with MapReduce	2015	SAC	Salamanca	G M, Demetrio
d7	Approach to detect subsumption relations between tags	2015	Appl. Comp.	Salamanca	REGO, Alex
d8	A Supervised learning approach to detect subsumption relations	2015	Appl. Comp.	Spain	R., Alex
d9	Building lexical-semantic resources	2016	SAC	Spain	G., Jose
d10	Lexical-semantic resources based on heterogeneous information	2015	SAC	Spain	GILDO, Jose
d11	Framework for screening	2016	SAC	Italy	Vinicius Rosa
d12	Framework for virtual screening	2016	SAC	Pisa	ROSA, Vinicius
d13	Nodule classification based on shape distributions	2016	Appl. Comp.	Pisa	Fernandes, Valeria
d14	Lung nodule classification based on shape distributions		SAC	Italy	Fernandes, V.
d15	A Classification based on shape distributions	2016	SAC	Italy	Valéria, P. M.
d16	A taxonomy of reliable request-response protocols	2015	Appl. Comp.	Salamanca	Naghmeh Ivaki

$F\{d_{11}, d_{12}\}$, $N\{d_{13}\}\}$. *Note que este resultado de blocagem reduz a qualidade do resultado de deduplicação de dados ao impedir a comparação entre os seguintes pares de entidades duplicadas:* $\{(d_1, d_2)$, (d_1, d_3), (d_1, d_4), (d_5, d_6), (d_9, d_{10}), (d_{13}, d_{14}), (d_{13}, d_{15}), $(d_{14}, d_{15})\}$. *Portanto, 8 entidades duplicadas não poderão ser identificadas.*

Exemplo 7.2.2 (Indexação Baseada em uma Única Função de Chave de Bloco (Ano)) *Ao indexar as entidades na base de dados* D_1 *utilizando uma única função de chave de bloco* $f_2 = Ano$ *(o valor do atributo Ano), é produzido o seguinte resultado de blocagem:* $\{2015\{d_1, d_2, d_6, d_7, d_8, d_{10}, d_{16}\}, 2016\{d_3, d_9, d_{11}, d_{12}, d_{13}, d_{15}\}\}$. *Note que este resultado de blocagem reduz a qualidade do resultado de deduplicação de dados ao impedir a comparação entre os seguintes pares de entidades duplicadas:* $\{(d_1, d_3)$, (d_1, d_4), (d_2, d_3), (d_2, d_4), (d_3, d_4), (d_5, d_6), (d_9, d_{10}), (d_{13}, d_{14}), $(d_{14}, d_{15})\}$. *Desse modo, 9 entidades duplicadas não poderão ser identificadas.*

Exemplo 7.2.3 (Indexação Baseada em uma Única Função de Chave de Bloco (Autor1)) *Ao indexar as entidades na base de dados* D_1 *utilizando uma única função de chave de bloco* $f_3 = F_1(Autor1)$ *(primeira letra do atributo Autor1), é produzido o seguinte resultado de blocagem:* $\{N\{d_1, d_3, d_{16}\}$, $F\{d_2, d_4, d_{14}, d_{13}\}$, $G\{d_5, d_6, d_9, d_{10}\}$, $R\{d_7, d_8, d_{12}\}$, $V\{d_{11}, d_{15}\}\}$. *Note que este resultado de blocagem reduz a qualidade do resultado de deduplicação de dados ao impedir a comparação entre os seguintes pares de entidades duplicadas:* $\{(d_1, d_2)$, (d_1, d_4), (d_2, d_3), (d_3, d_4), (d_{11}, d_{12}), (d_{13}, d_{15}), $(d_{14}, d_{15})\}$. *Portanto, 7 entidades duplicadas não poderão ser identificadas.*

Exemplo 7.2.4 (Indexação Baseada em uma Única Função de Chave de Bloco (Local)) *Ao indexar as entidades na base de dados* D_1 *utilizando uma única função de chave de bloco* $f_4 = (Local)$ *(o valor do atributo Local), é produzido o seguinte resultado de blocagem:* $\{Salamanca\{d_1, d_4, d_5, d_6, d_7, d_{16}\}$, $Spain\{d_2, d_3, d_8, d_9, d_{10}\}$, $Italy\{d_{11}, d_{14}, d_{15}\}$, $Pisa\{d_{12}, d_{13}\}\}$. *Note que este resultado de blocagem reduz a qualidade do resultado de deduplicação de dados ao impedir a comparação entre os seguintes pares de entidades duplicadas:* $\{(d_1, d_2)$, (d_1, d_3), (d_2, d_4), (d_3, d_4), (d_7, d_8), (d_{11}, d_{12}), (d_{13}, d_{14}), $(d_{13}, d_{15})\}$. *Assim, 8 entidades duplicadas não poderão ser identificadas.*

No intuito de aumentar a eficácia da abordagem de blocagem, é possível empregar múltiplas chaves de bloco para indexar as entidades na base de dados D_1, como ilustrado no

Exemplo[2] 7.2.5.

Exemplo 7.2.5 (Indexação Baseada em Múltiplas Funções de Chave de Bloco) *Ao indexar as entidades na base de dados D_1 utilizando uma abordagem baseada em múltiplas funções de chave de bloco, a qual emprega o conjunto de funções de chave de bloco $F_{key} = \{f_1, f_3, f_4\}$[3], é produzido o seguinte conjunto de blocagens $\mathcal{B}_{D_1}$: $\{\{A\{d_1, d_5, d_7, d_8, d_{15}, d_{16}\}, C\{d_2, d_3, d_4\}\}, S\{d_6\}, B\{d_9\}, L\{d_{10}, d_{14}\}, F\{d_{11}, d_{12}\}, N\{d_{13}\}\}$, $\{N\{d_1, d_3, d_{16}\}, F\{d_2, d_4, d_{14}, d_{13}\}, G\{d_5, d_6, d_9, d_{10}\}, R\{d_7, d_8, d_{12}\}, V\{d_{11}, d_{15}\}\}$, $\{Salamanca\{d_1, d_4, d_5, d_6, d_7, d_{16}\}, Spain\{d_2, d_3, d_8, d_9, d_{10}\}, Italy\ \{d_{11}, d_{14}, d_{15}\}$, $Pisa\{d_{12}, d_{13}\}\}\}$. Por conseguinte, um único par de entidades duplicadas (d_1, d_2) não poderá ser identificado.*

Ainda que a abordagem mostrada no Exemplo 7.2.5 permita a identificação da grande maioria das entidades duplicadas em D_1, a aplicação de múltiplas funções de chave de bloco também aumenta a quantidade de comparações entre entidades (em relação à abordagem baseada em uma função de chave de bloco única). Visando reduzir esta diferença, podem ser utilizados algoritmos que visam controlar o tamanho dos blocos produzidos.

7.3 Notação Adotada

Nesta seção, é apresentada a notação adotada nas definições, métricas e soluções associadas ao problema de controle de tamanho de blocos para deduplicação de dados. Seja $D = \{e_1, e_2, \cdots, e_{|D|}\}$ um conjunto de entidades em uma base de dados. Ao empregar uma única função de chave de bloco f para indexar as entidades na base de dados D, é produzido o resultado de blocagem $B = \{b_1, b_2, \cdots, b_{|B|}\}$, tal que $\forall e \in D(\exists b \in B(e \in b))$ e $\cup_{b \in B} b = D$. O resultado de blocagem pode produzir blocos disjuntos (i.e., $\cap_{b \in B} b = \emptyset$) ou blocos sobrepostos (i.e., $\cap_{b \in B} b \neq \emptyset$), dependendo da abordagem de blocagem que é empregada. Por

[2]Para a indexação utilizando múltiplas funções de chave de bloco, é empregada a seguinte notação: $\{\{bk_{i,1}\{e_1, e_2, \cdots\}, bk_{i,2}\{e_3, e_4, \cdots\}, bk_{i,3}\{e_5, e_6, \cdots\}, \cdots\}, \{bk_{j,1}\{e_1, e_2, \cdots\}, bk_{j,2}\{e_3, e_4, \cdots\}, bk_{j,3}\{e_5, e_6, \cdots\}, \cdots\}, \cdots\}$, tal que cada $bk_{l,m}$ represente o m-*ésimo* valor de chave de bloco produzido pela l-*ésima* função de chave de bloco

[3]as funções de chave de bloco $\{f_1, f_3, f_4\}$ são empregadas separadamente nos Exemplos 7.2.1, 7.2.3 e 7.2.4, respectivamente

sua vez, se o conjunto $F_{key} = \{f_1, f_2, \cdots, f_{|F_{key}|}\}$ de funções de chave de bloco é empregado, então uma coleção de blocagens $\mathcal{B} = \{B_1, B_2, \cdots, B_{|\mathcal{B}|}\}$ é produzida, tal que $\forall B \in \mathcal{B}(\forall e \in D(\exists b \in B(e \in b)))$ e $\forall B \in \mathcal{B}(\cup_{b \in B} b = D)$.

Utilizando a notação adotada, na Definição 7.3.1 é apresentada a noção de cardinalidade agregada.

Definição 7.3.1 (Cardinalidade Agregada) *A quantidade de comparações (cardinalidade agregada) produzida por um resultado de blocagem baseado em uma única função de chave de bloco é calculado como:* $||B|| = \sum_{b \in B} \frac{|b|*(|b|-1)}{2}$. *Por sua vez, a quantidade de comparações produzida por uma coleção de blocagens* $\mathcal{B}$ *é igual a* $||\mathcal{B}|| = \sum_{B \in \mathcal{B}} \sum_{b \in B} \frac{|b|*(|b|-1)}{2}$.

Exemplo 7.3.1 (Cardinalidade Agregada) *A aplicação (individual) das funções de chave de bloco* f_1 *(Exemplo 7.2.1),* f_2 *(Exemplo 7.2.2),* f_3 *(Exemplo 7.2.3) e* f_4 *(Exemplo 7.2.4) produz* $\frac{6*5}{2} + \frac{3*2}{2} + \frac{2*1}{2} + \frac{2*1}{2} = 20$, $\frac{7*6}{2} + \frac{6*5}{2} = 36$, $\frac{3*2}{2} + \frac{4*3}{2} + \frac{4*3}{2} + \frac{3*2}{2} + \frac{2*1}{2} = 19$ *e* $\frac{6*5}{2} + \frac{5*4}{2} + \frac{3*2}{2} + \frac{2*1}{2} = 29$ *comparações entre entidades, respectivamente. Por sua vez, a aplicação da abordagem baseada em múltiplas funções de chave de bloco utilizando o conjunto* $\{f_1, f_3, f_4\}$ *produz* $\frac{6*5}{2} + \frac{3*2}{2} + \frac{2*1}{2} + \frac{2*1}{2} + \frac{3*2}{2} + \frac{4*3}{2} + \frac{4*3}{2} + \frac{3*2}{2} + \frac{2*1}{2} + \frac{6*5}{2} + \frac{5*4}{2} + \frac{3*2}{2} + \frac{2*1}{2} = 68$ *comparações entre entidades. Ainda que a abordagem baseada em múltiplas funções de chave de bloco produza uma quantidade maior de comparações entre entidades (68), note que esta quantidade é ainda muito menor do que a quantidade de comparações produzida pela abordagem baseada básica de realizar comparações entre todas as entidades (produto cartesiano):* $\frac{16*(16-1)}{2} = 120$.

7.3.1 Métricas de qualidade

Nesta seção, são apresentadas métricas utilizadas para avaliar a qualidade do resultado de blocagens definidas por trabalhos do estado da arte [62; 57; 17].

Definição 7.3.2 (Pair Quality (PQ)) *Seja* $\mathcal{B}$ *uma coleção de blocagens. A métrica Pair Quality, a qual é equivalente à métrica precisão na área de Recuperação da Informação, visa calcular a porcentagem de duplicatas corretamente identificada pelo resultado de blo-*

cagem, como mostrado na equação Eq. (7.1)[4]*:*

$$PQ(\mathcal{B}) = \sum_{B \in \mathcal{B}} \sum_{b \in B} \frac{|(e_1, e_2) \in (b \times b),\ s.t.\ e_1 \neq e_2 \wedge match(e_1, e_2)|}{\frac{|b|*(|b|-1)}{2}} \tag{7.1}$$

O valor de PQ varia entre [0,1] e quanto mais próximo o valor for de 1, mais preciso é o resultado de blocagem.

Exemplo 7.3.2 (Pair Quality) *Considerando o conjunto de blocagem $\mathcal{B}_{D_1}$ apresentado no Exemplo 7.2.5, tal conjunto produz o seguinte resultado:* $PQ = |\{(d_1, d_3),(d_1, d_4),\ (d_2, d_3),\ (d_2, d_4),\ (d_3, d_4),\ (d_5, d_6),\ (d_7, d_8),\ (d_9, d_{10}),\ (d_{11}, d_{12}),\ (d_{13}, d_{14}),\ (d_{13}, d_{15}),\ (d_{14}, d_{15})\}|/68 = 0.18$ *(ver Exemplo 7.3.1).*

Definição 7.3.3 (Pair Completeness (PC)) *Seja D uma base de dados e $\mathcal{B}$ uma coleção de blocagens com base em D. A métrica Pair Completeness é equivalente à noção de cobertura na área de Recuperação da Informação, a qual visa calcular a relação entre a quantidade de entidades duplicadas que podem ser corretamente identificadas a partir do resultado de blocagem e a quantidade total de entidades duplicadas na base de dados, como apresentado na Eq. (7.2):*

$$PC(\mathcal{B}) = \frac{\sum_{B \in \mathcal{B}} \sum_{b \in B} |(e_1, e_2) \in (b \times b),\ s.t.\ e_1 \neq e_2 \wedge match(e_1, e_2)|}{|(e_1, e_2) \in (D \times D),\ s.t.\ e_1 \neq e_2 \wedge match(e_1, e_2)|} \tag{7.2}$$

O valor de PC varia entre [0,1] e quanto mais próximo o valor for de 1, mais completo é o resultado de blocagem.

Exemplo 7.3.3 (Pair Completeness) *Considerando o conjunto de blocagem $\mathcal{B}_{D_1}$ apresentado no Exemplo 7.2.1, tal conjunto produz o seguinte resultado:* $PC = |\{(d_2, d_3),\ (d_2, d_4),\ (d_3, d_4),\ (d_7, d_8),\ (d_{11}, d_{12})\}\ /|\{(d_1, d_2),\ (d_1, d_3),(d_1, d_4),\ (d_2, d_3),\ (d_2, d_4),\ (d_3, d_4),\ (d_5, d_6),\ (d_7, d_8),\ (d_9, d_{10}),\ (d_{11}, d_{12}),\ (d_{13}, d_{14}),\ (d_{13}, d_{15}),\ (d_{14}, d_{15})\}|= 0.38$. *Por sua vez, considerando o resultado de blocagem $\mathcal{B}_{D_1}$ apresentado no Exemplo 7.2.5, tal conjunto produz o seguinte resultado:* $PC = |\{(d_1, d_3),(d_1, d_4),\ (d_2, d_3),\ (d_2, d_4),\ (d_3, d_4),\ (d_5, d_6),\ (d_7, d_8),\ (d_9, d_{10}),\ (d_{11}, d_{12}),\ (d_{13}, d_{14}),\ (d_{13}, d_{15}),\ (d_{14}, d_{15})\}|\ /|\{(d_1,d_2),\ (d_1, d_3),(d_1, d_4),\ (d_2, d_3),\ (d_2, d_4),\ (d_3, d_4),\ (d_5, d_6),\ (d_7, d_8),\ (d_9, d_{10}),\ (d_{11}, d_{12}),\ (d_{13}, d_{14}),\ (d_{13}, d_{15}),\ (d_{14}, d_{15})\}|= 0.92$.

[4]as entidades duplicadas são contadas apenas uma vez na Eq. (7.1).

Visando reduzir a quantidade de comparações produzida por um resultado de blocagem, é possível utilizar um algoritmo de poda [62] para transformar um resultado de blocagem de entrada $\mathcal{B}$ em um resultado de blocagem podado $\mathcal{B}'$ que reduz a quantidade de comparações entre as entidades e (idealmente) mantém a qualidade (ver Definições 7.3.2 e 7.3.3) da blocagem. É possível calcular a eficácia da poda de uma coleção de blocos utilizando a métrica *Reduction Ratio* (RR) [57], a qual é apresentada na Definição 7.3.4.

Definição 7.3.4 (Reduction Ratio) *A métrica Reduction Ratio visa calcular a eficácia de uma operação de poda entre uma coleção de blocagens recebida como entrada $\mathcal{B}$ e uma coleção de blocagens podada $\mathcal{B}'$. Para tal, a taxa de redução é calculada como $RR(\mathcal{B}, \mathcal{B}') = 1 - \frac{||\mathcal{B}'||}{||\mathcal{B}||}$. O valor de RR varia entre [0,1] e quanto mais próximo de 1 é o resultado, mais eficaz é o processo de poda.*

7.3.2 Poda de Coleção de Blocagens

Na Definição 7.3.5, é apresentado o objetivo de um algoritmo de poda de coleção de blocagens, o qual visa reduzir a cardinalidade agregada de uma coleção de blocagens e ao mesmo tempo manter resultados de qualidade de blocagem aceitáveis (ver Definições 7.3.2 e 7.3.3).

Definição 7.3.5 (Poda de Coleção de Blocagens) *Seja D uma base de dados e $F_{key} = \{f_1, f_2, ..., f_{|F_{key}|}\}$ uma coleção de funções de chave de bloco inicialmente aplicadas simultaneamente para indexar as entidades de D e produzir uma coleção de blocagens $\mathcal{B}$. Um algoritmo de poda de coleção de blocagens visa criar uma coleção de blocagens podada $\mathcal{B}'$, tal que $RR(\mathcal{B}', \mathcal{B})$, $PC(\mathcal{B}')$ e $PQ(\mathcal{B}')$ são maximizados.*

Exemplo 7.3.4 (Poda de Coleção de Blocagens) *Considerando a coleção de blocagens $\mathcal{B}_{D_1}$ apresentada no Exemplo 7.2.5, é possível produzir uma coleção de blocagens podada ($\mathcal{B}'_{D_1}$) empregando diferentes estratégias. Por exemplo, é possível descartar blocos da coleção de blocagens recebida como entrada $\mathcal{B}_{D_1}$. Empregando esta abordagem, se os blocos $G\{d_5, d_6, d_9, d_{10}\}$, $A\{d_1, d_5, d_7, d_8, d_{15}, d_{16}\}$, $S\{d_6, d_8\}$, $L\{d_{10}, d_{14}\}$, $Pisa\{d_{12}, d_{13}\}\}$ e $V\{d_{11}, d_{15}\}$ são excluídos de $\mathcal{B}_{D_1}$, a seguinte coleção de blocagens podada é produzida: $\mathcal{B}'_{D_1} = \{\{C\{d_2, d_3, d_4\}, F\{d_{11}, d_{12}\}, B\{d_9\}, N\{d_{13}\}\}, \{Salamanca\{d_1, d_4, d_5, d_6, d_7, d_{16}\}, Spain\{d_2, d_3, d_8, d_9, d_{10}\}, Italy\{d_{11}, d_{14}, d_{15}\}\}, \{N\{d_1, d_3, d_{16}\},$*

$F\{d_2, d_4, d_{14}, d_{13}\},\ R\{d_7, d_8, d_{12}\}\}$. *Note que* $\mathcal{B}_{D_1}$ *reduz a cardinalidade agregada (*$||\mathcal{B}_{D_1}||= 68$ *e* $||\mathcal{B}'_{D_1}||= 44$*) e ao mesmo tempo mantém* $PC(\mathcal{B}) = PC(\mathcal{B}') = 0.92$.

Exemplo 7.3.5 (Reduction Ratio) *Considerando as coleções de blocagem* $\mathcal{B}_{D_1}$ *e* $\mathcal{B}'_{D_1}$ *apresentadas no Exemplo 7.3.4, a seguinte taxa de redução é produzida:* $RR = 1 - \frac{||\mathcal{B}'_{D_1}||}{||\mathcal{B}_{D_1}||} = 1 - \frac{44}{68} = 0.35$.

Note que a Definição 7.3.5 não especifica restrições associadas ao tamanho dos blocos produzidos na coleção de blocagens podada $\mathcal{B}'$. A seguir, é apresentada a definição da tarefa que consiste em controlar o tamanho dos blocos produzidos.

7.3.3 Poda de Coleção de Blocagens para o Controle do Tamanho dos Blocos

Na Definição 7.3.6, é formalizado o problema de podar uma coleção de blocagens indexada por múltiplas chaves de bloco no intuito de controlar o tamanho dos blocos produzidos.

Definição 7.3.6 (Poda de Coleção de Blocagens para o Controle do Tamanho dos Blocos) *Sejam* D *uma base de dados,* S_{min} *e* S_{max} *dois números inteiros e* $F_{key} = \{f_1, f_2, ..., f_{|F_{key}|}\}$ *um conjunto de funções de chave de bloco inicialmente aplicadas simultaneamente para indexar as entidades em* D *e produzir a coleção de blocagens* $\mathcal{B}$*. O problema consiste em criar uma coleção de blocagens podada* $\mathcal{B}' = \{B_1, B_2, ..., B_{|\mathcal{B}'|}\}$*, tal que para cada bloco* $b \in \cup_{B \in \mathcal{B}'} B : S_{min} \leq |b| \leq S_{max}$ *e* $RR(\mathcal{B}', \mathcal{B})$*,* $PC(\mathcal{B}')$ *e* $PQ(\mathcal{B}')$ *são maximizados.*

O principal desafio deste problema consiste em manipular a coleção de blocagens recebida como entrada para controlar o tamanho dos blocos produzidos e ao mesmo tempo manter resultados de qualidade da coleção de blocagens podada. Em outras palavras, a cardinalidade agregada da coleção de blocagens recebida como entrada é minimizada e a quantidade de entidades duplicadas que podem ser identificadas a partir do resultado de blocagem é maximizada. É também importante notar que, ao podar uma coleção de blocagens, um algoritmo pode reduzir o valor de $PC(\mathcal{B}')$. Este fenômeno acontece sempre que (ao menos) um par de entidades duplicadas que compartilham ao menos um bloco na coleção

de blocagens recebida como entrada ($\mathcal{B}$) passa a não compartilhar blocos na coleção de blocagens podada ($\mathcal{B}$). Portanto, uma solução para o problema apresentado na Definição 7.3.6 deve apresentar um bom balanceamento entre $RR(\mathcal{B}', \mathcal{B})$ e $PC(\mathcal{B}')$.

Outro objetivo que não é explicitamente mencionado na Definição 7.3.6 consiste em minimizar o tempo de execução necessário para podar a coleção de blocagens recebida como entrada $\mathcal{B}$. Uma vez que a etapa de indexação é parte de um processo maior da deduplicação de dados (ver Figura 2.1), ao minimizar o tempo utilizado na etapa de indexação, o tempo de execução total de uma solução para o problema de deduplicação de dados é também reduzido.

7.4 Abordagem proposta

Nesta seção, é apresentada a abordagem proposta para o problema de podar uma coleção de blocagens visando controlar o tamanho dos blocos produzidos (Definição 7.3.6). Para tal, inicialmente são apresentadas as estratégias adotadas (Seções 7.4.1-7.4.3). A visão geral da abordagem proposta é apresentada na Seção 7.4.4. Em seguida, na Seção 7.4.5 é proposta uma metaheurística, denominada *Multiple Keys blocking Pruner for Controlling Block Sizes* (MkPCBS), a qual visa podar uma coleção de blocagens recebida como entrada indexada por múltiplas funções de chave de bloco para produzir blocos de tamanho configurável e manter bons resultados de qualidade de blocagem. Por fim, são propostas cinco heurísticas para solucionar os passos em alto nível definidos na metaheurística MkPCBS.

7.4.1 Co Ocorrência de Entidades em Blocos

Uma vez que o problema investigado (Definição 7.3.6) recebe como entrada uma coleção de blocagens inicialmente indexada por múltiplas chaves de bloco, as heurísticas propostas neste trabalho exploram a coocorrência [62] de entidades nos blocos produzidos. A ideia básica desta intuição é que, quanto mais blocos são compartilhados por duas entidades, maior é a probabilidade destas entidades serem classificadas como duplicatas.

Por exemplo, se as entidades e_1 e e_2 são duplicatas exatas (i.e., e_1 e e_2 possuem exatamente os mesmos valores de atributos), então para qualquer conjunto de $|F_{key}|$ funções de chave de bloco que seja empregado para indexar estas entidades, e_1 e e_2 irão compartilhar

$|F_{key}|$ blocos (no contexto de uma abordagem de blocagem disjunta). Por sua vez, se e_2 e e_3 são duplicatas aproximadas (i.e., e_2 e e_3 tem valores de atributos similares), então quanto mais funções de chave de bloco são empregadas para indexar estas entidades, mais blocos e_2 e e_3 tendem a compartilhar. Por fim, se e_3 ed e_4 são entidades significativamente dissimilares, então se $|F_{key}|$ funções de chave de bloco são empregadas para indexar estas entidades, e_3 e e_4 tendem a compartilhar um número $a \ll |F_{key}|$ de blocos.

7.4.2 Esquema de Pontuação de Co Ocorrência

Em [62], os autores propuseram cinco estratégias para associar uma pontuação para a coocorrência de entidades nos blocos produzidos, as quais são denominadas *Weighting Schemes* (WS). Um esquema de peso visa explorar a seguinte intuição: quanto maior for o tamanho de um bloco, menor deve ser a pontuação associada ao fato de duas entidades coocorrerem neste bloco. Neste trabalho, são exploradas duas definições[5] de esquema de pontuação de coocorrência propostas em [62]: ARCS e ECBS. As definições destes esquemas são apresentadas a seguir.

Definição 7.4.1 (Aggregated Reciprocal Common Scheme (ARCS)) *A pontuação ARCS associada a um par de entidades* (e_1, e_2) *é definida como a soma do inverso das cardinalidades dos blocos compartilhados por* e_1 *e* e_2*. Formalmente,* $ARCS(e_1, e_2) = \sum_{b \in B_{e_1,e_2}} \frac{1}{||b||}$*, tal que* $\forall b \in B_{e_1,e_2} : e_1 \in b \wedge e_2 \in b$*.*

Definição 7.4.2 (Enhanced Common Blocking Scheme (ECBS)) *A pontuação ECBS associada a um par de entidades* (e_1, e_2) *leva em consideração o número total de blocos que são associados a* e_1 *e* e_2*. Formalmente,* $ECBS(e_1, e_2) = |B_{e_1,e_2}|.log\frac{\#\mathcal{B}}{|B_{e_1}|}.log\frac{\#\mathcal{B}}{|B_{e_2}|}$*, tal que* $\forall b \in B_{e_1,e_2} : e_1 \in b \wedge e_2 \in b$*,* $\forall b \in B_{e'} : e' \in b$ *e* $\#\mathcal{B} = \sum_{B \in \mathcal{B}} |B|$*.*

7.4.3 Pontuação de Co Ocorrência Agregada

Nesta seção, são propostas métricas que visam agregar a pontuação de coocorrência de uma entidade em um bloco (Definição 7.4.3) e a pontuação de coocorrência de um bloco (Definition 7.4.4).

[5] estes esquemas produziram resultados experimentais encorajadores em [62]

Definição 7.4.3 (Média de Peso de Co Ocorrência de Entidade) *A média de peso de coocorrência de uma entidade e_1 em um bloco b é calculada como a média de peso de coocorrência entre todos os pares (e_1, e_2) , tal que $e_2 \in b \backslash e_1$, como mostrado na Eq. (7.3).*

$$EntityCooScore(e, b, \mathcal{B}, WS) = \sum_{e_2 \in b \backslash e_1} \frac{WS(e_1, e_2)}{|b|-1} \tag{7.3}$$

, tal que WS é o esquema de pontuação de coocorrência empregado para calcular a pontuação de coocorrência entre e_1 e e_2.

Por sua vez, pontuação de coocorrência média de um bloco é formalizada na Definição 7.4.4.

Definição 7.4.4 (Média de Peso de Co Ocorrência de Bloco) *A média de peso de coocorrência de um bloco b é calculado como a média de pontuação de coocorrência de todas as entidades em b, como mostrado na Eq. (7.4).*

$$BlockCooScore(b, \mathcal{B}, WS) = \frac{\sum_{e \in b} EntityCooScore(e, b, \mathcal{B}, WS)}{|b|} \tag{7.4}$$

, tal que WS é o esquema de pontuação de coocorrência empregado para calcular a pontuação de coocorrência entre as entidades em b.

Desse modo, utilizando a Definição 7.4.4, o problema de podar uma coleção de blocagens para controlar o tamanho dos blocos produzidos (Definição 7.3.6) é explorado com base em heurísticas que visam atingir dois objetivos: controlar o tamanho dos blocos e maximizar a média de peso de coocorrência dos blocos na coleção de blocagens podada. Note que os pesos de coocorrência das entidades e dos blocos podem ser inicialmente calculados e incrementalmente atualizados ao longo da manipulação dos blocos pelas heurísticas de poda.

7.4.4 Visão Geral da Abordagem

A visão geral[6] da abordagem proposta é apresentada na Figura 7.1. A abordagem recebe como entrada uma coleção de blocagens, a qual é gerada por uma etapa de indexação inicial que emprega múltiplas funções de chave de bloco para indexar as entidades na base de dados.

[6] as operações de $merge$ podem ser realizadas utilizando blocos de diferentes resultados de blocagem (ainda que esta possibilidade não seja mostrada na Figura 7.1)

Então, os blocos da coleção de blocagens são reduzidos, i.e., são removidas entidades dos blocos que provavelmente produziriam comparações desnecessárias (i.e., comparações entre entidades não duplicadas). Em seguida, os blocos grandes são ($|b| > S_{max}$) são divididos para garantir um tamanho máximo dos blocos (S_{max}). Prosseguindo, os blocos pequenos ($|b| < S_{min}$) são unidos para garantir um tamanho mínimo dos blocos (S_{min}). Por fim, são excluídos os blocos que provavelmente produziriam uma grande quantidade de comparações desnecessárias entre entidades.

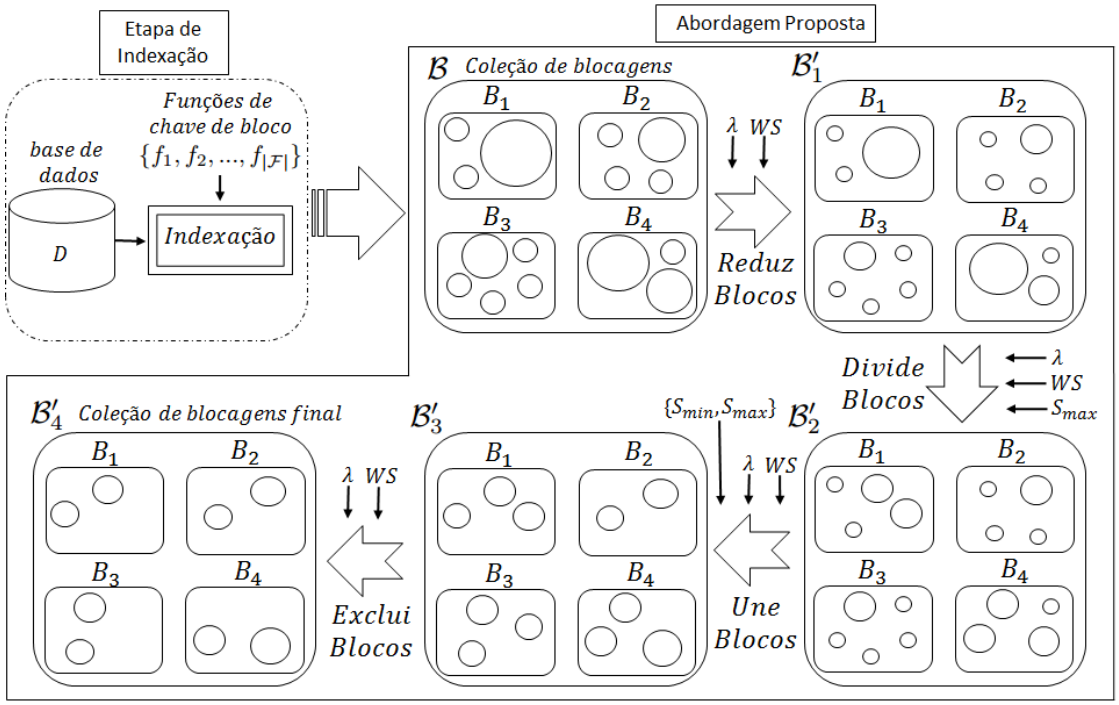

Figura 7.1: Visão geral da abordagem proposta para poda de coleção de blocagens para controle do tamanho de blocos.

As operações $Shrink$, $Split$, $Merge$ e $Exclude$ utilizam os parâmetros WS e λ, os quais representam o esquema de pontuação de coocorrência e o alcance das heurísticas de poda, respectivamente. Na prática, o parâmetro λ pode ser utilizado para calcular a quantidade de entidades a ser removida de um bloco, a quantidade de blocos dos quais uma entidade será removida ou a quantidade de blocos a serem excluídos da coleção de blocagens. Portanto, a semântica do parâmetro λ depende da heurística empregada.

7.4.5 Metaheurística MkPCBS

A metaheurística MkPCBS é utilizada para estruturar os passos em alto nível definidos pela abordagem proposta (Figura 7.1). O pseudocódigo da metaheurística é apresentado no Algoritmo 6. O algoritmo é composto por quatro passos que podem ser substituídos por implementações específicas de heurísticas para realizar as seguintes operações: $ShrinkBlocks$ (linha 3), o qual visa remover entidades dos blocos; ii) $SplitBlocks$ (linha 5), o qual visa dividir blocos para garantir um tamanho máximo (S_{max}) para os blocos; iii) $MergeBlocks$ (linha 7), o qual visa unir blocos para garantir um tamanho mínimo para os blocos (S_{min}); e iv) $ExcludeBlocks$ (linha 9), o qual visa excluir blocos da coleção de blocagens.

Algoritmo 6: Metaheurística $MkPCBS$

Input : $\mathcal{B}$: coleção de blocos de entrada
S_{max}: tamanho máximo dos blocos
S_{min}: tamanho mínimo dos blocos
WS: esquema de peso de coocorrência
λ: porcentagem (0,1) que determina o alcance das heurísticas de poda

Output: $\mathcal{B}'$: coleção de blocos podada contendo blocos com tamanho entre S_{min} e S_{max}

```
1  begin
2      // remove entidades de blocos
3      B' ← ShrinkBlocks(B, WS, λ)
4      // divide blocos para garantir um tamanho máximo
5      B' ← SplitBlocks(B', WS, S_max)
6      // une blocos para garantir um tamanho mínimo
7      B' ← MergeBlocks(B', WS, S_min, S_max)
8      // exclui blocos da coleção de blocagens
9      B' ← ExcludeBlocks(B', WS, λ)
10     return B'
```

As demais heurísticas propostas neste trabalho para controlar o tamanho de blocos são apresentadas na Seção C.5 (Apêndice C).

7.5 Considerações Finais

Neste capítulo, foram propostos definições, métricas e algoritmos para o contexto do problema de controlar o tamanho de blocos produzidos por múltiplas funções de chave de bloco na etapa de indexação de um problema de deduplicação de dados. Foi proposta uma abordagem geral para tratar o problema, a qual envolve a execução de algoritmos para remover entidades de blocos, dividir blocos, unir blocos e excluir blocos. Em seguida, a abordagem proposta foi estruturada na forma de uma metaheurística ($MkPCBS$) e foram propostas cinco heurísticas para tratar os passos genéricos definidos na metaheurística proposta. As heurísticas propostas podem ser exploradas por um SMQD porque o controle do tamanho dos blocos está intrinsecamente relacionado aos custos da deduplicação de dados em nuvem, uma vez que influencia: i) na quantidade de comparações a serem realizadas entre as entidades e, portanto, no custo da etapa de comparação; ii) no custo de classificar as entidades duplicadas (e, por este motivo, no custo da etapa de classificação), pois os algoritmos de classificação são executados por bloco; e iii) na eficácia da execução distribuída do algoritmo de deduplicação de dados, pois paralelizar as comparações de entidades quando os blocos apresentam tamanhos similares reduz significativamente o problema de desbalanceamento de carga entre nós de uma infraestrutura distribuída.

Capítulo 8

Avaliação

Neste capítulo, são apresentados os experimentos conduzidos para avaliar empiricamente as abordagens, algoritmos e heurísticas propostos neste trabalho. Mais especificamente, os experimentos visam avaliar a eficácia e eficiência das estratégias propostas para diminuir custos relacionados à execução de tarefas de deduplicação de dados no contexto de um SMQD. Desse modo, são avaliadas diferentes estratégias para alocação dinâmica de recursos computacionais no contexto de um SMQD, deduplicação incremental de dados utilizando classificação coletiva e controle do tamanho de blocos gerados na etapa de indexação.

8.1 Hipóteses e Experimentos

A avaliação apresentada nesta seção é dividida em quatro experimentos. O Experimento I tem como objetivo avaliar estratégias de alocação dinâmica de recursos computacionais no contexto em que múltiplas tarefas de deduplicação de dados são disparadas por um DQSLA (apresentadas na Seção 5.2.3). Por sua vez, o Experimento II visa avaliar estratégias de alocação dinâmica de recursos computacionais no contexto em que as tarefas de deduplicação de dados são todas disparadas por DQSLAs diferentes (apresentadas na Seção 5.2.4). Em seguida, o Experimento III visa avaliar as abordagens propostas para o problema de deduplicação incremental de dados no contexto de classificação coletiva (apresentadas no Capítulo 6). Por fim, o Experimento IV tem como objetivo avaliar diferentes abordagens para o problema de controle do tamanho dos blocos produzidos por múltiplas funções de chave de bloco no contexto de deduplicação de dados.

O Experimento I visa investigar a seguinte hipótese:

- *H1*: No contexto de tarefas de deduplicação de dados disparadas por um mesmo DQSLA de um SMQD, a adoção das técnicas de aprendizado de máquina avaliadas para alocação dinâmica de recursos computacionais é mais eficaz do que estratégias baseadas nas heurísticas propostas?

Por sua vez, o Experimento II visa investigar as seguintes hipóteses:

- *H2*: No contexto de tarefas de deduplicação de dados disparadas por diferentes DQSLAs em um SMQD, diferentes técnicas de aprendizado de máquina produzem resultados significativamente diferentes?

- *H3*: No contexto de tarefas de deduplicação de dados disparadas por diferentes DQSLAs em um SMQD, a combinação das heurísticas propostas com técnicas de aprendizado de máquina para ajustar as estimativas de classes de configuração é mais eficaz do que a aplicação direta das técnicas de aprendizado de máquina avaliadas?

Em seguida, o Experimento III visa investigar a seguintes hipóteses:

- *H4*: A aplicação de algoritmos eficientes de agrupamento automático produz resultados satisfatórios (comparados aos produzidos pela abordagem do estado da arte) no contexto de DID?

- *H5*: A combinação de algoritmos eficientes de agrupamento automático com algoritmos baseados em minimização de função de penalidade produz resultados satisfatórios no contexto de DID?

- *H6*: A aplicação de filtros de componente de cobertura produz resultados satisfatórios no contexto de DID?

Por fim, o Experimento IV visa investigar a seguintes hipóteses:

- *H7*: Diferentes combinações de heurísticas aplicadas ao problema de controle do tamanho de blocos produzem resultados de qualidade significativamente diferentes?

- *H8*: Diferentes combinações de heurísticas aplicadas ao problema de controle do tamanho de blocos produzem resultados de eficiência significativamente diferentes?
- *H9*: Diferentes valores de λ utilizados por heurísticas para podar coleções de blocagens produzem resultados de qualidade significativamente diferentes?
- *H10*: Diferentes estratégias de pontuação de coocorrência empregadas por heurísticas para podar coleções de blocagens produzem resultados de qualidade significativamente diferentes?
- *H11*: Diferentes estratégias de pontuação de coocorrência empregadas por heurísticas para podar coleções de blocagens produzem resultados de eficiência significativamente diferentes?

8.2 Experimento I

Nesta seção, é apresentada a avaliação de diferentes estratégias para a alocação dinâmica de recursos computacionais no contexto de um SMDQ no intuito de processar diferentes requisições de execução de algoritmos de deduplicação de dados disparados por um mesmo DQSLA (este cenário é avaliado repetidas vezes envolvendo diversos DQSLAs diferentes contidos na carga de trabalho). Desse modo, a avaliação consiste em medir a eficácia de diferentes algoritmos de provisionamento de recursos, considerando os custos do serviço e as penalidades sofridas, os quais são gerados com base no Algoritmo 2 (metaheurística *Hill Climbing Provisioning*).

8.2.1 Design

A carga de trabalho (gerada sinteticamente) empregada no Experimento I é mostrada na Tabela 8.1, na qual são apresentados os tamanhos das bases de dados processadas, as características dos blocos gerados a partir da blocagem padrão (referentes ao problema de deduplicação de dados), a quantidade de clientes e de DQSLAs processados e a quantidade média de execuções por DQSLA.

Os diferentes algoritmos de provisionamento avaliados neste experimento são mostrados na Tabela 8.2. Nesta tabela, são apresentadas as abreviações de 12 diferentes algorit-

Tabela 8.1: Caracterização da Carga de Trabalho do Experimento I.

Parâmetro	**Valor**
período	1 ano
#clientes	100
#DQSLAS por cliente	random(1, 20)
|D| (quantidade de entidades)	random(100, 3×10^7)
quantidade média de entidades por bloco	100
Algoritmo de Indexação	Blocagem padrão
média de #execuções por mês	121
#execuções	1452

mos gerados a partir da modificação das chamadas intercambiáveis da metaheurística *Hill Climbing Provisioning*. Desse modo, são avaliados quatro algoritmos baseados em heurísticas (heur#(1-4)), quatro algoritmos baseados na técnica de aprendizado de máquina KNN (knn#(1-4)) e quatro algoritmos baseados na seleção de protótipos (proto#(1-4)). Por sua vez, na Tabela 8.3, são apresentados os valores dos parâmetros globais utilizados no experimento.

Assim, o design do experimento consistiu em utilizar cada um dos algoritmos listados na Tabela 8.2 para processar separadamente a carga de trabalho descrita na Tabela 8.1, utilizando os parâmetros apresentados na Tabela 8.3.

8.2.2 Modelo de Execução

Nesta seção, é apresentado o modelo matemático adotado para simular a execução dos algoritmos de deduplicação de dados no Experimento I. A notação adotada no modelo é mostrada na Tabela 8.4 e o modelo é apresentado na Eq. (8.1). Em resumo, o modelo visa estimar a influência da classe de configuração (ou seja, o número N de nós e a configuração γ dos nós) adotada no ambiente distribuído sobre o tempo de execução de uma tarefa de deduplicação de dados, dado um valor de *speedup*, o qual visa simular os atrasos gerados em decorrência de fatores como o tempo de comunicação entre os nós na rede e o problema de desbalanceamento de carga [11]. Este segundo problema ocorre quando algum nó específico causa um gargalo no tempo de execução da tarefa como um todo. Além disso, note que o parâmetro η

Tabela 8.2: Chamadas intercambiáveis dos algoritmos de provisionamento utilizados no Experimento I.

Algoritmo	**InitialConfigurationClass**	**Tweak**
heur#1	$\lfloor(1 + L_{index})/2\rfloor$	Binary
heur#2	randomClass(1, L_{index})	Sliced
heur#3	$\lfloor(1 + L_{index})/2\rfloor$	Sliced
heur#4	$\lfloor(1 + L_{index})/2\rfloor$	Tunable
knn#1	kNearestNeighbors(τ, TD, 1, Euc)	Tunable
knn#2	kNearestNeighbors(τ, TD, 7, Euc)	Tunable
knn#3	kNearestNeighbors(τ, TD, 1, Cos)	Tunable
knn#4	kNearestNeighbors(τ, TD, 7, Cos)	Tunable
proto#1	Prototype(τ, TB, 3, wGeoMean, Euc)	Tunable
proto#2	Prototype(τ, TD, 7, wGeoMean, Euc)	Tunable
proto#3	Prototype(τ, TB, 3, Median, Cos)	Tunable
proto#4	Prototype(τ, TD, 7, Median, Cos)	Tunable

Tabela 8.3: Valores de parâmetros globais utilizados no Experimento I.

Parâmetro	**Local**	**Valor**
thr	Algoritmo 2	$0.2 \times T_{res}$
α	Eq. (5.3)	\$100
β	Eq. (5.3)	\$0.2 por minuto
F	Algoritmo 8	10
γ (#vCPUs)	Seção 5.1	{1, 2, 3, 4}
s ($speedupfactor$)	Eq. (8.1)	random(1, 3)
N	Seção 5.1	{1, 2, . . . , 100}
L	Algoritmos{2, 3, 4, 5}	400 ($\lvert\mathcal{C}_{conf}\rvert$)
$\lvert TD\rvert$	Definição 5.2.1	252 execuções

Tabela 8.4: Notação do modelo matemático adotado no Experimento I.

D	Base de dados a ser monitorada
$\Lambda(\tau)$	Número de comparações a serem realizadas pela tarefa τ
η	Custo de comparação entre duas entidades
γ	Número de vCPUs
N	Número de nós (VM's)
s	valor de *speedup*

pode ser estimado empiricamente ao empregar uma única comparação entre duas entidades na base de dados utilizando um nó que possui um único processador ($vCPU$).

A implementação do ambiente simulado do SMQD, a carga de trabalho, o modelo matemático e os algoritmos utilizados no Experimento I foram implementados na linguagem de programação Java. Por fim, os custos de infraestrutura (usado na Eq. (5.1)) foram calculados com base nos preços empregados pela empresa Amazon[1].

$$ExecTime(\tau) = \begin{cases} \dfrac{\eta(\tau) \times \Lambda(\tau)}{N(\tau) \times \gamma(\tau)} & \text{se } N(\tau) = 1 \\ \\ \dfrac{\eta(\tau) \times \Lambda(\tau)}{s^{-1} \times N(\tau) \times \gamma(\tau)} & \text{se } N(\tau) > 1 \end{cases} \tag{8.1}$$

8.2.3 Resultados

Na Figura 8.1, são reportados os custos do serviço (custo de infraestrutura + custos de penalidade) produzidos pelo algoritmo mais eficaz (i.e., que produziu o menor valor de custo acumulado do serviço) de cada grupo ($heur\#*$, $knn\#*$ e $proto\#*$ na Tabela 8.2) avaliado. Por sua vez, na Figura 8.2, é apresentada a quantidade de violações de SLA produzida pelo algoritmo mais eficaz de cada grupo avaliado.

[1] https://aws.amazon.com/pt/ec2/pricing/

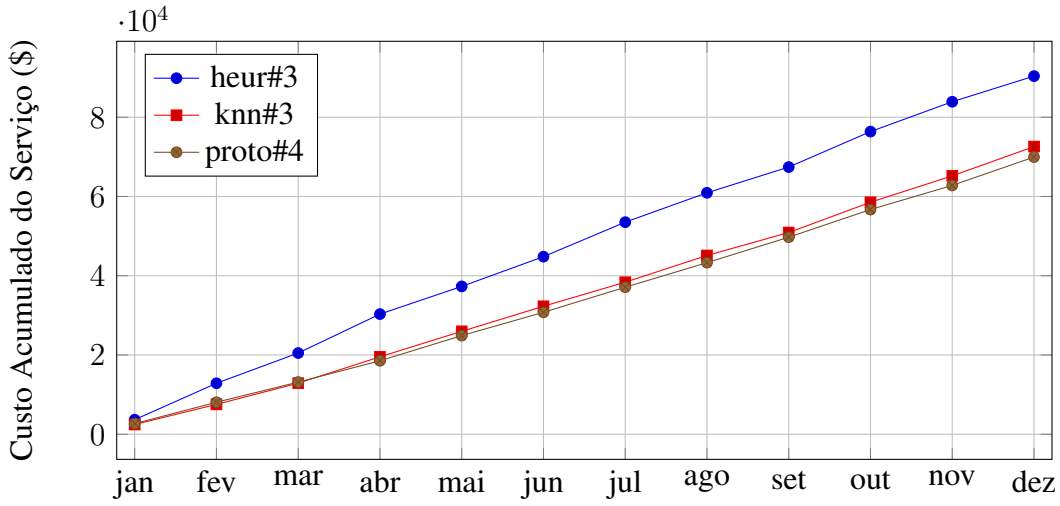

Figura 8.1: Custo acumulado do serviço produzido pelo algoritmo mais eficaz de cada grupo avaliado.

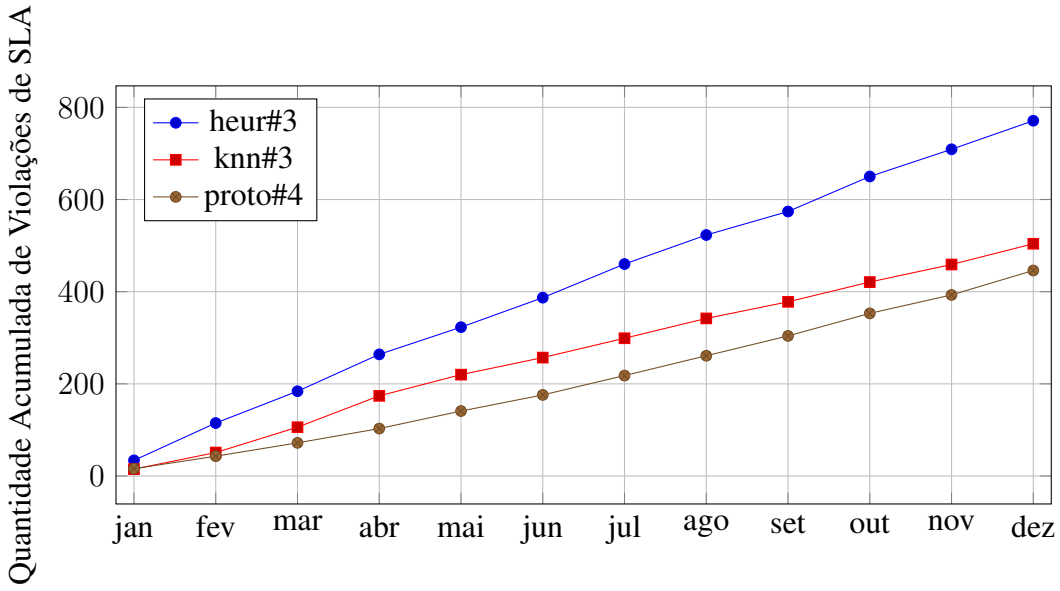

Figura 8.2: Quantidade de violações de SLA produzida pelo algoritmo mais eficaz de cada grupo avaliado.

8.2.4 Discussão

Ao analisar os dados apresentados na Figura 8.1, é possível observar que os algoritmos de provisionamento baseados em técnicas de aprendizado de máquina produziram custos para o SMQD inferiores aos custos produzidos pelos algoritmos baseados nas heurísticas propostas. Este resultado é melhor explicado pelo fato de as técnicas de aprendizado de máquina proverem aos algoritmos de provisionamento duas vantagens principais em relação aos algoritmos baseados em heurísticas: i) a estimativa de uma boa classe de configuração inicial; e ii) como consequência da primeira vantagem, uma quantidade menor de ajustes na classe de configuração estimada para atingir o resultado ótimo.

Os algoritmos de provisionamento baseados em aprendizado de máquina que apresentaram melhores resultados foram knn#3 e $proto$#4 (ver Tabela 8.2), os quais produziram resultados inferiores ao algoritmo de provisionamento baseado em heurísticas que alcançou a melhor eficácia no experimento conduzido ($heur$#3). É interessante notar que os algoritmos knn#3 e $proto$#4 empregam como função de similaridade, para calcular os vizinhos mais próximos na base de treinamento, a distância de cosseno. Além disso, o algoritmo baseado no vizinho mais próximo (knn#3) produziu melhores resultados do que seu algoritmo equivalente (knn#4), mas que emprega 7 vizinhos como parâmetro de entrada. Outrossim, note que a função de agregação utilizada pelo algoritmo $proto$#4 é a mediana, a qual apresentou-se mais eficaz do que a função de agregação média geométrica ponderada (empregada pelo algoritmo $proto$#2, por exemplo).

Com base nos resultados mostrados na Figura 8.2, é possível notar que os algoritmos de provisionamento baseados em técnicas de aprendizado de máquina também produziram uma menor quantidade de violações de restrições de tempo especificadas nos SLAs ao longo do tempo, quando comparados ao algoritmo ($heur$#3) baseado em heurísticas que apresentou o melhor resultado considerando a quantidade de violações dos SLAs. Além disso, ao analisar a curva que representa a quantidade de violações de SLAs produzidas pelos três algoritmos mostrados na Figura 8.2, é possível notar que a quantidade de violações produzidas pelo algoritmo baseado em heurísticas cresce de maneira aproximadamente constante ao longo do processamento dos SLAs. Por sua vez, os algoritmos de provisionamento baseados em técnicas de aprendizado de máquina apresentam uma curva de crescimento, referente ao número de violações de SLA, inferior ao longo do tempo. Este fenômeno pode ser melhor explicado

pela ocorrência de dois fatores: i) o fato dos algoritmos de provisionamento baseados em técnicas de aprendizado de máquina produzirem melhores estimativas de classes de configuração iniciais tende a minimizar a violação de SLAs produzida pela primeira execução de uma tarefa de deduplicação de dados; e ii) o fato de a base de treinamento utilizada pelos algoritmos baseados em técnicas de aprendizado de máquina ser constantemente atualizada e aumentada ao longo do tempo potencializa o efeito explicado no item i.

Desse modo, os resultados dos experimentos indicam que, para o processamento de tarefas de qualidade de dados bastante custosas, a utilização de algoritmos de provisionamento baseados em técnicas de aprendizado de máquina tende a produzir resultados mais eficazes (tanto em relação aos custos quanto à quantidade de violações de SLA) do que algoritmos baseados nas heurísticas propostas. Portanto, os resultados obtidos dão suporte para a hipótese $H1$. No entanto, para cenários que exigem o processamento de uma pequena quantidade de tarefas de qualidade de dados e/ou processamento de tarefas pouco custosas, a adoção de algoritmos de provisionamento baseados em heurísticas pode ser uma estratégia promissora, uma vez que este tipo de algoritmo não requer a utilização de uma base de treinamento, a qual pode ser custosa de ser inicialmente produzida em um cenário prático.

8.3 Experimento II

Nesta seção, é apresentada a avaliação das estratégias propostas para a alocação dinâmica de recursos computacionais no contexto de um SMQD no intuito de processar requisições de execução de algoritmos de deduplicação de dados que são todas disparadas por diferentes DQSLAs. A avaliação consiste em medir a eficácia de diferentes algoritmos de provisionamento de recursos computacionais, considerando os custos de infraestrutura e a quantidade de violações de SLA produzidos no processo.

8.3.1 Design

A carga de trabalho utilizada no Experimento II é mostrada na Tabela 8.5, na qual são apresentados o tamanho das bases de dados processadas, a quantidade de tarefas de deduplicação de dados, as características do blocos gerados a partir da blocagem padrão, o tamanho da janela adotado na técnica de indexação janela deslizante e o custo unitário de comparação

entre entidades. Por sua vez, os valores de parâmetros globais utilizados no Experimento II são mostrados na Tabela 8.6. Por fim, os algoritmos de provisionamento avaliados neste experimento são mostrados na Tabela 8.7.

Note que a carga de trabalho empregada neste experimento é especialmente desafiadora porque a mesma é composta por tarefas de deduplicação de dados disparadas por diferentes DQSLAs. Por este motivo, os algoritmos de provisionamento avaliados neste experimento devem se basear exclusivamente na base de treinamento adotada pelo sistema de decisão para tarefas de deduplicação de dados (Definição 5.2.1) e (opcionalmente) nos resultados produzidos pelas estimativas de classes de configuração realizadas ao longo do tempo.

Tabela 8.5: Carga de trabalho do Experimento II.

Parâmetro	**Valor**
$\|\mathcal{T}\|$ (conjunto de tarefas de deduplicação de dados)	1,5$\times 10^3$
$\|\mathcal{D}_d\|$ (uma base de dados a ser avaliada em $\mathcal{T}$)	$random(3 \times 10^4 \ldots\ 3 \times 10^7)$
Método de Indexação	$random$(Sorted Neighborhood (SN), Standard Blocking (SB))
Chave de bloco da base de dados D_d	bk_d
b (quantidade de blocos gerados ao aplicar bk_d em $\mathcal{D}_d$)	10^2
w (tamanho da janela fixa)	10^3
$\Lambda(\mathcal{D}SB, b)$	$\frac{\|\mathcal{D}\|}{2}\left(\frac{\|\mathcal{D}\|}{b} - 1\right)$ [13]
$\Lambda(\mathcal{D}, SN, w)$	$(w-1) \times (\|\mathcal{D}\| - \frac{w}{2})$ [13]
$\eta(\mathcal{D})$	$random(10^{-1} \ldots\ 3 \times 10^{-1})$ (segundos)

O design experimental adotado consiste em gerar uma carga de trabalho (WL) como mostrada na Tabela 8.5 e processar as mesmas tarefas de deduplicação de dados presentes

Tabela 8.6: Valores de parâmetros globais utilizados no Experimento II.

Parâmetro	**Localidade**	**Valor**
$overThr$	Algoritmos {2, 11, 12, 13}	$0.2 \times T_{res}(\tau)$
ϕ	Algoritmos {12, 13}	$10^{-1} \times \lvert\mathcal{C}_{conf}\rvert$
δ	Algoritmo 13	10^{-2}
γ (#vCPUs)	Seção 5.1	{1, 2, 3, 4}
N	Seção 5.1	{1, 2, . . . , 100}
$\lvert TD\rvert$	Definição 5.2.1	6×10^2 execuções

em WL, usando os valores de parâmetros globais mostrados na Tabela 8.6, empregando as técnicas de aprendizado de máquina consideradas neste trabalho (KNN, RF, SVM), assim como a combinação destas técnicas com as heurísticas propostas (*Best Performing Allocatiom*, *Probabilistic Best Performing Allocatiom*, *Tunable Allocation*, *Adaptive Allocation* e *Sliced Training Data*).

O design é sumarizado na Tabela 8.7. A notação $\{alg_1, alg_2, \ldots, alg_n\}$ significa que os algoritmos são individualmente empregados no intuito de estimar as classes de configuração utilizadas para processar as tarefas em WL. Por sua vez, a notação $[alg_1, alg_2, \ldots, alg_n]$ significa que os algoritmos são coletivamente considerados por uma heurística no intuito de processar as tarefas em WL. Por exemplo, a segunda linha da Tabela 8.7 define que os algoritmos 1NN e 5NN são separadamente utilizados no intuito de processar WL. Por sua vez, a sétima linha da Tabela 8.7 define que a combinação dos algoritmos $[1NN(d.f. = E), 5NN(d.f. = E), 1NN(d.f. = M), 5NN(d.f. = M), RF(\#Tree = 5), RF(\#Tree = 10), SVM]$ com a heurística *Best Performing Allocation* é utilizada para processar WL. Por fim, os valores dos parâmetros dos algoritmos de aprendizado de máquina empregados no Experimento II foram definidos a partir de resultados experimentais.

8.3.2 Modelo de execução

Nesta seção, é apresentado o modelo matemático adotado para simular a execução dos algoritmos de deduplicação de dados no Experimento II. Em resumo, o modelo visa estimar a influência da classe de configuração (ou seja, o número de nós N e a configuração γ dos

Tabela 8.7: Design experimental do Experimento II.

Algoritmo de Provisionamento	**Heurística**	**Parâmetros**
$\{Over_{100}, Over_{75}\}$ (abordagens básicas)	-	-
{1NN, 5NN}	-	função de distância = Euclidiana
{1NN, 5NN}	-	função de distância = Manhattan
Random Forests (RF)	-	#Tree = 5
Random Forests (RF)	-	#Tree = 10
[1NN(d.f.=E), 5NN(d.f.=E), 1NN(d.f.=M), 5NN(d.f.=M), RF(#Tree=5), RF(#Tree=10), SVM]	*Best Performing Allocation* (BPA)	optInc = 1
SVM	-	kernel = Polinomial Normalizado
[1NN(d.f.=E), 5NN(d.f.=E), 1NN(d.f.=M), 5NN(d.f.=M), RF(#Tree=5), RF(#Tree=10), SVM]	*Best Performing Allocation* (BPA)	optInc = 0
{5NN(d.f.=E), RF(#Tree=5), SVM}	*Tunable Allocation*	$\phi = 10^-1 \times \lvert\mathcal{C}_{conf}\rvert$
[1NN(d.f.=E), 5NN(d.f.=E), 1NN(d.f.=M), 5NN(d.f.=M), RF(#Tree=5), RF(#Tree=10), SVM]	*Probabilistic Best Performing Allocation* (PBPA)	-
{5NN(d.f.=E), RF(#Tree=5), SVM}	*Adaptive Allocation*	$\phi = 10^-1 \times \lvert\mathcal{C}_{conf}\rvert$, $\delta = 10^-2$
{5NN(d.f.=E), RF(#Tree=5), SVM}	*Sliced Training Data*	-

nós) adotada no ambiente distribuído sobre o tempo de execução de uma tarefa de deduplicação de dados, dado um valor de *speedup*, o qual visa simular os atrasos típicos gerados em decorrência de fatores como o tempo de comunicação entre os nós na rede e o problema de desbalanceamento de carga [11]. Este segundo problema ocorre quando são alocadas sub tarefas de maior complexidade para alguns nós computacionais específicos e, por este motivo, é acarretado um aumento no tempo de execução da tarefa como um todo. Por fim, note que o parâmetro η pode ser estimado empiricamente ao empregar uma única comparação entre duas entidades na base de dados (seguindo a especificação do conteúdo do parâmetro M_{rules} do DQSLA) utilizando um nó que possui um único processador (vCPU).

Dada uma tarefa de deduplicação de dados τ e um algoritmo de provisionamento alg, então $alg(\tau).N$ e $alg(\tau).\gamma$ representam a quantidade de máquinas virtuais e v$CPUs$ (respectivamente) associada a uma classe de configuração $alg(\tau) \in \mathcal{C}_{conf}$ estimada pelo algoritmo de provisionamento alg para a execução de τ. Empregando esta notação, na Eq. (8.2) é apresentado o modelo de execução utilizado para estimar o tempo de execução serial (ou seja, quando $alg(\tau).N = 1$)) da tarefa τ.

Por sua vez, na Eq. (8.3) é mostrada a relação entre o tempo de execução serial de uma tarefa de qualidade de dados e a fórmula de $speedup$ [44], a qual permite estimar o tempo de execução da tarefa (quando executada em um sistema distribuído, ou seja, $alg(\tau).N > 1$), dado o valor de $speedup$ ($speedUp(dedup\_method(\tau), alg(\tau).N)$) e a quantidade de nós alocados ($alg(\tau).N$). Na Eq. (8.4), é apresentado o fator de $speedup$, o qual é utilizado para representar a influência da execução distribuída (ou seja, quando $alg(\tau).N > 1$) sobre o tempo de execução estimado ($ExecTime$). Finalmente, os modelos adotados para estimar o tempo de execução de uma tarefa de deduplicação de dados utilizando os métodos de indexação Blocagem Padrão (*Standard Blocking* - SB) e Janela Deslizante (*Sorted Neighborhood* - SN) são mostrados na Eq. (8.5) e Eq. (8.6), respectivamente.

$$SerialExecTime(alg, \tau) = \frac{\eta(\tau) \times \Lambda(\mathcal{D}, dedup\_method)}{alg(\tau).\gamma} \tag{8.2}$$

$$\frac{SerialExecTime(alg,\tau)}{ExecTime(alg,\tau)} = speedUp(dedup\_method(\tau), alg(\tau).N)$$
$$ExecTime(alg,\tau) = \frac{SerialExecTime(alg,\tau)}{speedUp(dedup\_method(\tau), alg(\tau).N)} \tag{8.3}$$

$$sF(alg(\tau).N,\tau) = \frac{1}{speedUp(dedup\_method(\tau), alg(\tau).N)} \tag{8.4}$$

$$ExecTime(alg,\tau,Standard\_Blocking,b) = \begin{cases} \frac{\eta(\tau)\times\Lambda(\mathcal{D},SB,b)}{alg(\tau).\gamma} & \text{se } alg(\tau).N = 1 \\ \frac{\eta(\tau)\times\Lambda(\mathcal{D},SB,b)}{alg(\tau).\gamma} \times sF(alg(\tau).N,\tau) & \text{se } alg(\tau).N > 1 \end{cases} \tag{8.5}$$

$$ExecTime(alg,\tau,Sorted\_Neighborhood,w) = \begin{cases} \frac{\eta(\tau)\times\Lambda(\mathcal{D},SN,w)}{alg(\tau).\gamma} & \text{se } alg(\tau).N = 1 \\ \frac{\eta(\tau)\times\Lambda(\mathcal{D},SN,w)}{alg(\tau).\gamma} \times sF(alg(\tau).N,\tau) & \text{se } alg(\tau).N > 1 \end{cases} \tag{8.6}$$

No intuito de estimar os valores de *speedup* utilizados pelos modelos de execução (Eq. (8.5) e Eq. (8.6)), foi aplicada a técnica de regressão linear empregando dados de execução de tarefas de deduplicação de dados processadas em um ambiente real de computação em nuvem. Os dados utilizados foram produzidos por dois algoritmos do estado da arte (***RepSN*** [42] e ***Block Slicer*** [52]), propostos para tratar a execução distribuída com balanceamento de carga das técnicas de indexação $Sorted\ Neighborhood$ (SN) (janela fixa) e $Standard\ Blocking$ (SB), respectivamente. Com base nos resultados da regressão linear, os valores de $speedUp(SB, alg(\tau).N)$ e $speedUp(SN, alg(\tau).N)$ podem ser estimados utilizando a Eq. (8.7) e Eq. (8.8), respectivamente.

$$speedUp(SB, alg(\tau).N) = (-0,3834) + (0,7039 \times alg(\tau).N) \tag{8.7}$$

$$speedUp(SN, alg(\tau).N) = (0,3633) + (0,7100 \times alg(\tau).N) \tag{8.8}$$

8.3.3 Implementação

O ambiente simulado do SMQD, a carga de trabalho, o modelo matemático e os algoritmos utilizados no Experimento II foram implementados na linguagem de programação Java. As técnicas de aprendizado de máquina foram empregadas por meio da utilização da API da ferramenta WEKA [89]. Por fim, a referência de preço adotada pela Amazon EC2 foi utilizada para estimar os custos de infraestrutura (Eq. (5.1)) das execuções dos algoritmos de deduplicação de dados.

No intuito de evitar o enviesamento dos resultados, não foram atribuídos valores específicos aos parâmetros α (penalidade fixa) e β (taxa de penalidade), ambos utilizados pela Eq. (5.3). Ao invés disso, os resultados obtidos são apresentados na forma de quantidade de violações de SLA produzidas pelos algoritmos de provisionamento. Desse modo, para cada algoritmo de provisionamento avaliado, são reportados o custo de infraestrutura acumulado (para processar 500, 1.000 e 1.500 tarefas) e a quantidade de violações de SLA produzidas ao fim do processamento das 1.500 tarefas.

8.3.4 Resultados

O custo acumulado do serviço e a quantidade de violações de SLA produzidos pelos algoritmos 1NN(d.f.=E), 5NN(d.f.=E), 1NN(d.f.=M), 5NN(d.f.=M), RF(#Tree=5), RF(#Tree=10), SVM, $Over_{75}$ e $Over_{100}$ são mostrados nas Figuras 8.3 e 8.4, respectivamente. Em seguida, o custo acumulado do serviço e a quantidade de violações de SLA produzidos pelos algoritmos 5NN(d.f.=E), RF(#Tree=10) e SVM combinados com as heurísticas BPA(inc=1), BPA(inc=0) e PBPA são apresentados nas Figuras 8.5 e 8.6, respectivamente. Prosseguindo, o custo acumulado do serviço e a quantidade de violações de SLA produzidos pelos algoritmos 5NN(d.f.=E), RF(#Tree=10) e SVM combinados com a heurística *tunable allocation* são mostrados nas Figuras 8.7 e 8.8, respectivamente.

Por sua vez, o custo acumulado do serviço e a quantidade de violações de SLA produzidos pelos algoritmos 5NN(d.f.=E), RF(#Tree=10) e SVM combinados com a heurística *adaptive allocation* são mostrados nas Figuras 8.9 e 8.10, respectivamente. Por fim, o custo acumulado do serviço e a quantidade de violações de SLA produzidos pelos algoritmos 5NN(d.f.=E), RF(#Tree=10) e SVM combinados com a heurística *sliced training data* são

mostrados nas Figuras 8.11 e 8.12, respectivamente.

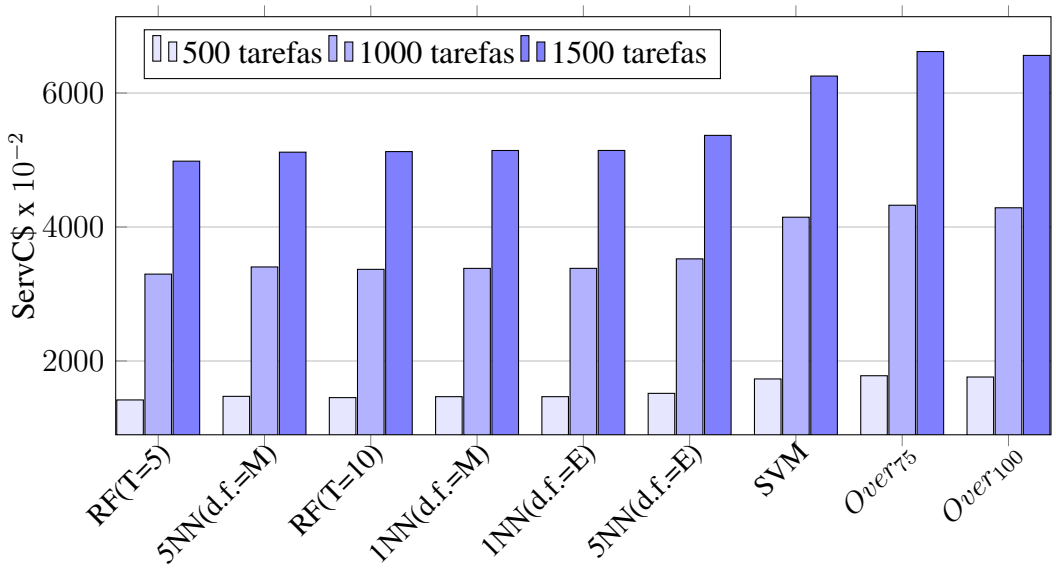

Figura 8.3: Custo acumulado do serviço produzido, por algoritmo de provisionamento.

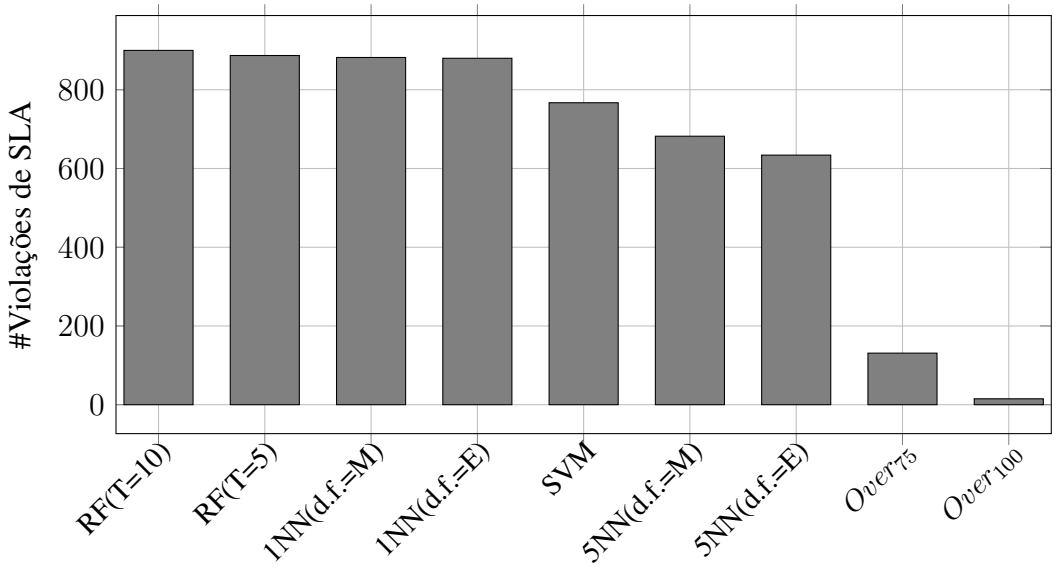

Figura 8.4: Quantidade de violações de SLA produzida, por algoritmo de provisionamento.

8.3.5 Discussão

De acordo com os resultados obtidos no Experimento II, em todos os cenários avaliados, as abordagens básicas ($Over_{75}$ e $Over_{100}$) produziram custos (calculados por meio da Eq. (5.1)) superiores em relação aos custos gerados pelos algoritmos baseados em aprendizado

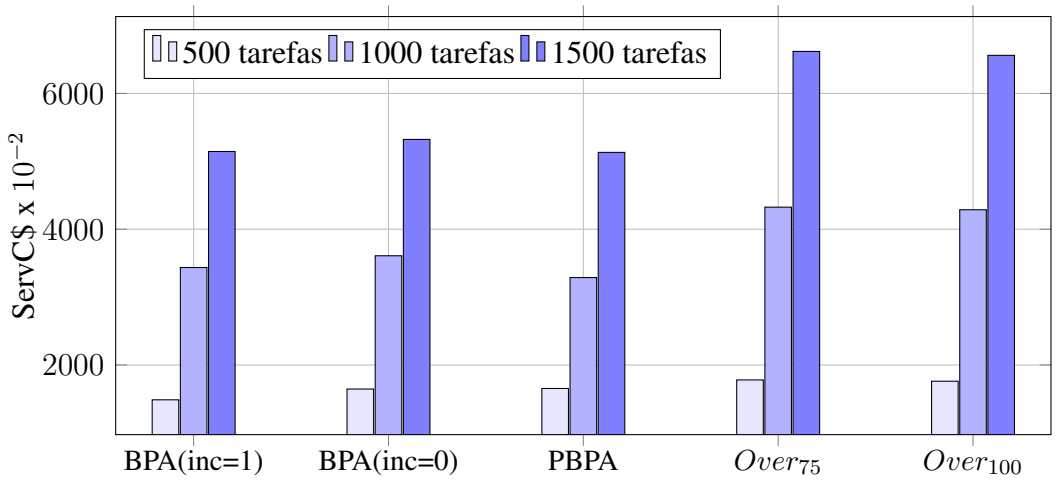

Figura 8.5: Custo acumulado do serviço produzido, por algoritmo de provisionamento.

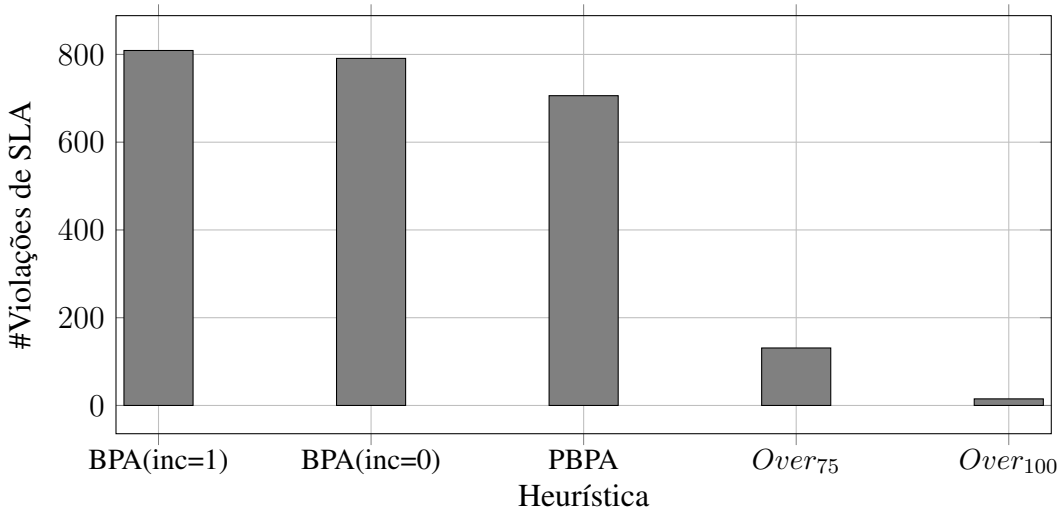

Figura 8.6: Quantidade de violações de SLA produzida, por algoritmo de provisionamento.

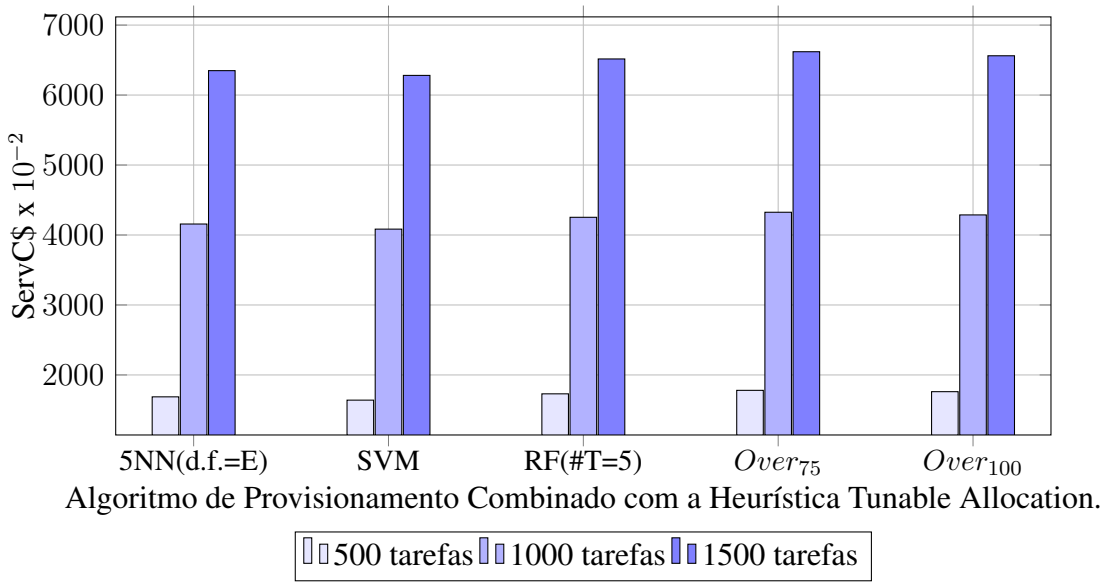

Figura 8.7: Custo acumulado do serviço produzido, por algoritmo de provisionamento.

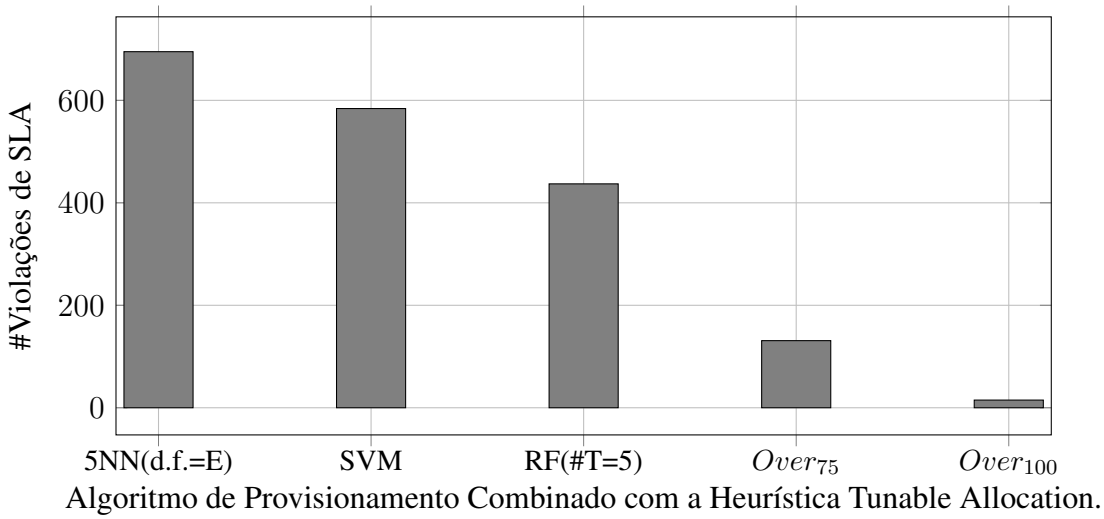

Figura 8.8: Quantidade de violações de SLA produzida, por algoritmo de provisionamento.

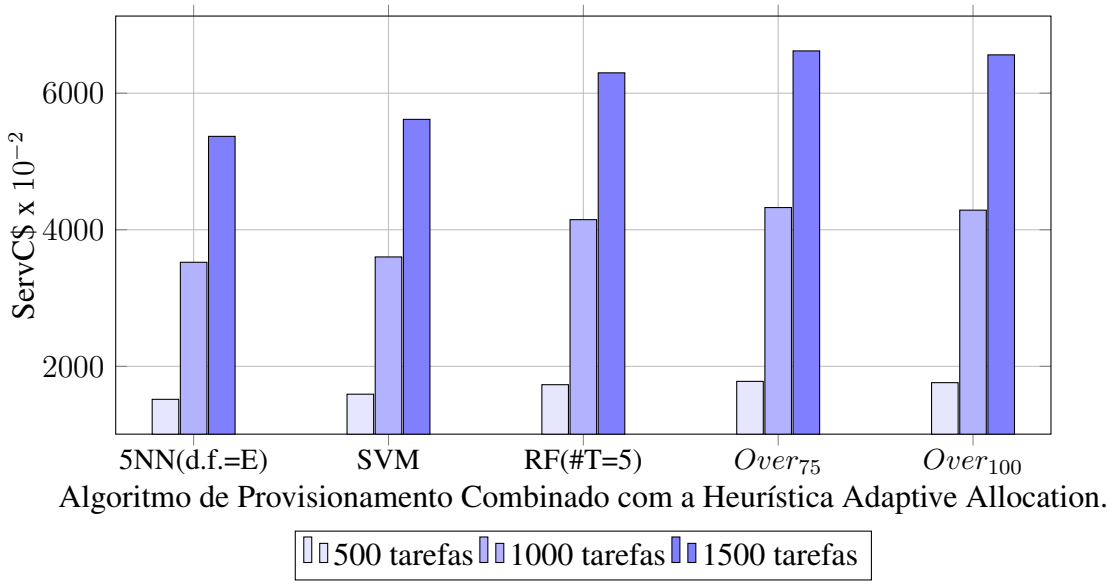

Figura 8.9: Custo acumulado do serviço produzido, por algoritmo de provisionamento.

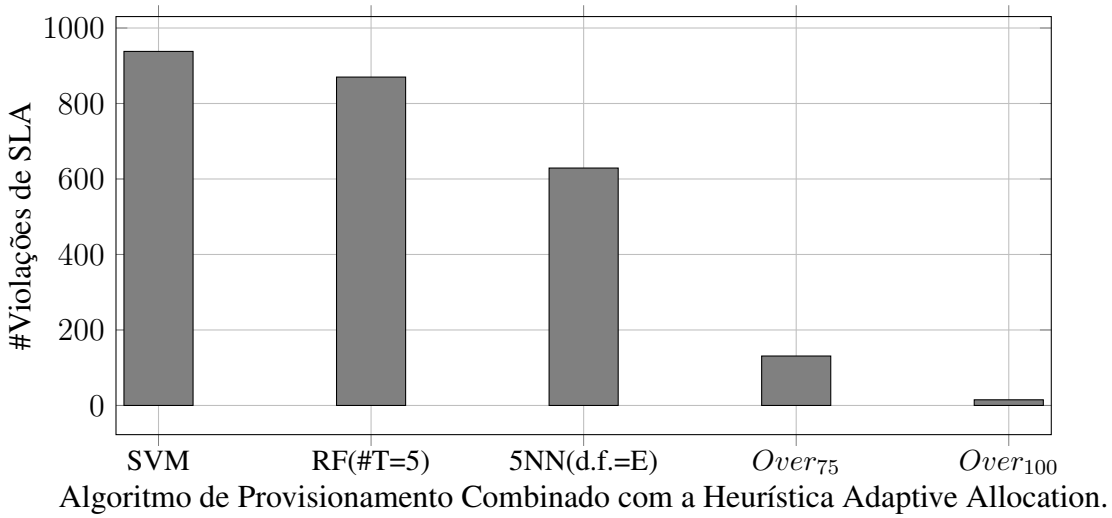

Figura 8.10: Quantidade de violações de SLA produzida, por algoritmo de provisionamento.

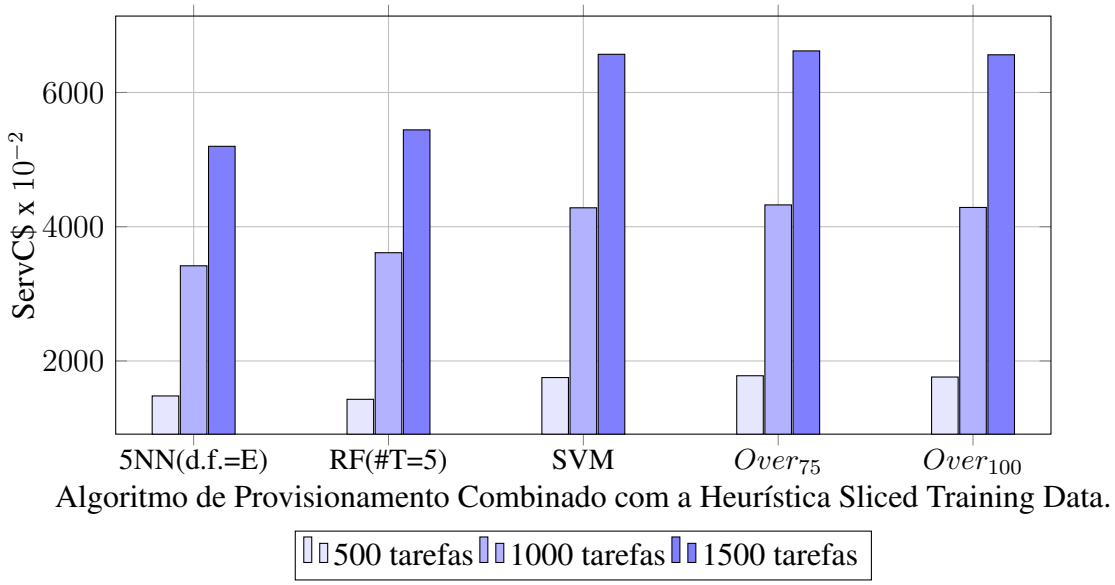

Figura 8.11: Custo acumulado do serviço produzido, por algoritmo de provisionamento.

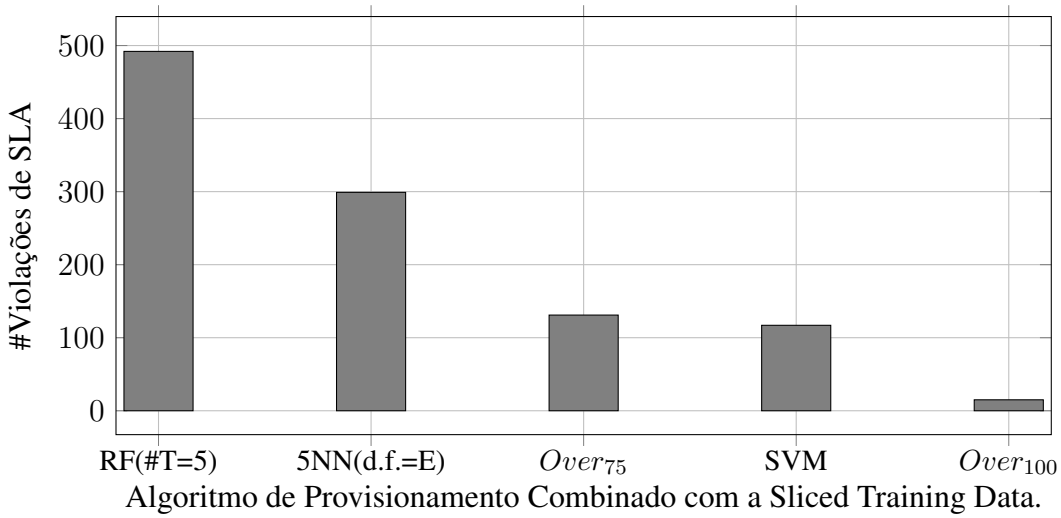

Figura 8.12: Quantidade de violações de SLA produzida, por algoritmo de provisionamento.

de máquina (por exemplo, na Figura 8.3). No entanto, apesar de terem produzido custos inferiores, os algoritmos baseados em aprendizado de máquina geraram uma quantidade maior de violações (#v) de SLA (em comparação com as abordagens básicas).

Em um cenário real, estas violações produziriam penalidades (por exemplo, por meio da aplicação da Eq. (5.3)) para o provedor do serviço. Por este motivo, um algoritmo de provisionamento eficaz deve apresentar um balanceamento entre estas duas saídas (custos de infraestrutura e penalidades). Por exemplo, ainda que as abordagens básicas tenham produzido uma pequena quantidade de violações de SLA, a aplicação destes algoritmos em um cenário prático é desencorajada, pois estes algoritmos empregam toda (ou uma parte significativa) da infraestrutura de hardware disponível para executar cada tarefa de qualidade de dados, o que torna impraticável a execução de múltiplas tarefas ao mesmo tempo e, por conseguinte, a utilização do SMQD por múltiplos clientes. Desse modo, algoritmos de provisionamento que produzam saídas mais balanceadas são preferíveis.

Considerando os resultados apresentados na Figura 8.3 e na Figura 8.4, a abordagem baseada na técnica $random\ forests$ (#Tree=5) produziu o menor custo de infraestrutura para processar as tarefas contidas na carga de trabalho WL. No entanto, note que este algoritmo também produziu a maior quantidade de violações de SLA (887) quando comparada com a quantidade de violações produzidas por outras abordagens baseadas em aprendizado de máquina (por exemplo, as variações do algoritmo knn avaliadas produziram menos de 700 violações de SLA). É também importante observar a influência dos valores dos parâmetros dos algoritmos de aprendizado de máquina sobre os resultados produzidos por estes algoritmos. Primeiramente, a quantidade de árvores (#T) apresentou pouca influência no comportamento do algoritmo $random\ forests$ no contexto do problema de provisionamento de recursos computacionais em um SMQD. Similarmente, a efetividade do algoritmo KNN não foi muito afetada pela função de similaridade adotada (por exemplo, note a similaridade entre as saídas produzidas pelos algoritmos 1NN(d.f=M) e 1NN(d.f.=E) na Figura 8.3).

Porém, o valor de K influenciou de maneira significativa a efetividade do algoritmo KNN. Por exemplo, note que a diferença entre as quantidades de violações de SLA produzidas pelas abordagens $5NN(d.f. = E)$ e $1NN(d.f. = E)$ foi 248, o que é uma quantidade bastante significativa levando em consideração a quantidade total (1500) de tarefas em WL. Além disso, note que a efetividade da técnica SVM foi bastante baixa (ou seja, esta téc-

nica produziu altos valores de infraestrutura e uma grande quantidade de violações de SLA). Uma possível explicação para este fenômeno é a inadequação da abordagem SVM para a quantidade reduzida de características que quantifica uma tarefa de deduplicação de dados (juntamente com a restrição de tempo associada). Por fim, é importante destacar as saídas produzidas pelos algoritmos $5NN(d.f. = M)$ e $5NN(d.f. = E)$, pois estes algoritmos reduziram significativamente os custos de infraestrutura (*VMC$*) (quando comparados aos custos produzidos pelas outras técnicas de aprendizado de máquina), assim como geraram uma quantidade reduzida de violações de SLA.

Considerando os dados mostrados na Figura 8.5 e na Figura 8.6, é possível observar que o valor do parâmetro $optInc$ não produziu efeitos significativos sobre as saídas geradas pela heurística BPA, uma vez que a similaridade das saídas produzidas pelas abordagens (BPA(optInc=0) e BPA(optInc=1)) é bastante alta. No entanto, a heurística probabilística (PBPA) produziu resultados significativamente melhores do que a heurística BPA, o que confirma a adequação desta estratégia para superar as desvantagens produzidas pela adoção de ambos os valores (0 e 1) no parâmetro $optInc$ na heurística BPA.

Em relação aos resultados da Figura 8.7 e da Figura 8.8, a aplicação da heurística *tunable allocation* produziu uma quantidade significativa de alocações classificadas como super provisão. Ainda que esta heurística tenha produzido uma quantidade reduzida de violações de SLA (o que é observável ao comparar as saídas das Figuras 8.4, 8.6 e 8.8), o ajuste nas classes de configuração realizados pela heurística também gerou um alto valor de custo de infraestrutura (quando comparado aos custos mostrados nas Figuras 8.3 e 8.5). Em resumo, o ajuste nas classes de configuração realizados pela heurística *tunable allocation* aproximou as classes de configuração estimadas pelas técnicas de aprendizado de máquina às classes de configuração adotadas pelas estratégias básicas baseadas em super provisão ($Over_{75}$ e $Over_{100}$). Desse modo, é possível concluir que a heurística *tunable allocation* não apresentou resultados eficazes na avaliação conduzida.

Por sua vez, considerando os resultados apresentados na Figura 8.9 e na Figura 8.10, é possível notar que a heurística *adaptive allocation* produziu resultados mais eficazes do que a heurística *tunable allocation*, uma vez que os custos de infraestrutura mostrados na Figura 8.9 são significativamente inferiores aos custos de infraestrutura reportados na Figura 8.7. Em particular, a utilização da heurística *adaptive allocation* combinada com a abordagem

$5NN(d.f. = E)$ produziu a menor quantidade de violações de SLA (quando comparada com a quantidade de violações produzidas pelas abordagens não básicas na Figura 8.4 e na Figura 8.6), e ao mesmo tempo produziu um custo de infraestrutura inferior aos custos reportados pelas abordagens básicas baseadas em super provisionamento.

Desse modo, é notável que a heurística *adaptive allocation* é mais eficaz do que a heurística *tunable allocation* em relação à capacidade de adaptar as classes de configuração estimadas pelas técnicas de aprendizado de máquina empregadas no Experimento II, especialmente quando as características associadas às tarefas de qualidade de dados sendo processadas pelo SMQD variam de maneira notória ao longo do tempo. Mais uma vez, considerando os resultados da Figura 8.9 e da Figura 8.10, a técnica SVM produziu custos e quantidade de violações de SLA bastante elevados, o que reafirma a inadequação desta técnica para o problema investigado (dada a pequena quantidade de características que quantificam uma tarefa de deduplicação de dados).

Em relação aos resultados mostrados na Figura 8.11 e na Figura 8.12, é simples notar que a heurística *sliced training data* se mostrou bastante efetiva para reduzir tanto os custos de infraestrutura do serviço quanto a quantidade de violações de SLA. Em especial, a combinação da heurística *sliced training data* com as abordagens $5NN(d.f. = E)$ e $RF(\#T = 5)$ produziram resultados mais eficazes do que todas as outras heurísticas e/ou algoritmos de aprendizado de máquina avaliados no Experimento II. Ainda que a heurística *sliced training data* empregue uma estratégia simples para filtrar as instâncias contidas na base de treinamento, esta estratégia produziu resultados bastante eficazes e, portanto, a combinação desta estratégia pode ser futuramente explorada juntamente com outras heurísticas por meio da proposição de metaheurísticas específicas para lidar com o problema de provisionamento de recursos computacionais no contexto de um SMQD.

Por fim, é importante destacar os resultados da abordagem $5NN(d.f. = E)$ combinada com a heurística *sliced training data*, a qual foi capaz de aproximar a quantidade de violações de SLA produzidas com a quantidade de violações geradas pelas abordagens básicas (baseadas em super provisionamento) e ao mesmo tempo produziu custos de infraestrutura (*VMC$*) significativamente menores.

Em resumo, um algoritmo de provisionamento apropriado para um SMQD (possivelmente combinado com estratégias baseadas em heurísticas) deve produzir um bom balance-

amento entre o custo de infraestrutura gerado para executar as tarefas do serviço e a quantidade de violações de SLA ocorridas neste processo. Assim, em um cenário prático, a escolha do algoritmo de provisionamento adotado deve levar em consideração tanto a criticidade das tarefas de qualidade de dados executadas (ou seja, tolerância à violações de SLA) quanto a necessidade dos clientes do serviço em reduzir os custos de infraestrutura e, consequentemente, os custos que serão pagos pelo serviço.

8.4 Experimento III

Nesta seção, é apresentada a investigação empírica conduzida no intuito de avaliar diferentes heurísticas no contexto do problema de DID. O principal objetivo deste experimento foi avaliar as heurísticas considerando as diferentes métricas propostas no Capítulo 6, assim como discutir as hipóteses definidas na Seção 8.1.

Foram utilizadas três bases de dados para avaliar os métodos de DID baseados em heurísticas: $Cora$[2], $DBLPM4$[3] e $Febrl10k$[4]. A primeira base de dados contém dados de publicações e foi anteriormente utilizada em diversos trabalhos disponíveis no estado da arte sobre deduplicação de dados [12]. Por sua vez, a base de dados $DBLPM4$ também contém dados de publicações científicas e foi gerada artificialmente pelos autores em [33] para avaliar diferentes técnicas de classificação coletiva de entidades duplicadas. Por fim, a base de dados $Febrl10k$ foi gerada artificialmente por meio da ferramenta Febrl[5] empregando os seguintes parâmetros: $max\_duplicate\_per\_record = 10$, $max\_modification\_per\_record = 3$, $max\_modification\_per\_field = 2$ e $\#duplicates \approx 5\%$ do tamanho da base de dados.

Estas bases de dados foram selecionadas para o Experimento III por duas razões principais: i) todas as bases de dados possuem os respectivos gabaritos das entidades duplicadas, o que é crucial para medir os resultados de eficácia produzidos pelos algoritmos de DID avaliados; e ii) as bases de dados $Cora$, $DBLPM4$ e $Febrl10k$ apresentam estatísticas diferentes (ver Tabela 8.8) e, portanto, é possível avaliar os resultados gerados pelos algoritmos de DID ao processarem grafos de similaridade consideravelmente distintos (principalmente

[2]http://secondstring.sourceforge.net/

[3]http://dblab.cs.toronto.edu/project/stringer/

[4]https://sites.google.com/site/dqgroupufcg/datasets/Febrl_10k.csv

[5]http://sourceforge.net/projects/febrl/

em relação ao grau médio dos vértices e a quantidade de arestas).

Para as bases de dados $Cora$ e $DBLPM4$, foram empregados os seguintes valores de parâmetros para a geração dos grafos de similaridade: $\theta = 0,8$ e a função de similaridade $JaroWinkler$ [12]. Por sua vez, para a base de dados $Febrl10k$, foi empregado o valor $\theta = 0,95$ e a função de similaridade $JaroWinkler$. Foram utilizados valores altos para o parâmetro θ (especialmente para a base de dados $Febrl10k$) para evitar que os métodos de DID produzam uma grande quantidade de classificações falso positivas e, desse modo, seja reduzida significativamente a eficácia reportada pelos métodos. Na Tabela 8.8, são apresentadas algumas estatísticas das bases de dados, as quais foram produzidas após a aplicação do algoritmo $Center$ sobre o grafo de similaridade correspondente a cada base de dados como um todo.

Tabela 8.8: Estatísticas das bases de dados utilizadas no Experimento III (empregando o algoritmo $Center$).

Estatística		**Cora**	**DBLPM4**	**Febrl10k**
Vértices	Quantidade	1.916	5.381	10.000
	Média #grau	30,03	17,4	0,13
	Max #degree	102	79	9
Arestas	Quantidade	28.778	46.838	658
	#IntraCluster	24.069	38.258	656
	#InterCluster	4.709	8.580	2
Agrupamentos	Quantidade	240	514	9.748
	Média #vértice	7,98	10,46	1,02
	Max #vértice	93	62	10
	Max #vizinhos	6	20	1
Blocos	Quantidade	-	-	11
	Média #Agrupamentos	-	-	76
	Max #Agrupamento	-	-	773
	Média #vértice	-	-	873,18
	Max #vértice	-	-	2.985

Uma vez que as bases de dados empregadas no Experimento III não são naturalmente

incrementais, foi empregado um processo (denominado *Sliced Increments*) proposto pelos autores em [30] para gerar incrementos de diferentes tamanhos sobre os dados das bases. Este processo é realizado da seguinte maneira: no primeiro incremento ($i = 1$) sobre a base de dados, é removida uma entidade de maneira aleatória; por sua vez, no i-*ésimo* incremento, as entidades removidas na $(i - 1)$-*ésima* iteração são inseridas novamente na base de dados e são removidas aleatoriamente $2^{(i-1)}$ entidades. Na última iteração, apenas as entidades removidas na iteração anterior são adicionadas novamente e nenhuma entidade é removida. No intuito de aumentar a expressividade dos resultados dos experimentos e tornar a diferença entre as saídas dos algoritmos mais expressiva, este processo é repetido r vezes. Desse modo, os resultados reportados no Experimento III são provenientes da execução do processo $Sliced\ Increments$ r vezes empregando a estratégia CONT [30], ou seja, o agrupamento utilizado como base para o processamento de um novo incremento nos dados é sempre o resultado do agrupamento gerado a partir do processamento do incremento nos dados da iteração anterior.

8.4.1 Design Experimental

Note que, a partir da variação de alguns parâmetros relacionados aos métodos de DID ($\dot{T}$, μ, $\mathcal{F}$), juntamente com a variação do próprio método de DID, é possível gerar uma grande quantidade de algoritmos de DID diferentes a partir da metaheurística CFG (Algoritmo 15). Desse modo, os parâmetros da metaheurística CFG foram variados no intuito de gerar métodos de DID significativamente diferentes. Os algoritmos de DID avaliados empregando as bases de dados $Cora$ e $DBLPM4$ são mostrados na Tabela 8.9.

Foi empregado o valor $\mu = \frac{2}{3}$ para as heurísticas baseadas em filtros de cobertura no processamento da base de dados $Cora$. Por sua vez, para a base de dados $DBLPM4$, foi utilizado o valor $\mu = \frac{1}{4}$ para as heurísticas baseadas em filtros de cobertura. Estes valores de μ foram adotados no experimento porque produziram melhores resultados tendo em vista os diferentes tamanhos de incrementos processados pelos métodos de DID ao processar bases de dados distintas (na prática, foram empregados valores de μ maiores para processar bases de dados menores, e vice-versa).

Por fim, para a base de dados $Febrl10k$, foi empregada uma quantidade menor de métodos de DID: {$Greedy(\bar{T}(\Delta G))$, $CENTER(\bar{T}(\Delta G))$, $M\text{-}CENTER(\bar{T}(\Delta G))$,

Tabela 8.9: Algoritmos de DID gerados a partir da metaheurística CFG (utilizados na avaliação das bases de dados $Cora$ e $DBLPM4$).

	Parâmetros da Metaheurística CFG			
Abreviatura	$\mathcal{C}l_{alg}$	$\mathcal{F}$	$\dot{T}$	$greedy\_flag$
$Greedy(\bar{T}(\Delta G))$	-	-	$\bar{T}$	true
Q-$Greedy(\mathcal{R}(\bar{T}(\Delta G)))$	-	$\mathcal{R}$-$Filter$	$\bar{T}$	false
Q-$Greedy(\mathcal{MN}(\bar{T}(\Delta G)))$	-	$\mathcal{MN}$-$Filter$	$\bar{T}$	false
Q-$Greedy(l\mathcal{C}(\bar{T}(\Delta G)))$	-	$l\mathcal{C}$-$Filter$	$\bar{T}$	false
Q-$Greedy(\mathcal{M}d(\bar{T}(\Delta G)))$	-	$\mathcal{M}d$-$Filter$	$\bar{T}$	false
Q-$Greedy(\mathcal{M}w(\bar{T}(\Delta G)))$	-	$\mathcal{M}w$-$Filter$	$\bar{T}$	false
Q-$Greedy(m\mathcal{C}(\bar{T}(\Delta G)))$	-	$m\mathcal{C}$-$Filter$	$\bar{T}$	false
Q-$Greedy(CENTER(\bar{T}(\Delta G)))$	CENTER	-	$\bar{T}$	true
Q-$Greedy(M\text{-}CENTER(\bar{T}(\Delta G)))$	M-CENTER	-	$\bar{T}$	true
Q-$Greedy(STAR(\bar{T}(\Delta G)))$	STAR	-	$\bar{T}$	true
$CENTER(\bar{T}(\Delta G))$	CENTER	-	$\bar{T}$	false
$M\text{-}CENTER(\bar{T}(\Delta G))$	M-CENTER	-	$\bar{T}$	false
$STAR(\bar{T}(\Delta G))$	STAR	-	$\bar{T}$	false

$STAR(\bar{T}(\Delta G))\}$. Esta escolha se deu pelo fato de que os métodos de DID baseados em filtros de cobertura serem mais efetivos quando processam grafos de similaridade densos e, por este motivo, esta categoria de métodos de DID não produziria resultados satisfatórios ao processar a base de dados $Febrl10k$ (ver estatísticas apresentadas na Tabela 8.8).

No intuito de avaliar os métodos de DID no contexto do processamento de incrementos sobre os dados de tamanho significativos, não foram empregadas técnicas de indexação na base de dados $Cora$ nem na base de dados $DBLPM4$, mas foi empregada a técnica de indexação blocagem padrão (Definição 2.2.2) sobre a base de dados $Febrl10k$. Os detalhes relacionados à blocagem aplicada na base de dados $Febrl10k$ e às regras de similaridade aplicadas em cada uma das bases de dados avaliadas no Experimento III são apresentados no Apêndice E.

Antes de aplicar o processo $Sliced\ Increments$ sobre as bases de dados, o agrupamento inicial das bases de dados foi realizado utilizando o algoritmo $Center$. Além disso, para

repetições, foi empregado o valor $r = 5$ para as bases de dados $DBLPM4$ e $Febrl10k$, e o valor $r = 10$ para a base de dados $Cora$ (foram empregados valores de r suficientemente grandes para tornarem as diferenças entre os tempos de execução dos algoritmos de DID evidentes em cada base de dados).

8.4.2 Implementação

A implementação dos métodos de DID foi realizada utilizando a linguagem de programação Java. Foram empregadas as APIs Jgraph[6] e SecondString[7] para representar os grafos de similaridade e executar funções de similaridade, respectivamente. O algoritmo $Greedy$ [30] foi implementado com base na instanciação baseada em *Correlation Clustering* (GreedyCorr) proposta em [30]. Por fim, os testes foram realizados utilizando um computador com 6GB de memória RAM e processador Core i-7.

Nos tempos de execução reportados, não foram computados o tempo para gerar e atualizar (ao longo do processamento dos incrementos nos dados) os grafos de similaridade nem o tempo necessário para calcular $\dot{T}(\Delta G)$, uma vez que a avaliação visou medir unicamente a eficiência dos métodos de DID.

8.4.3 Resultados

Na Tabela 8.10, são reportados os resultados de eficácia dos métodos de DID utilizando a base de dados $Febrl10k$. Por sua vez, os resultados utilizando a base de dados $Cora$ e $DBLPM4$ são apresentados na Tabela 8.11 e na Tabela 8.12, respectivamente. Algumas métricas são reportadas nos resultados considerando tanto o valor médio (A.) quanto o valor final (F.). Em relação às métricas de eficiência, o valor acumulado dos tempos de execução dos métodos de DID ao processar as bases de dados $Cora$, $DBLPM4$ e $Febrl10k$ são reportados nas Figuras 8.13, 8.14 e 8.15, respectivamente.

Os métodos de DID baseados na combinação de algoritmos de agrupamento automático simples (ou seja, $Center$, $MergeCenter$ e $Star$) com o algoritmo $Greedy$ produziram tempos de execução 5 a 10 vezes maiores do que os demais métodos de DID baseados em

[6] http://jgrapht.org/

[7] http://secondstring.sourceforge.net/

Tabela 8.10: Resultados de eficácia dos algoritmos de DID no Experimento III utilizando a base de dados $Febrl10k$.

Método de DID	A. F-m	F. Pr	F. Rc	A. wCPr
$Greedy(\bar{T}(\Delta G))$	0,652	0,860	0,623	0,662
$Center(\bar{T}(\Delta G))$	0,640	0,840	0,620	0,660
$M\text{-}Center(\bar{T}(\Delta G))$	0,703	0,833	0,753	0,674
$Star(\bar{T}(\Delta G))$	0,684	0,837	0,694	0,657

minimização de função de penalidade, sem que tenham apresentado resultados de eficácia significativamente melhores. Uma vez que neste trabalho são investigadas abordagens para otimizar a execução de métodos de DID, é possível concluir que esta combinação não é uma abordagem efetiva para este fim. Este fenômeno pode ser explicado pelo fato de que os algoritmos de agrupamento automático simples usualmente produzem agrupamentos com uma função de penalidade associada bastante elevada e, por este motivo, um algoritmo baseado em minimização de função de penalidade usualmente requer um tempo de execução elevado para tentar minimizar em seguida a penalidade produzida pelos agrupamentos (usualmente executando operações de $Merge$, $Split$ e $Move$ sobre os agrupamentos). Por este motivo, os resultados produzidos pelos métodos de DID baseados na combinação de algoritmos de agrupamento automático simples (ou seja, $Center$, $MergeCenter$ e $Star$) com o algoritmo $Greedy$ não foram reportados nem discutidos no Experimento III.

Tabela 8.11: Resultados de eficácia dos algoritmos de DID no Experimento III utilizando a base de dados $Cora$.

Método de DID	A. F-m	F. Pr	F. Rc	dgEfficiency	dgEfficacy	A. wCPr	A. PCPr	CC
$Greedy(\bar{T}(\Delta G))$	0,764	0,930	0,661	0,338	0,642	0,922	0,561	7.071
Q-$Greedy(\mathcal{R}(\bar{T}(\Delta G)))$	0,750	0,932	0,634	0,342	0,556	0,499	0,530	8.067
Q-$Greedy(\mathcal{MN}(\bar{T}(\Delta G)))$	0,741	0,916	0,614	0,330	0,561	0,623	0,542	8.818
Q-$Greedy(l\mathcal{C}(\bar{T}(\Delta G)))$	0,736	0,924	0,609	0,370	0,598	0,627	0,530	8.903
Q-$Greedy(\mathcal{M}d(\bar{T}(\Delta G)))$	0,723	0,958	0,602	0,352	0,563	0,600	0,522	9.165
Q-$Greedy(\mathcal{M}w(\bar{T}(\Delta G)))$	0,734	0,962	0,612	0,358	0,558	0,607	0,493	8.822
Q-$Greedy(m\mathcal{C}(\bar{T}(\Delta G)))$	0,744	0,913	0,599	0,344	0,512	0,590	0,520	9.143
$CENTER(\bar{T}(\Delta G))$	0,741	0,920	0,677	0,335	0,320	0,856	0,629	8.815
$M\text{-}CENTER(\bar{T}(\Delta G))$	0,848	0,836	0,892	0,351	0,418	0,666	0,592	36.672
$STAR(\bar{T}(\Delta G))$	0,790	0,866	0,766	0,333	0,458	0,805	0,707	1.3911

Tabela 8.12: Resultados de eficácia dos algoritmos de DID no Experimento III utilizando a base de dados $DBLPM4$.

Método de DID	A. F-m	F. Pr	F. Rc	dgEfficiency	dgEfficacy	A. wCPr	A. PCPr	CC
$Greedy(\bar{T}(\Delta G))$	0,894	0,923	0,902	0,332	0,923	0,864	0,850	12.538
Q-$Greedy(\mathcal{R}(\bar{T}(\Delta G)))$	0,752	0,915	0,536	0,343	0,784	0,703	0,743	23.335
Q-$Greedy(\mathcal{MN}(\bar{T}(\Delta G)))$	0,860	0,894	0,848	0,339	0,833	0,785	0,663	13.997
Q-$Greedy(l\mathcal{C}(\bar{T}(\Delta G)))$	0,854	0,891	0,834	0,331	0,753	0,776	0,673	14.408
Q-$Greedy(\mathcal{M}d(\bar{T}(\Delta G)))$	0,852	0,899	0,839	0,347	0,775	0,773	0,657	14.386
Q-$Greedy(\mathcal{M}w(\bar{T}(\Delta G)))$	0,846	0,892	0,834	0,340	0,765	0,769	0,656	14.559
Q-$Greedy(m\mathcal{C}(\bar{T}(\Delta G)))$	0,825	0,888	0,778	0,339	0,781	0,715	0,609	16.152
$CENTER(\bar{T}(\Delta G))$	0,873	0,905	0,858	0,308	0,866	0,786	0,850	19.088
$M\text{-}CENTER(\bar{T}(\Delta G))$	0,922	0,900	0,990	0,395	0,801	0,618	0,569	296.994
$STAR(\bar{T}(\Delta G))$	0,842	0,881	0,858	0,356	0,803	0,746	0,733	25.252

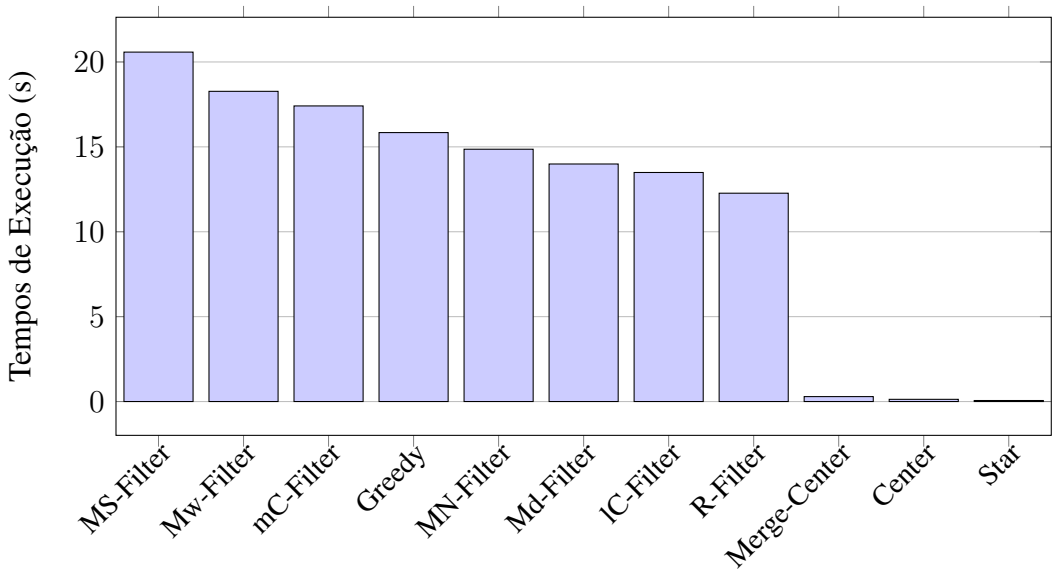

Figura 8.13: Resultados de eficiência utilizando a base de dados Cora.

8.4.4 Discussão

Resultados utilizando a base de dados Cora (*Eficácia*). Em relação aos resultados da Tabela 8.10, note que o algoritmo $Greedy$ produziu o menor valor de penalidade (CC). Este fato refletiu positivamente no valor de F-measure reportado por este algoritmo. Além disso, devido à efetividade do processo de minimização de função de penalidade realizado pelo algoritmo $Greedy$, este algoritmo tende a produzir agrupamentos bastante coesos, o que reflete de maneira significativa nos valores de precisão reportados. No entanto, este processo usualmente favorece melhores resultados de precisão do que resultados de cobertura. Ainda assim, o algoritmo $Greedy$ produziu o terceiro melhor valor de F-measure entre os métodos de DID avaliados.

Como esperado, em relação às abordagens baseadas em filtros de cobertura, é fácil notar que quando uma heurística processa apenas uma porção dos agrupamentos em $\mathcal{L}_{\bar{T}(\Delta G)}$ (ou seja, $\mathcal{L}_{\bar{T}(\Delta G)} \backslash \mathcal{L}^{\mathcal{F}}_{\bar{T}(\Delta G)}$), a mesma tende a produzir resultados de F-measure ligeiramente inferiores (em comparação aos resultados produzidos pelo algoritmo $Greedy$ processando todos os agrupamentos em $\mathcal{L}_{\bar{T}(\Delta G)}$), assim como produz valores de penalidade associados aos agrupamentos ligeiramente maiores (também comparados aos resultados do algoritmo $Greedy$). Além disso, uma vez que as heurísticas de filtros de cobertura são baseadas em

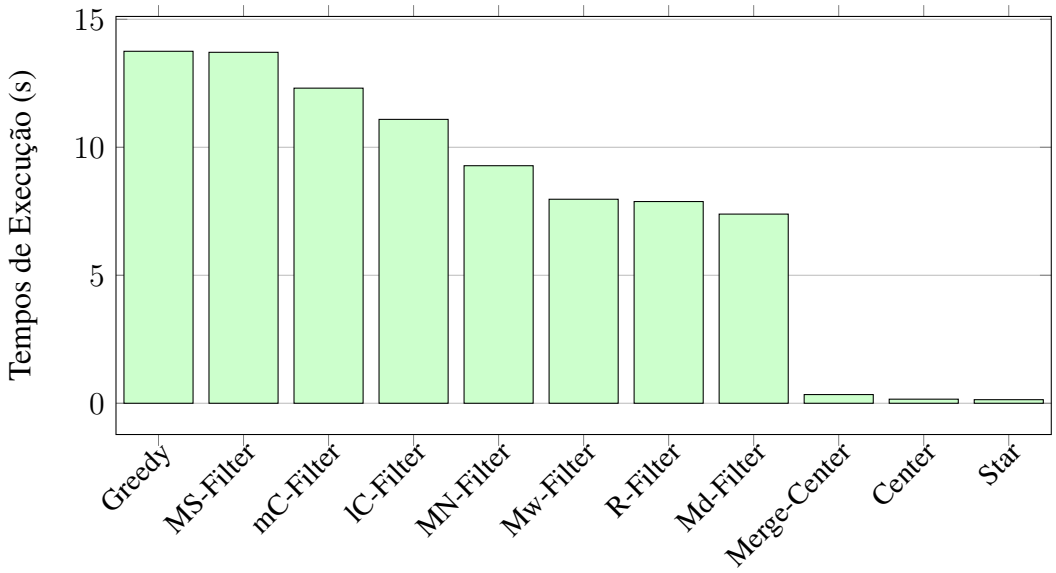

Figura 8.14: Resultados de eficiência utilizando a base de dados DBLPM4.

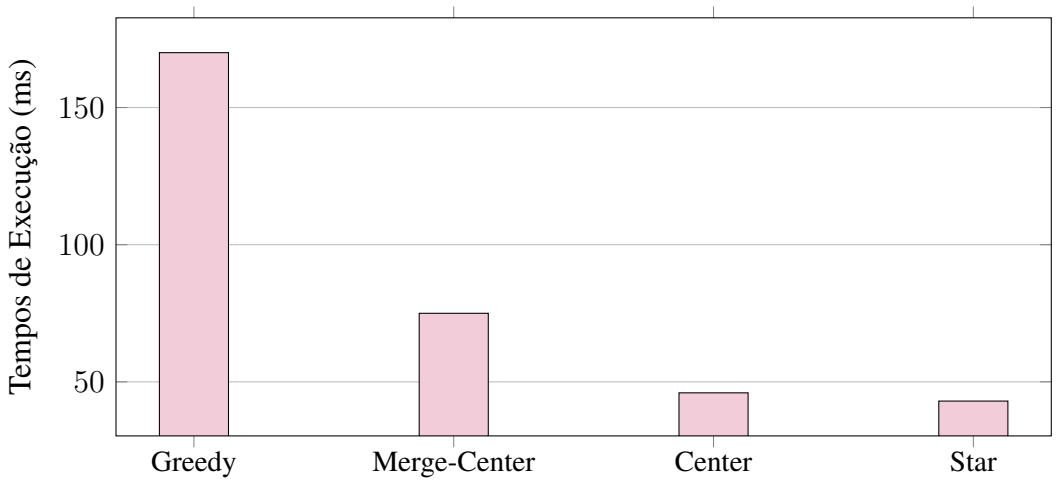

Figura 8.15: Resultados de eficiência utilizando a base de dados Febrl10k.

minimização de função de penalidade, as mesmas também favorecem a produção de agrupamentos coesos e, assim, os agrupamentos produzidos apresentaram valores de precisão elevados.

É também possível notar que os filtros de cobertura produziram pouco impacto sobre a F-measure reportada (tanto em relação ao valor médio, quando no valor final), o que confirma a intuição utilizada pelos filtros de cobertura de que muitos agrupamentos em $\bar{T}(\Delta G)$ não podem ser significativamente melhorados por um algoritmo de DID, seja porque não é mais possível otimizar o valor de uma função de penalidade associada a estes agrupamentos ou porque o processamento destes agrupamentos produz pouco impacto sobre o valor de F-measure reportado. Em geral, os filtros de cobertura $\mathcal{R}$, $m\mathcal{C}$, $\mathcal{MN}$ produziram resultados de F-measure ligeiramente melhores do que as demais heurísticas também baseadas em filtros.

Em relação aos algoritmos de agrupamento automático avaliados utilizando a base de dados Cora (Tabela 8.10), a avaliação dos mesmos produziu resultados significativamente diferentes. Similarmente ao que ocorreu ao serem empregadas abordagens baseadas em minimização de função de penalidade, o algoritmo $Center$ produziu resultados que favoreceram a precisão sobre a cobertura dos agrupamentos. Este fenômeno ocorreu porque este algoritmo tende a gerar agrupamentos bastante coesos, mas usualmente pequenos. Esta característica foi observada tanto ao longo do processamento dos incrementos nos dados quanto nos resultados de agrupamento finais.

Por sua vez, os algoritmos $Merge\text{-}Center$ e $Star$ produziram resultados de eficácia bastante satisfatórios, tanto em relação ao valor médio quanto ao valor final de F-measure. Note que, os resultados de F-measure reportados por estes dois algoritmos superaram os resultados de F-measure produzidos pelo algoritmo $Greedy$. No entanto, é importante destacar que estes valores de F-measure elevados (produzidos pelos algoritmos $Merge\text{-}Center$ e $Star$) foram reportados ainda que a penalidade associada aos agrupamentos gerados por estes algoritmos tenha sido significativamente maior do que a penalidade associada aos agrupamentos produzidos pelas abordagens baseadas em minimização de função de penalidade.

Este fenômeno pode ser explicado pelo fato de os algoritmos $Star$ e $Merge\text{-}Center$ produzirem uma quantidade significativa de agrupamentos com uma grande quantidade de vértices e, neste processo, os algoritmos não levam em consideração a avaliação de uma função de penalidade. Desse modo, os agrupamentos produzidos usualmente contém muitos

pares de vértices (v_1, v_2) (duplicados ou não), nos quais $e(v_1, v_2) = 0$ (porque $e(v_1, v_2) < \theta$). Assim, uma vez que estes pares são frequentes nos agrupamentos gerados, a penalidade associada a estes algoritmos é normalmente alta. Outra consequência desse fenômeno é o fato de os agrupamentos gerados pelo algoritmo $Merge\text{-}Center$ usualmente apresentarem melhor valor de cobertura do que de precisão. Este fato pode ser observado ao serem comparados os valores das métricas $F.Pr$, $F.Rc$, $A.wCPr$ e $A.PCPr$ produzidas pelos algoritmos $Star$, $Center$, $Merge\text{-}Center$ e $Greedy$ (note que os algoritmos $Greedy$, $Center$ e $Star$ produziram menor cobertura, mas usualmente geram valores de métricas relacionadas à precisão maiores do que os valores produzidos pelo algoritmo $Merge\text{-}Center$).

Em relação à métrica *grau de estabilidade de eficácia* (ainda na Tabela 8.10), note que o algoritmo $Greedy$ gerou o maior valor de $dgEfficacy$ dentre os algoritmos avaliados, o que confirma a adequação deste algoritmo para manter valores de F-measure satisfatórios ao longo do tempo. Por sua vez, as abordagens baseadas em filtros de cobertura produziram resultados de grau de estabilidade de eficácia ligeiramente inferiores ao valor reportado pelo algoritmo $Greedy$ e ligeiramente superiores aos valores reportados pelos algoritmos $Center$, $Merge\text{-}Center$ e $Star$.

Resultados utilizando a base de dados Cora (*Eficiência*). Em relação aos resultados apresentados na Figura 8.13, é possível notar que dois dos filtros de cobertura avaliados ($m\mathcal{C}$ e $\mathcal{M}w$) reportaram tempos de execução superiores ao tempo de execução do algoritmo $Greedy$, ou seja, estes filtros de cobertura não produziram resultados que satisfazem a restrição especificada na última linha da Tabela 6.2. O resultado reflete a complexidade intrínseca das operações realizadas por estes dois filtros para selecionar os agrupamentos filtrados. Por sua vez, quatro outros filtros ($\mathcal{M}d$, $l\mathcal{C}$, $\mathcal{R}$ e $\mathcal{MN}$) executaram mais rápido do que o algoritmo $Greedy$. Similarmente, este resultado é explicado pela complexidade inferior associada às operações realizadas por estes quatro filtros (comparada à complexidade das operações realizadas por filtros mais sofisticados). Desse modo, os filtros $\mathcal{MN}$, $\mathcal{M}d$, $l\mathcal{C}$ e $\mathcal{R}$ executaram 12%, 15%, 22% e 32% mais rápido do que o algoritmo $Greedy$, respectivamente. Estes diferentes resultados de eficiência devem ser avaliados conjuntamente aos resultados de eficácia produzidos por estes filtros no intuito de selecionar um bom balanceamento entre estes dois aspectos reportados pelas heurísticas.

Por sua vez (ainda na Figura 8.13), os três algoritmos de agrupamento automático ava-

liados ($Center$, $Merge\text{-}Center$ e $Star$) executaram 97%, 98% e 99% mais rápido do que a heurística baseada em minimização de função de penalidade mais eficiente ($\mathcal{R}\text{-}Filter$) avaliada no Experimento III. Este resultado é esperado devido à baixa complexidade das operações realizadas por estes algoritmos para gerar os agrupamentos. Além disso, é também importante destacar que os três algoritmos de agrupamento automático avaliados no Experimento III são muito menos afetados pela complexidade (quantidade de arestas) do grafo de similaridade processado do que as abordagens baseadas em minimização de função de penalidade. Por fim, ainda que os tempos de execução dos algoritmos de agrupamento automático sejam enormemente inferiores aos tempos de execução das abordagens baseadas em minimização de função de penalidade, é importante ressaltar que os primeiros usualmente produzem agrupamentos menos coesos, o que afeta negativamente os resultados das métricas relacionadas à precisão dos agrupamentos. Portanto, os algoritmos $Center$, $Merge\text{-}Center$ e $Star$ apresentam uma clara opção de abordagem de DID que sacrifica resultados de precisão dos agrupamentos para executar o processo de maneira muito mais eficiente.

Finalmente, em relação à métrica $dgEfficiency$ (Tabela 8.10), todos os métodos de DID avaliados produziram valores entre 0,3 e 0,4. Desse modo, é possível predizer que todos os algoritmos avaliados no Experimento III irão aumentar os respectivos tempos de execução, ao processarem componentes de cobertura maiores, de maneira aproximadamente semelhante em relação aos respectivos tempos de execução reportados ao processarem componentes de cobertura menores.

Resultados utilizando a base de dados DBLPM4 (*Eficácia*). Novamente (ver Tabela 8.11), o algoritmo $Greedy$ produziu o menor valor de penalidade associado aos agrupamentos gerados, seguido pelas abordagens baseadas em filtros de cobertura e pelos algoritmos de agrupamento automático. Este fato refletiu positivamente sobre os valores das métricas de precisão produzidos pelo algoritmo $Greedy$. Por exemplo, note que o algoritmo $Greedy$ produziu o maior valor de $A.wCPr$. No entanto, diferente dos resultados de eficácia apresentados utilizando a base de dados $Cora$, na base de dados DBLPM4, o algoritmo $Greedy$ produziu resultados de cobertura bastante elevados, o que contribuiu para gerar um dos maiores valores de F-measure dentre os valores reportados pelos algoritmos avaliados. Com exceção do resultado de F-measure produzido pelo filtro $\mathcal{R}\text{-}Filter$, todas as outras abordagens baseadas em filtros de cobertura produziram resultados de F-measure apenas ligeiramente

diferentes dos valores de F-measure reportados pelo algoritmo $Greedy$. Em particular, as heurísticas $\mathcal{MN}$, $\mathcal{M}d$ e $l\mathcal{C}$ produziram os maiores valores de F-measure (dentre os valores de F-measure produzidos pelos filtros de cobertura).

Em relação aos algoritmos de agrupamento automático avaliados (ver Tabela 8.11), o algoritmo $Merge\text{-}Center$ produziu o maior valor de A. F-measure, seguido pelos algoritmos $Center$ e $Star$. Em particular, o algoritmo $Merge\text{-}Center$ produziu altos valores de precisão e cobertura, superando todos os outros métodos de DID avaliados. No entanto, é importante notar a relação entre os altos valores de cobertura produzidos pelo algoritmo $Merge\text{-}Center$ (como consequência dos agrupamentos produzidos com uma grande quantidade de vértices) e os valores reportados por este algoritmo nas métricas $wCPr$ e $PCPr$ (ou seja, o algoritmo $Merge\text{-}Center$ acaba produzindo melhores valores de cobertura do que valores de precisão). Por fim, o algoritmo $Greedy$ produziu o maior valor de $dgEfficacy$, o que significa que este algoritmo produziu uma menor perda de precisão e cobertura ao longo do tempo. Em relação aos algoritmos de agrupamento automático, o algoritmo $Star$ reportou um valor de $dgEfficacy$ bastante significativo, possivelmente porque este algoritmo produziu agrupamentos notoriamente menores do que os agrupamentos produzidos pelo algoritmo $Merge\text{-}Center$ e, assim, o algoritmo $Star$ gerou uma perda de precisão e cobertura inferior ao longo do tempo. Por sua vez, dentre as heurísticas baseadas em filtros de cobertura, o filtro $\mathcal{MN}$ produziu o maior valor de $dgEfficacy$, seguido dos demais filtros de cobertura.

Resultados utilizando a base de dados DBLPM4 (*Eficiência*). Primeiramente, é importante mencionar que, como a base de dados $DBLPM4$ contém mais vértices do que a base de dados $Cora$, foram empregadas mais repetições do processo $Sliced\ Increments$ sobre a base de dados $Cora$ do que sobre a base de dados $DBLPM4$ e, portanto, os tempos de execução dos métodos de DID reportados na base de dados $DBLPM4$ foram ligeiramente inferiores aos respectivos tempos de execução na base de dados $Cora$. Segundo, os tempos de execução das abordagens baseadas em filtros de cobertura também foram ligeiramente diferentes dos tempos reportados na base de dados $Cora$. Na base de dados $DBLPM4$, todos os filtros avaliados executaram mais rápido do que o algoritmo $Greedy$. Além disso, mais uma vez os algoritmos de agrupamento automático executaram significativamente mais rápidos (em torno de 97%) do que os outros métodos de DID avaliados. Por fim, todos os algoritmos apresentaram valores similares de $dgEfficiency$ (entre 0,3 e 0,4), o que reafirma

os resultados e discussões relacionadas a esta métrica para a base de dados $Cora$.

Resultados utilizando a base de dados Febrl10k (*Eficácia*). Primeiramente, note que as estatísticas do grafo de similaridade da base de dados $Febrl10k$ são consideravelmente diferentes (ver Tabela 8.8) das estatísticas dos grafos de similaridade das outras duas bases de dados. O grafo de similaridade da base de dados $Febrl10k$ é muito mais esparso do que os grafos das outras duas bases de dados e, por este motivo, agrupar os vértices do grafo de similaridade da base de dados $Febrl10k$ representa uma tarefa de deduplicação de dados mais simples. Dois dos algoritmos de agrupamento automático ($Merge\text{-}Center$ e $Star$) avaliados produziram resultados de F-measure que superaram os resultados reportados pelo algoritmo $Greedy$ e, neste caso particular, os algoritmos $Star$, $Merge\text{-}Center$ e $Greedy$ também produziram valores de precisão aproximadamente iguais. Estes resultados refletem duas características principais deste tratamento: i) a aplicação de um valor de θ bastante alto (o que produziu um grafo de similaridade esparso); e ii) os graus pequenos associados aos vértices do grafo de similaridade processado.

É também possível notar que os algoritmos $Greedy$, $Star$ e $Merge\text{-}Center$ melhoraram a eficácia reportada ao longo do processamento dos incrementos nos dados, uma vez que os valores de precisão e cobertura finais são significativamente superiores aos respectivos valores médios destas métricas. Mais uma vez, o algoritmo $Greedy$ produziu a menor penalidade (50,74), seguida pelas penalidades produzidas pelos algoritmos $Star$ (59,50), $Center$ (58,90) e $Merge\text{-}Center$ (77,63). Por fim, todos os algoritmos avaliados reportaram valores de $wCPr$ aproximadamente iguais.

Resultados utilizando a base de dados Febrl10k (*Eficiência*). Em relação aos resultados apresentados na Figura 8.15, todos os algoritmos de agrupamento automático executaram significativamente mais rápido (em torno de 56%-75%) do que a abordagem $Greedy$, mesmo processando um grafo de similaridade esparso. Ao processar a base de dados Febrl10k, os algoritmos $Star$ e $Merge\text{-}Center$ também produziram resultados de eficácia superiores (em relação ao algoritmo $Greedy$). Ambos os resultados (eficácia e eficiência) indicam que, para o processamento de grafos de similaridade esparsos, técnicas de agrupamento automático simples são mais recomendadas para resolver problemas de DID do que algoritmos mais complexos baseados em minimização de função de penalidade.

8.5 Experimento IV

Nesta seção, é apresentada a avaliação de diferentes estratégias para o controle de tamanho de blocos produzidos pela aplicação de múltiplas funções de chave de bloco no contexto de deduplicação de dados. As diferentes estratégias são geradas a partir da combinação das heurísticas propostas no Capítulo 7 utilizando a metaheurística $MkPCBS$. O objetivo geral da avaliação é avaliar a eficácia (resultados de qualidade) e eficiência das heurísticas propostas no processo de controle de tamanho de blocos, assim como a influência de valores de parâmetros como o esquema de peso de coocorrência e o valor de λ (o qual define o alcance das heurísticas) sobre os resultados das heurísticas.

8.5.1 Bases de Dados

Foram selecionadas 4 bases de dados para avaliar as heurísticas propostas: $CoraATDV$[8], $DBLPM4$[9], $DS3$[10] e $FebrlData$[11]. A primeira base de dados consiste de dados de pulbicações e foi empregada na avaliação de diversos outros trabalhos na área de deduplicação de dados. Por sua vez, a base de dados $DBLPM4$ é composta por títulos de publicações e foi gerada pelos autores em [33] para avaliar técnicas de classificação coletiva. A base de dados $DS3$ contém dados de publicações extraídas dos repositórios DBLP e Google Scholar. Por fim, a base de dados $FebrlData$ contém dados artificiais de indivíduos e foi gerada a partir da ferramenta Febrl[12] empregando os seguintes parâmetros: $max\_duplicate\_per\_record = 10$ $max\_modification\_per\_record = 3$, $max\_modification\_per\_field = 2$ e $\#duplicates \approx 5\%$ do tamanho da base de dados.

As bases de dados $Cora$, $DBLPM4$, $DS3$ e $FebrlData$ contém aproximadamente $2 * 10^3$, $4 * 10^3$, $8 * 10^3$, e $7 * 10^4$ entidades, respectivamente. Estas bases de dados foram selecionadas principalmente pelo fato de apresentarem os gabaritos (*ground truth*) das entidades duplicadas, o que permite uma avaliação dos resultados de qualidade produzidos pelas heurísticas propostas.

[8] https://github.com/TeamCohen/secondstring/tree/master/data

[9] http://dblab.cs.toronto.edu/project/stringer/

[10] extracted from [44]

[11] https://sites.google.com/site/datasetfebrl10k/febrl10k/febrl_10k.csv

[12] http://sourceforge.net/projects/febrl/

Tabela 8.13: Lista de algoritmos gerados a partir da metaheurística $MkPCBS$.

	Passos da Metaheurística MkPCBS			
Denominação do Algoritmo	**Shrink**	**Split**	**Merge**	**Exclude**
lECP	lECP	CooSlicer	MaxIntersectionMerge	-
lBCP	lBCP	CooSlicer	MaxIntersectionMerge	-
LBSP	LBSP	CooSlicer	MaxIntersectionMerge	-
lBCE	-	CooSlicer	MaxIntersectionMerge	lBCE
lECP+lBCE	lECP	CooSlicer	MaxIntersectionMerge	lBCE
Default	-	CooSlicer	MaxIntersectionMerge	-

8.5.2 Design Experimental

Note que ao variar os valores dos parâmetros empregados pela metaheurística $MkPCBS$ (por exemplo, λ, WS, S_{min}, S_{max}, passo $Prune$, passo $Merge$, passo $Split$ e passo $Exclude$) é possível gerar uma grande quantidade de algoritmos diferentes para controlar o tamanho de blocos no contexto de deduplicação de dados. No entanto, os valores destes parâmetros foram limitados visando produzir uma quantidade razoável de algoritmos levando em consideração diferentes abordagens. Com este objetivo, são exploradas 6 combinações de heurísticas para tratar o problema investigado. Estas combinações são apresentadas na Tabela 8.13, na qual são apresentadas: a denominação do algoritmo gerado a partir da combinação de outras heurísticas; e ii) as heurísticas que são empregadas para executar os 4 passos definidos na metaheurística $MkPCBS$.

Uma vez que os objetivos do Experimento IV também englobam a investigação da influência de valores de parâmetros como o esquema de pontuação de coocorrência (WS) e o valor de λ sob os resultados, os valores destes parâmetros são também variados no design experimental. Para cada base de dados avaliada, os algoritmos apresentados na Tabela 8.13 são combinados com três valores de λ ($\{0.2, 0.3, 0.4\}$)[13] e dois esquemas de pontuação de coocorrência ($ECBS$ e $ARCS$). Além disso, para cada base de dados, são empregados dois intervalos de tamanho diferentes (i.e., S_{min} e S_{max}).

[13] valores de λ maiores do que 0.4 produziram baixos resultados de qualidade no experimentos conduzidos, e, portanto, não são reportados neste trabalho

Desse modo, para cada base de dados, são avaliados um total de: i) 5 algoritmos combinados com 3 valores de λ, 2 esquemas de pontuação de coocorrência e 2 intervalos de tamanho; assim como ii) 1 algoritmo[14] combinado com 2 esquemas de pontuação de coocorrência e 2 intervalos de tamanho. Portanto, são exploradas $(4 * 5 * 3 * 2 * 2)$ + $(4 * 1 * 2 * 2)$ = (240) + (16) = 256 combinações de valores de parâmetros no design experimental. Por fim, os conjuntos de funções de chave de bloco empregadas para indexar as bases de dados são apresentados no Apêndice E.

Como abordagem básica (*baseline*), foi empregado o algoritmo *Sorted Neighborhood* (SN) [12] utilizando uma janela fixa de tamanho igual a w. Este algoritmo foi selecionado pois, com base no valor de w empregado, é possível configurar de antemão a quantidade de comparações entre entidades a ser produzida. Desse modo, é possível limitar a quantidade de comparações gerada pelo algoritmo SN levando em consideração a quantidade máxima de comparações que pode ser produzida por uma coleção de blocagens $\mathcal{B}'$, a qual contém blocos com tamanho entre S_{min} e S_{max}. Assim, para cada base de dados avaliada, o algoritmo SN foi executado $|\mathcal{F}|$ vezes, tal que cada execução utilizou uma função de chave de bloco $f \in \mathcal{F}$ distinta. Além disso, o algoritmo SN foi executado empregando duas variações do valor de janela máximo (w_{max}), como detalhado no Apêndice F.

Métricas

Para cada execução de algoritmo, são medidos tanto a eficácia quanto a eficiência dos algoritmos avaliados. A eficiência dos algoritmos é medida pelo tempo necessário para executar os 4 passos da metaheurística $MkPCBS$. Os resultados de eficiência reportados representam a média de 3 execuções de cada algoritmo. Por sua vez, a eficácia dos algoritmos é medida utilizando as métricas definidas na Seção 7.3.1, i.e., *Pair Completeness* (PC), *Reduction Ratio* (RR) e *Pair Quality* (PQ).

Para cada algoritmo na Tabela 8.13, são reportados os tempos de execução da combinação de valores de parâmetros que produziu o maior valor da média harmônica (F-measure) entre as métricas PC e RR (para cada base de dados e intervalo de tamanho avaliados). Mais formalmente, sejam $D_{set} = \{CoraATDV, DBLPM4, DS3, FebrlData\}$,

[14] A combinação de heurística denominada "Default"(na Tabela 8.13) é a única combinação da tabela que não é influenciada pelo valor do parâmetro λ

$WS_{set} = \{ARCS, ECBS\}$ e $\lambda_{set} = \{0.2, 0.3, 0.4\}$. Para cada algoritmo alg na Tabela 8.13, é reportado o tempo de execução de cada algoritmo que produziu o seguinte resultado[15] de qualidade (para cada base de dados e intervalo de tamanho avaliados):

$$\underset{D \in D_{set}, WS \in WS_{set}, \lambda \in \lambda_{set}}{arg\,max} 2.\frac{PC(alg(D,WS,\lambda)) * RR(alg(D,WS,\lambda))}{PC(alg(D,WS,\lambda)) + RR(alg(D,WS,\lambda))}$$

No intuito de calcular o tempo de indexação produzido pelo algoritmo SN, foram medidos apenas o tempo de indexação necessário para ordenar as entidades (para cada chave de bloco empregada) e par produzir os pares de entidades a serem comparados (ao realizar a varredura da janela sobre as entidades da base de dados). Os resultados de eficiência do algoritmo SN são reportados como a média de 3 execuções para cada combinação de parâmetros empregada. Para cada base de dados, é reportado o tempo de execução do algoritmo SN utilizando o tamanho de janela que produziu o maior valor referente à média harmônica entre PC e RR. Além disso, note que não foi avaliada a influência do valor do parâmetro λ sobre os resultados de eficiência dos algoritmos de poda de blocos, uma vez que não foram reportados os tempos de execuções dos algoritmos para todas as combinações de valores de parâmetros explorados.

Implementação

As heurísticas propostas no Capítulo 7 e o algoritmo SN foram implementados em Java e as execuções dos experimentos utilizaram uma máquina com CPU Core i-7 e 6GB de memória RAM. Não foram contabilizados os tempos de leitura e indexação (a geração da coleção de blocagens a partir da utilização de um conjunto de funções de chave de bloco) das bases de dados nem o tempo necessário para calcular as métricas de qualidade (PQ, PC e RR), uma vez que o objetivo da avaliação (em relação ao aspecto de eficiência) consiste em medir apenas o tempo de execução das heurísticas propostas.

8.5.3 Resultados

Os resultados de qualidade produzidos pelos algoritmos avaliados utilizando as bases de dados $CoraATDV$, $DBLPM4$, $DS3$ e $FebrlData$ são mostrados nas Tabelas 8.14, 8.15, 8.16 e 8.17, respectivamente. Por sua vez, os tempos de execução dos algoritmos avaliados

[15] estes resultados são destacados em negrito nas Tabelas 8.14-8.17

utilizando a base de dados $CoraATDV$ são apresentados nas Figuras 8.16 e 8.17. Prosseguindo, os resultados de eficiência empregando a base de dados $DBLPM4$ são mostrados nas Figuras 8.18 e 8.19. Os tempos de execução dos algoritmos ao processar a base de dados $DS3$ são apresentados nas Figuras 8.20 e 8.21. Por fim, os tempos de execução dos algoritmos empregando a base de dados $FebrlData$ são mostrados nas Figuras 8.22 e 8.23.

8.5.4 Discussão

Primeiramente, é importante notar que para a grande maioria dos resultados obtidos: i) quanto maior é o valor de λ empregado, menor tende a ser o resultado de PC e maior tende a ser o resultado de RR; e ii) quanto maiores os limites do intervalo de tamanho empregado, maior tende a ser o valor de PC e menor tende a ser o valor de RR. Estes dois padrões nos resultados são importantes para interpretar os resultados obtidos e destacar o fato dos valores dos parâmetros λ e (S_{min},S_{max}) produzirem grande influência sobre os resultados reportados.

CoraATDV (*Resultados de Qualidade*).

Em relação à Tabela 8.14, note que o esquema de pontuação de coocorrência ARCS produziu resultados de qualidade melhores do que o esquema ECBS. É também importante notar que o valor de λ empregado produziu diferentes níveis de influência sobre os resultados, dependendo das heurísticas empregadas. Para o algoritmo $lECP$, valores de λ maiores produziram maiores valores de RR sem prejudicar os resultados de PC, o que significa que o algoritmo foi capaz de remover entidades dos blocos sem impedir a comparação de entidades duplicadas nos blocos. Este resultado é ainda mais evidente quando o esquema ARCS é empregado.

Por sua vez, o valor de λ produziu muito mais influência sobre os resultados de qualidade do algoritmo $lBCP$. É fácil notar que este algoritmo reportou resultados de PC significativamente diferentes considerando os valores de λ empregados. Por esta razão, o algoritmo $lBCP$ reportou resultados de PC significativamente menores e resultados de RR significativamente maiores (ambos em relação aos resultados do algoritmo $lECP$). Similarmente ao algoritmo $lBCP$, os resultados reportados pelo algoritmo $LBSP$ foram também significativamente influenciados pelos valores de λ empregados (note as diferenças significativas entre os valores de PC reportados por este algoritmo). Os resultados de qualidade reporta-

Tabela 8.14: Resultados de Qualidade utilizando a base de dados Cora.

Algoritmo	Cora								
	$\{S_{min}, S_{max}\}$		$\{50, 150\}$			$\{100, 200\}$		
	λ		0,2	0,3	0,4	0,2	0,3	0,4
lECP	WS=ARCS	PC	0,94	0,94	**0,95**	0,98	0,97	**0,98**
		RR	0,80	0,82	**0,85**	0,75	0,78	**0,80**
	WS=ECBS	PC	0,94	0,93	0,92	0,97	0,96	0,96
		RR	0,89	0,82	0,85	0,75	0,78	0,80
lBCP	WS=ARCS	PC	**0,92**	0,85	0,80	0,95	**0,90**	0,84
		RR	**0,80**	0,83	0,86	0,74	**0,79**	0,82
	WS=ECBS	PC	0,88	0,82	0,76	0,92	0,87	0,80
		RR	0,80	0,83	0,86	0,74	0,79	0,82
LBSP	WS=ARCS	PC	**0,92**	0,86	0,83	0,96	0,91	**0,87**
		RR	**0,79**	0,82	0,85	0,74	0,78	**0,82**
	WS=ECBS	PC	0,89	0,84	0,80	0,93	0,89	0,83
		RR	0,79	0,82	0,84	0,74	0,78	0,82
lBCE	WS=ARCS	PC	0,92	0,91	0,89	0,95	0,94	**0,93**
		RR	0,77	0,81	0,84	0,73	0,77	**0,80**
	WS=ECBS	PC	0,93	0,91	**0,90**	0,96	0,93	0,90
		RR	0,80	0,83	**0,85**	0,74	0,78	0,81
LECP+lBCE	WS=ARCS	PC	0,91	0,87	**0,87**	**0,95**	0,89	0,85
		RR	0,84	0,88	**0,91**	**0,81**	0,85	0,89
	WS=ECBS	PC	0,92	0,87	0,83	0,93	0,86	0,83
		RR	0,85	0,87	0,91	0,81	0,85	0,89
Default	WS=ARCS	PC	**0,96**			**0,99**		
		RR	**0,72**			**0,65**		
	WS=ECBS	PC	0,93			0,98		
		RR	0,72			0,65		
SN (*baseline*)	$w = 2^{-1}\sqrt{w_{max}}$	PC	**0,74**					
		RR	**0,83**					
	$w = 2^{-2}\sqrt{w_{max}}$	PC	0,51					
		RR	0,92					

Tabela 8.15: Resultados de qualidade utilizando a base de dados DBLPM4,

Algoritmo	DBLPM4								
	$\{S_{min}, S_{max}\}$			$\{20, 60\}$			$\{60, 100\}$		
	λ			0,2	0,3	0,4	0,2	0,3	0,4
lECP	WS=ARCS		PC	**0,84**	0,83	0,76	**0,86**	0,86	0,79
			RR	**0,87**	0,87	0,92	**0,78**	0,78	0,85
	WS=ECBS		PC	0,73	0,72	0,66	0,78	0,78	0,72
			RR	0,87	0,87	0,92	0,78	0,78	0,85
lBCP	WS=ARCS		PC	**0,82**	0,77	0,72	**0,84**	0,8	0,74
			RR	**0,83**	0,86	0,88	**0,72**	0,76	0,8
	WS=ECBS		PC	0,71	0,67	0,61	0,77	0,73	0,66
			RR	0,83	0,86	0,88	0,72	0,76	0,80
LBSP	WS=ARCS		PC	**0,84**	0,80	0,77	0,86	**0,83**	0,79
			RR	**0,83**	0,85	0,88	0,71	**0,75**	0,79
	WS=ECBS		PC	0,75	0,72	0,69	0,8	0,77	0,74
			RR	0,83	0,85	0,88	0,72	0,75	0,79
lBCE	WS=ARCS		PC	**0,88**	0,87	0,84	0,9	**0,88**	0,83
			RR	**0,82**	0,84	0,86	0,7	**0,74**	0,78
	WS=ECBS		PC	0,77	0,74	0,68	0,81	0,78	0,71
			RR	0,82	0,84	0,87	0,70	0,74	0,77
LECP+lBCE	WS=ARCS		PC	**0,78**	0,75	0,59	**0,81**	0,76	0,6
			RR	**0,90**	0,91	0,95	**0,82**	0,94	0,91
	WS=ECBS		PC	0,69	0,64	0,53	0,74	0,69	0,58
			RR	0,9	0,91	0,95	0,82	0,85	0,91
Default	WS=ARCS		PC	**0,91**			**0,92**		
			RR	**0,78**			**0,63**		
	WS=ECBS		PC	0,80			0,85		
			RR	0,78			0,63		
SN (*baseline*)	$w = 2^{-1}\sqrt{w_{max}}$		PC	0,87					
			RR	0,47					
	$w = 2^{-2}\sqrt{w_{max}}$		PC	**0,86**					
			RR	**0,75**					

Tabela 8.16: Resultados de qualidade utilizando a base de dados DS3,

Algoritmo	DS3								
	$\{S_{min}, S_{max}\}$		$\{100, 200\}$			$\{200, 300\}$		
	λ		0,2	0,3	0,4	0,2	0,3	0,4
lECP	WS=ARCS	PC	0,86	0,80	0,73	0,87	0,83	0,76
		RR	0,84	0,86	0,89	0,76	0,79	0,82
	WS=ECBS	PC	**0,86**	0,8	0,73	**0,88**	0,83	0,76
		RR	**0,84**	0,86	0,89	**0,75**	0,79	0,82
lBCP	WS=ARCS	PC	0,74	0,65	0,55	**0,78**	0,7	0,6
		RR	0,83	0,85	0,88	**0,73**	0,77	0,8
	WS=ECBS	PC	**0,76**	0,66	0,56	0,78	0,70	0,6
		RR	**0,83**	0,85	0,88	0,72	0,77	0,8
LBSP	WS=ARCS	PC	0,8	0,75	0,70	0,83	0,78	0,72
		RR	0,82	0,85	0,88	0,72	0,76	0,79
	WS=ECBS	PC	**0,81**	0,76	0,70	**0,84**	0,78	0,73
		RR	**0,83**	0,85	0,88	**0,72**	0,76	0,79
lBCE	WS=ARCS	PC	0,81	0,76	0,69	**0,86**	0,82	0,74
		RR	0,82	0,84	0,87	**0,73**	0,77	0,8
	WS=ECBS	PC	**0,83**	0,76	0,66	0,82	0,74	0,6
		RR	**0,81**	0,83	0,84	0,72	0,75	0,77
LECP+lBCE	WS=ARCS	PC	0,65	0,63	0,37	**0,76**	0,70	0,48
		RR	0,85	0,90	0,92	**0,80**	0,86	0,89
	WS=ECBS	PC	**0,78**	0,65	0,51	0,75	0,59	0,5
		RR	**0,86**	0,89	0,92	0,79	0,84	0,88
Default	WS=ARCS	PC	0,89			**0,90**		
		RR	0,78			**0,78**		
	WS=ECBS	PC	**0,90**			0,9		
		RR	**0,78**			0,67		
SN (*baseline*)	$w = 2^{-1}\sqrt{w_{max}}$	PC	0,88					
		RR	0,79					
	$w = 2^{-2}\sqrt{w_{max}}$	PC	**0,83**					
		RR	**0,90**					

Tabela 8.17: Resultados de qualidade utilizando a base de dados FebrlData,

Algoritmo	FebrlData								
	$\{S_{min}, S_{max}\}$		$\{200, 300\}$			$\{300, 400\}$		
	λ		0,2	0,3	0,4	0,2	0,3	0,4
lECP	WS=ARCS	PC	**0,96**	0,96	0,84	**0,96**	0,96	0,84
		RR	**0,93**	0,93	0,96	**0,90**	0,90	0,95
	WS=ECBS	PC	0,83	0,82	0,81	0,85	0,83	0,82
		RR	0,93	0,93	0,93	0,9	0,9	0,95
lBCP	WS=ARCS	PC	**0,92**	0,88	0,75	**0,90**	0,88	0,76
		RR	**0,91**	0,92	0,93	**0,88**	0,89	0,91
	WS=ECBS	PC	0,80	0,73	0,68	0,82	0,79	0,69
		RR	0,91	0,92	0,93	0,88	0,89	0,91
LBSP	WS=ARCS	PC	**0,92**	0,89	0,80	**0,91**	0,89	0,81
		RR	**0,91**	0,92	0,92	**0,87**	0,89	0,90
	WS=ECBS	PC	0,81	0,77	0,72	0,82	0,81	0,74
		RR	0,91	0,92	0,93	0,87	0,89	0,90
lBCE	WS=ARCS	PC	**0,96**	0,94	0,87	**0,98**	0,92	0,86
		RR	**0,91**	0,92	0,93	**0,87**	0,89	0,91
	WS=ECBS	PC	0,87	0,85	0,82	0,87	0,84	0,82
		RR	0,90	0,92	0,93	0,86	0,89	0,9
ECP+lBCE	WS=ARCS	PC	**0,87**	0,80	0,47	**0,88**	0,8	0,46
		RR	**0,94**	0,95	0,98	**0,92**	0,92	0,97
	WS=ECBS	PC	0,79	0,76	0,57	0,82	0,76	0,49
		RR	0,94	0,95	0,97	0,92	0,93	0,97
Default	WS=ARCS	PC	**0,99**			**0,99**		
		RR	**0,88**			**0,85**		
	WS=ECBS	PC	0,87			0,88		
		RR	0,88			0,85		
SN (*baseline*)	$w = 3^{-1}2^{-4}\sqrt{w_{max}}$	PC	**0,99**					
		RR	**0,88**					
	$w = 2^{-6}\sqrt{w_{max}}$	PC	0,99					
		RR	0,76					

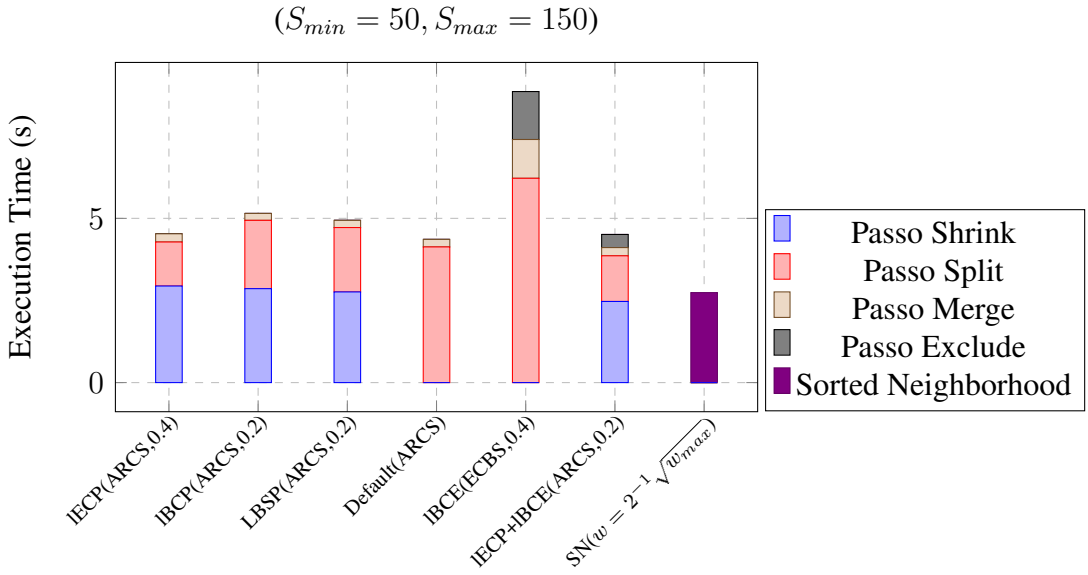

Figura 8.16: Resultados de eficiência utilizando a base de dados CoraATDV.

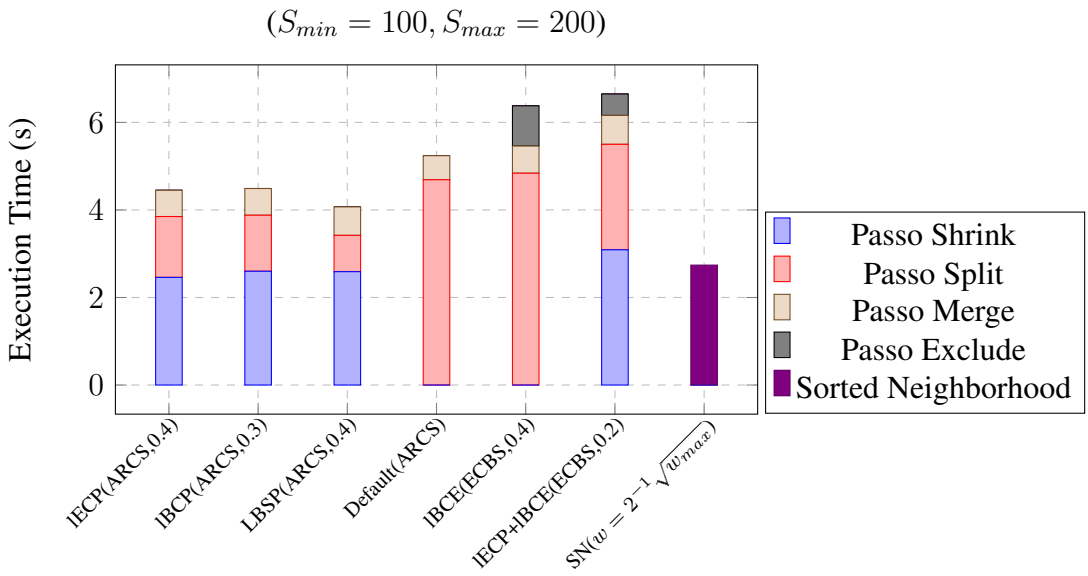

Figura 8.17: Resultados de eficiência utilizando a base de dados CoraATDV.

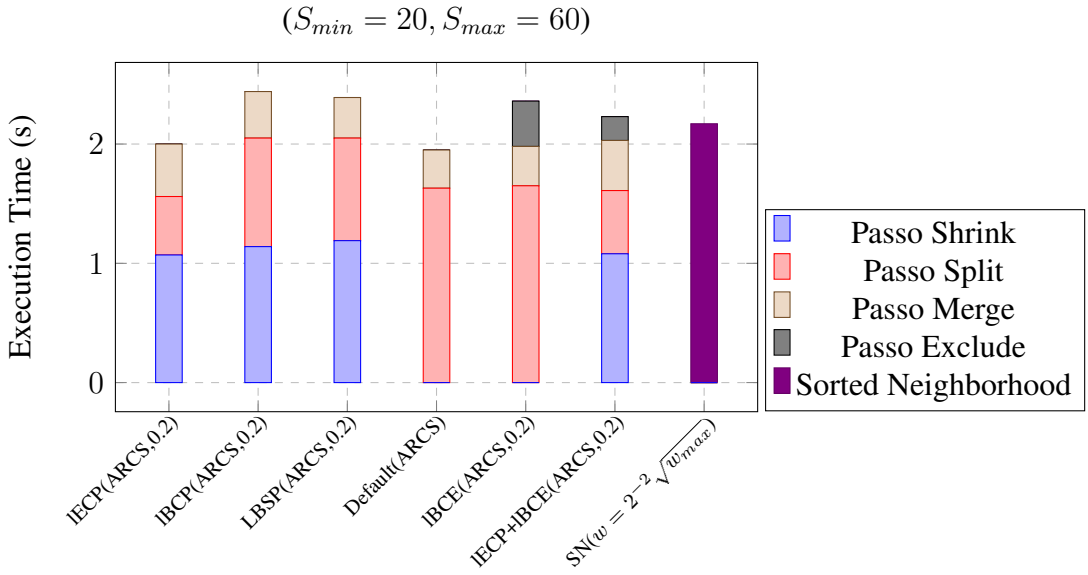

Figura 8.18: Resultados de eficiência utilizando a base de dados DBLPM4.

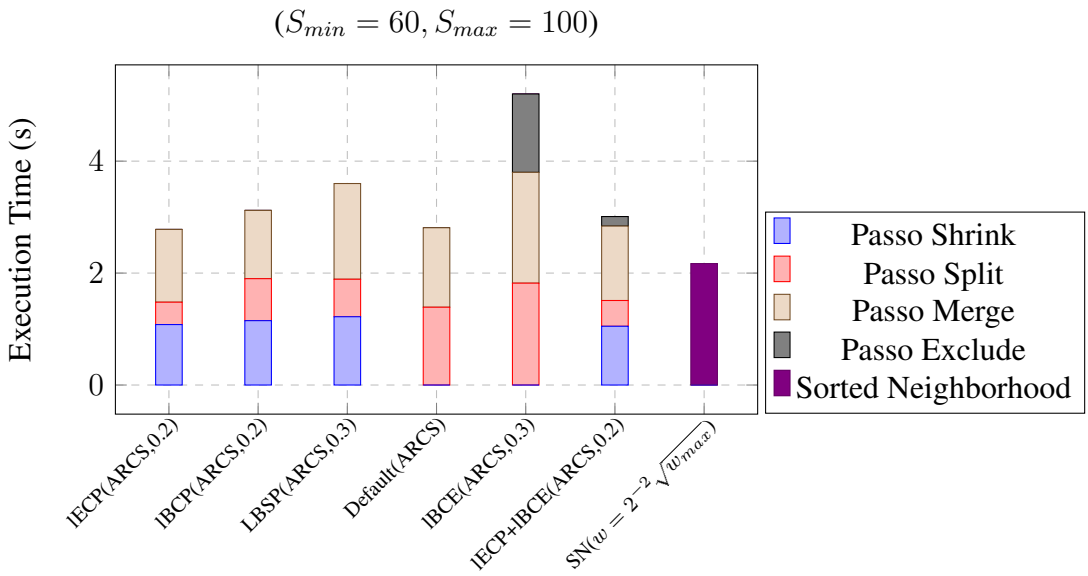

Figura 8.19: Resultados de eficiência utilizando a base de dados DBLPM4.

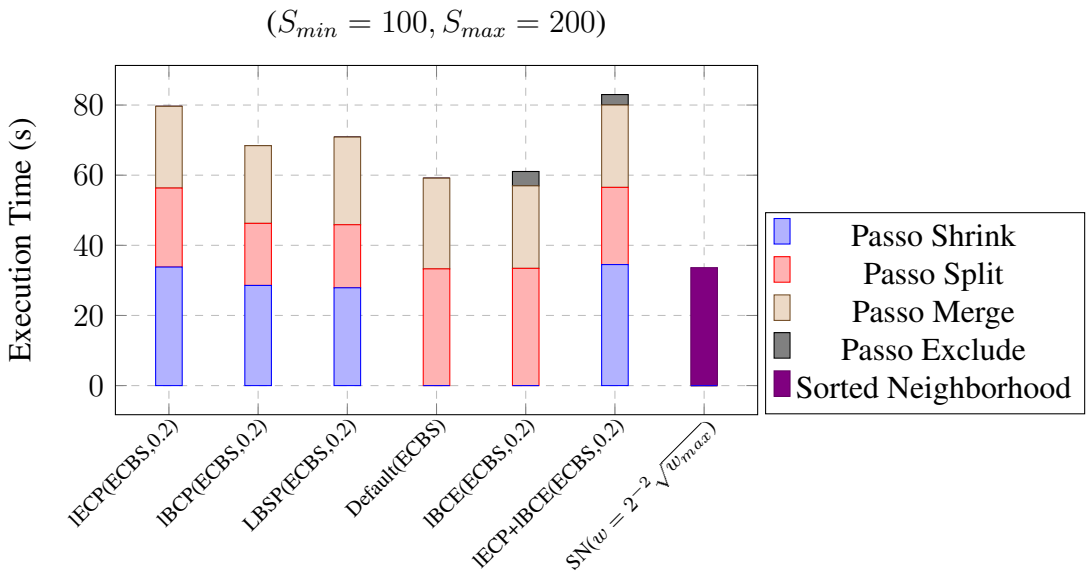

Figura 8.20: Resultados de eficiência utilizando a base de dados DS3.

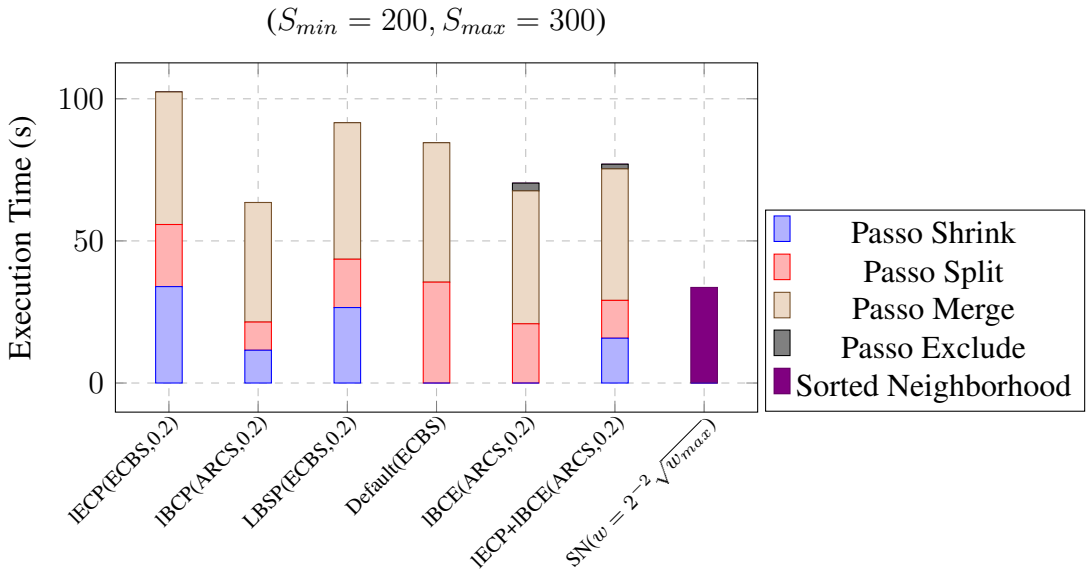

Figura 8.21: Resultados de eficiência utilizando a base de dados DS3.

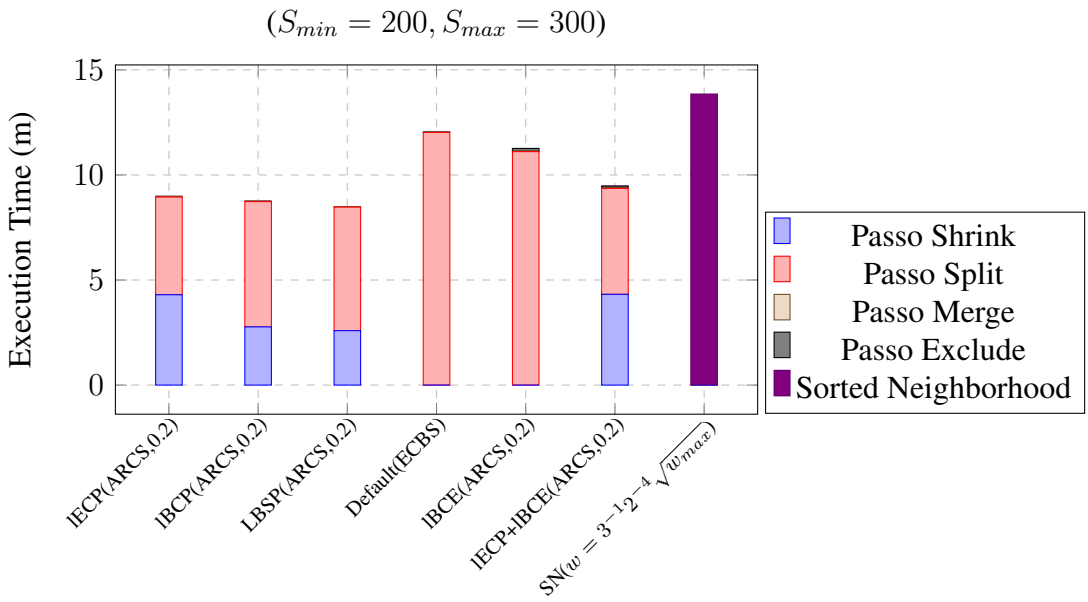

Figura 8.22: Resultados de eficiência utilizando a base de dados FebrlData.

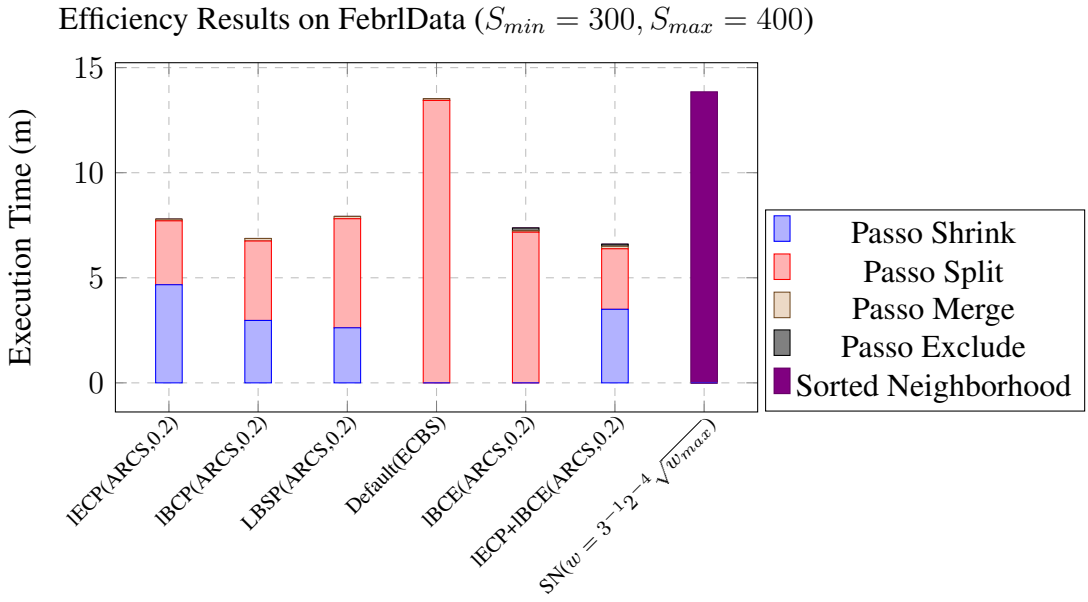

Figura 8.23: Resultados de eficiência utilizando a base de dados FebrlData.

dos pelos algoritmos $lECP$ e $LBSP$ são bastante similares, o que pode ser explicado pelo fato da remoção de entidades de blocos grandes (ver linha 5 do Algoritmo 18) ser suficiente para produzir impactos significativos sobre os resultados de RR e PC. Na grande maioria dos casos, o algoritmo $LBSP$ produziu resultados de PC melhores e resultados de RR menores (em relação ao algoritmo $lECP$), o que foi causado pelo fato do algoritmo $LBSP$ remover entidades apenas de um sub conjunto dos blocos processado pelo algoritmo $lECP$ (ou seja, o conjunto dos blocos grandes).

Na base de dados Cora, o algoritmo $lBCE$ foi a única combinação de heurísticas que produziu resultados de qualidade melhores quando o esquema ECBS foi empregado. Este algoritmo produziu resultados similares aos resultados reportados pelos algoritmos $lBCP$ e $LBSP$ (e resultados ligeiramente menores do que os reportados pelo algoritmo $lECP$). Este resultado destaca que a exclusão de blocos que apresentam baixas pontuações de coocorrência agregada pode ser uma estratégia tão efetiva quando a remoção de entidades dos blocos (realizada pelos algoritmos $lBCE$, $lBCP$ e $LBSP$). Nesta base de dados específica, o algoritmo $lECP + lBCE$ reportou resultados melhores do que os algoritmos $lBCP$, $LBSP$ e $lBCE$ (considerando a média harmônica entre PC e RR), mas ligeiramente menores do que os resultados do algoritmo $lECP$ (especialmente em relação a PC). Por fim, note que o algoritmo $Default$ produziu os maiores resultados de PC (especialmente quando o esquema ARCS e os limites de tamanho maiores foram empregados), mas uma vez que este algoritmo não executa os passos $Shrink$ e $Exclude$ da metaheurística $MkBPCBS$, os resultados de RR reportados pelo algoritmo $Default$ foram significativamente menores do que os das demais abordagens.

Na grande maioria dos casos, as heurísticas propostas reportaram resultados de qualidade superiores aos reportados pelo algoritmo SN, considerando a média harmônica entre PC e RR. É possível notar que, ao empregado o algoritmo SN, o resultado referente a RR pode ser aumentado de maneira simples ao diminuir o tamanho da janela empregado, mas usualmente esta diminuição sacrifica os resultados de PC. Além disso, os resultados de PC produzidos pelo algoritmo SN foram fortemente influenciados pelo tamanho da janela empregado. Por exemplo, ao reduzir o tamanho da janela (w) de $2^{-1}\sqrt{w_{max}}$ para $2^{-2}\sqrt{w_{max}}$, o algoritmo SN reduziu o resultado de PC em 31%. Uma vez que as heurísticas propostas empregam abordagens mais sofisticadas para manipular o tamanho dos blocos, as mesmas produziram

resultados de RR semelhantes aos produzidos pelo algoritmo SN, mas mantiveram resultados de PC superiores.

Em relação aos resultados de PQ, todos os algoritmos avaliados reportaram resultados entre 0.3 e 1.1. Em geral, quando mais abordagens de poda (remoção de entidades de blocos e exclusão de blocos) são empregadas, maior tende a ser o resultado de PQ reportado pelo algoritmo. Por esta razão, o algoritmo $lECP + lBCE$ empregando os maiores valores de λ avaliados produziu os maiores valores de PQ. Finalmente, note que, como esperado, todos os algoritmos reportaram resultados de PC ligeiramente melhores quando o intervalo de tamanho com limites maiores foi empregado, mas usualmente sacrificando resultados de RR (em relação aos resultados do mesmo algoritmo empregando o intervalo de tamanho com limites menores). Este padrão nos resultados é esperado uma vez que quanto maiores os tamanhos dos blocos gerados, maior é a chance de identificar entidades duplicadas e menor se torna a cardinalidade agregada da coleção de blocagens podada. Por sua vez, em relação ao algoritmo SN, quanto maior é o tamanho da janela empregado, menor tende a ser os resultados de PQ produzidos. Este algoritmo reportou resultados de PQ entre 0.04 e 0.08. Portanto, todas as heurísticas propostas produziram resultados de PQ superiores ao reportados pelo algoritmo SN.

Cora (*Eficiência*).

Considerando os resultados na Figura 8.16, é fácil notar que os algoritmos avaliados reportaram tempos de execução bastante eficientes. No geral, os passos $Shrink$ e $Split$ requerem mais tempo para serem executados do que os passos $Merge$ e $Exclude$. Note também que quando o passo $Shrink$ da metaheurística não é executado pelo algoritmo (por exemplo, quando os algoritmos $lBCE$ e $Default$ são empregados) o passo $Split$ se torna mais custoso. Em relação ao intervalo de tamanho com limites menores $\{50, 150\}$ (Figura 8.16), os algoritmos $lECP$ e $Default$ reportaram os menores tempos de execução dentre os algoritmos avaliados. Note também que o tempo de execução reportado pelo algoritmo $lBCE(ECBS, 0.4)$ é significativamente maior do que os tempos de execução dos demais algoritmos (especialmente no passo $Split$). Este resultado pode ser explicado pelo fato deste algoritmo utilizar o esquema de pontuação ECBS e este esquema utilizar uma fórmula mais complexa do que a fórmula empregada pelo esquema $ARCS$. Por este motivo, os algoritmos que utilizaram o esquema de pontuação ECBS reportaram tempos de execução maiores

(quando comparados aos tempos de execução do mesmo algoritmo utilizando o esquema ARCS).

Em relação ao intervalo de tamanho que emprega limites maiores $\{100, 200\}$ (Figura 8.17), note que, em geral, os algoritmos reportaram tempos de execução maiores para completar o processo (quando comparados aos tempos de execução utilizando intervalo de tamanho que emprega limites menores, i.e., Figura 8.16), especialmente nos passos $Shrink$ e $Split$. Considerando o maior intervalo de tamanho, os algoritmos $LBSP$, $lECP$ e $lBSP$ reportaram os menores tempos de execução, seguidos pelos demais algoritmos avaliados. Por fim, em ambos os intervalos de tamanho avaliados, o algoritmo SN executou mais rapidamente do que as heurísticas propostas (com o custo de produzir resultados de qualidade inferiores, como mostrado na Tabela 8.14).

DBLPM4 (*Resultados de Qualidade*).

Considerando os resultados na Tabela 8.15, note que os resultados reportados pela maioria dos algoritmos foi fortemente influenciado pelo valor de λ empregado. Em todos os casos, valores de λ menores produziram resultados de PC maiores e resultados de RR maiores (quando comparados com os resultados produzidos por valores de λ maiores). Além disso, similarmente aos resultados na base de dados Cora, todos os algoritmos produziram resultados de qualidade melhores (considerando a média harmônica entre PC e RR) quando o esquema ARCS foi empregado. Com relação à variação do intervalo de tamanho, note que em todos os casos a utilização de um intervalo de tamanho com limites maiores produziu resultados de RR ligeiramente menores e resultados de PC maiores (quando comparados com os resultados empregando o intervalo de tamanho com limites menores).

Na base de dados DBLPM4, os algoritmos que executam apenas o passo $Shrink$ ($lECP$, $lBCP$ e $LBSP$) produziram resultados similares. Em particular, o algoritmo $lECP$ reportou resultados de qualidade ligeiramente melhores do que os outros dois algoritmos ($lBCP$ e $LBSP$), especialmente empregando o esquema ARCS combinado com baixos valores de λ. Por sua vez, o algoritmo $lBCE$ reportou resultados de PC bastante altos, mas gerou resultados de RR menores dos que os resultados produzidos pelo algoritmo $lECP$. O algoritmo $lECP + lBCE$ produziu resultados de PC muito menores do que os produzidos pelos demais algoritmos, mas produziu os melhores resultados de RR, o que indica que à medida em que as bases de dados se tornam maiores, os resultados se tornam mais sensíveis ao valor

de λ utilizado e ao emprego de duas abordagens de poda (os passos $Shrink$ e $Exclude$ da metaheurística $MkBPCBS$). Por fim, semelhante aos resultados obtidos na base de dados Cora, o algoritmo $Default$ produziu os maiores valores de PC com o custo de gerar os menores valores de RR dentre os algoritmos avaliados.

Os resultados de PC produzidos pelo algoritmo SN foram razoavelmente influenciados pelo tamanho da janela w empregado. Ao empregar janelas de tamanho menores, este algoritmo aumentou os resultados de RR sem sacrificar os resultados de PC significativamente. No entanto, os melhores resultados produzidos pelo algoritmo SN ($w = 2^{-1}\sqrt{w_{max}}$) foram inferiores ao resultado produzido pelo algoritmo $lECP$ usando o esquema de pontuação ARCS e $\lambda = 0.2$ (considerando a média harmônica entre PC e RR).

Em relação aos resultados de PQ, todos os algoritmos reportaram resultados entre 0.03 e 0.1, ao passo que o algoritmo SN reportou resultados de PQ entre 0.02 e 0.09 (como discutido anteriormente, quando menor o tamanho da janela w empregado, maior tende a ser o resultado de PQ produzido). Similarmente aos resultados na base dados Cora, os algoritmos que executaram mais passos de poda dos blocos ($Shrink$ e $Exclude$) e empregaram maiores valores de λ produziram os maiores valores de PQ. Considerando todos os resultados reportados na Tabela 8.15, o algoritmo $lECP$ utilizando o esquema de pontuação de coocorrência ARCS e $\lambda = 0.2$ reportou o melhor resultado considerando a média harmônica entre PC e RR.

DBLPM4 (*Eficiência*).

Semelhante aos resultados na base de dados Cora, na base de dados DBLPM4 os algoritmos que empregaram o esquema ARCS reportaram tempos de execução menores do que os algoritmos que empregaram o esquema ECBS. Note que para o intervalo de tamanho que emprega limites menores (Figura 8.18), os algoritmos requereram mais tempo para executar os passos $Shrink$ e $Split$. Por sua vez, para o intervalo de tamanho que emprega limites maiores (Figura 8.19), os algoritmos requereram mais tempo para executar os passos $Shrink$ e $Merge$. Assim, é fácil notar que o intervalo de tempo empregado pode influenciar significativamente o tempo necessário para executar cada um dos passos da metaheurística $MkPCBS$ e o tempo de execução total dos algoritmos. Para ambos intervalos de tamanho empregados, dois dos algoritmos avaliados ($lECP$ e $Default$) produziram os menores tempos de execução.

Outro padrão interessante pode ser notado na Figura 8.19, na qual o passo $Exclude$ reportado pelo algoritmo $lECP + lBCE$ foi executado mais rapidamente do que o passo $Exclude$ reportado pelo algoritmo $lBCE$, o que indica que o passo $Shrink$ (executado pelo algoritmo $lBCE$) pode influenciar significativamente os tempos de execução dos passos subsequentes da metaheurística $MkPCBS$. Similarmente aos resultados com a base de dados Cora, o algoritmo SN reportou resultados de eficiência inferiores aos reportados pelas heurísticas propostas.

DS3 (*Resultados de Qualidade*).

Similarmente aos resultados na base de dados DBLPM4, na Tabela 8.16 é fácil notar que os resultados de todos os algoritmos foram fortemente influenciados pelo valor de λ empregado (por exemplo, note a diferença entre os resultados de qualidade produzidos pelos valores $\lambda = 0.3$ e $\lambda = 0.4$). No entanto, diferente dos resultados produzidos na base de dados DBLPM4, na base de dados DS3 a maioria dos algoritmos reportaram melhores resultados (considerando a média harmônica entre PC e RR) quando o esquema ECBS foi empregado. O algoritmo $lECP$ reportou resultados significativamente melhores do que os demais algoritmos (especialmente para baixos valores de λ). Por sua vez, os algoritmo $lECP + lBCE$ produziu resultados de PC bastante baixos (quando comparados com os resultados de PC dos demais algoritmos), o que indica mais uma vez (similarmente aos resultados na base de dados DBLPM4) que uma abordagem que executa ambos os passos $Shrink$ e $Exclude$ não é adequada para controlar o tamanho de blocos em bases de dados grandes (especialmente se valores de λ grandes forem empregados). Similarmente aos resultados nas bases de dados Cora e DBLPM4, a variação do limite de tamanho empregada na Tabela 8.16 produziu pequenas melhorias nos resultados de PC ao custo de sacrificar ligeiramente os resultados de RR.

Na Tabela 8.16, o tamanho da janela empregado produziu influência moderada sobre os resultados gerados pelo algoritmo SN. No melhor cenário ($w = 2^{-2}\sqrt{w_{max}}$), este algoritmo produziu resultados similares aos reportados pela heurística proposta que produziu a melhor média harmônica entre os resultados de PC e RR (o algoritmo $lECP$ empregando o esquema de pontuação ECBS e $\lambda = 0.2$).

Em relação aos resultados de PQ, todos os algoritmos reportaram resultados entre 0.002 e 0.005, ao passo que o algoritmo SN produziu resultado de PQ aproximadamente iguais a

0.01. Além disso, o mesmo padrão envolvendo o valor de λ empregado, a execução dos passos $Shrink$ e $Exclude$ e os resultados de PQ discutidos em relação às bases de dados Cora e DBLPM4 foram também observados na base de dados DS3. Finalmente, considerando todas as combinações de valores de parâmetros apresentados na Tabela 8.16, o algoritmo $lECP$ empregando o esquema de pontuação de coocorrência ARCS e $\lambda = 0.2$ produziu o melhor resultado de qualidade considerando a média harmônica entre PC e RR.

DS3 (*Eficiência*).

Similarmente aos resultados obtidos com as bases de dados Cora e DBLPM4, ao processar a base de dados DS3 os algoritmos que empregaram o esquema ARCS produziram tempos de execução menores do que os algoritmos que empregaram o esquema ECBS. É também importante notar que para o intervalo de tamanho que emprega limites menores, na grande maioria dos casos os algoritmos reportaram tempos de execução menores (quando comparados com o intervalo de tamanho que emprega limites maiores). Nas Figuras 8.20 e 8.21, os algoritmos $lBCE$, $lBCP$ e $Default$ reportaram os menores tempos de execução. Na maioria dos casos, os algoritmos requereram mais tempos para executar os passo $Shrink$ e $Merge$, o que indica que no caso específico da base de dados DS3 os algoritmos precisaram realizar uma grande quantidade de operações de união entre blocos para garantir um tamanho mínimo (S_{min}) dos blocos.

É também importante destacar o fato do passo $Exclude$ ter sido executado de maneira bastante eficiente pelos algoritmos $lBCE$ e $lECP + lBCE$ (em relação aos tempos utilizados nos demais passos, i.e., $Shrink$, $Merge$ e $Split$). Finalmente, similarmente aos resultados obtidos com as bases de dados Cora e DBLPM4, ao processar a base de dados DS3 o algoritmo SN reportou tempos de execução inferiores aos reportados pelas heurísticas propostas.

FebrlData (*Resultados de Qualidade*).

Similarmente aos resultados obtidos com as bases de dados Cora e DS3, na Tabela 8.17 os resultados de qualidade reportados por todos os algoritmos foram também fortemente influenciados pelo valor de λ empregado (por exemplo, note as diferenças significativas entre os resultados de qualidade produzidos pelo emprego de $\lambda = 0.2$ e $\lambda = 0.4$). Além disso, similarmente aos resultados das bases de dados DBLPM4 e CoraATDV, os resultados de qualidade dos algoritmos que empregaram o esquema ARCS foram melhores do que os

resultados dos algoritmos que utilizando o esquema ECBS.

Os algoritmos $lECP$ e $lBCE$ reportaram resultados de qualidade significativamente melhores (considerando a média harmônica entre PC e RR) do que as outras abordagens (especialmente para valores de λ pequenos). Note também que o algoritmo $LECP + lBCE$ produziu resultados de PC bastante baixos (quando comparados com os demais resultados na Tabela 8.17), o que reforça a discussão relacionada a este ponto apresentada sobre a base de dados DS3 e indica mais uma vez que um abordagem para controlar o tamanho de blocos baseada em duas estratégias de poda (passos $Shrink$ e $Exclude$) não é ideal para processar bases de dados grandes (especialmente empregando altos valores de λ). Por sua vez, o algoritmo $Default$ reportou resultados de PC bastante elevados ($\approx 100\%$), especialmente quando o esquema de pontuação de coocorrência ARCS foi empregado, mas ao mesmo tempo produziu os menores valores de RR da Tabela 8.17. Similarmente aos resultados obtidos com as bases de dados Cora e DS3, a heurística proposta que produziu os melhores resultados de qualidade ($lECP$ usando o esquema de pontuação ARCS e $\lambda = 0.2$) superou os resultados produzidos pelo algoritmo SN (considerando a média harmônica entre PC e RR).

De forma semelhante aos resultados obtidos com as bases de dados Cora e DBLPM4, a variação dos limites de tamanho empregados na Tabela 8.17 produziram pequenas melhorias nos resultados de PC reportados pelos algoritmos ao custo de sacrificar ligeiramente os resultados de RR. Com relação aos resultados de PQ, todos os algoritmos reportaram resultados entre 1.10^{-4} e 7.10^{-4} e, mais uma vez, o padrão já discutido envolvendo os resultados de PQ, a quantidade de passos de poda e o valor de λ empregado é mantido; isto é, quanto mais passos de poda são executados pelo algoritmos e maior é o valor de λ empregado, maior tende a ser o resultado de PQ produzido. Por sua vez, o algoritmo SN reportou resultados de PQ entre 7.10^{-3} e 4.10^{-3}. Por fim, considerando todas as combinações de valores de parâmetros reportadas na Tabela 8.17, o algoritmo $lECP$ empregando o esquema de pontuação de coocorrência ARCS e $\lambda = 0.2$ produziu o melhor valor média harmônica entre PC e RR.

FebrlData (*Eficiência*).

Semelhante aos resultados de eficiência obtidos com todos as demais bases de dados avaliadas no Experimento IV, ao processar a base de dados FebrlData todos os os algoritmos que empregaram o esquema ARCS produziram tempos de execução menores do que os tempos

de execução dos algoritmos que empregaram o esquema ECBS. Nas Figuras 8.22 e 8.23, os algoritmos $lECP, lBCP, LBSP$ e $lBCE$ apresentaram tempos de execução próximos, seguidos do tempo de execução algoritmo $Default$. Em particular, o algoritmo $Default$ reportou tempos de execução significativamente altos, o que pode ser explicado pela custosa execução do passo $merge$, a qual foi causada pelo fato do algoritmo não executar o passo $Shrink$. Em todos os casos nas Figuras 8.22 e 8.23, os algoritmos passaram a maior parte do tempo executando o passo $Merge$, o que indica que, similarmente aos resultados obtidos com o processamento da base de dados DS3, uma grande quantidade de operações de união entre blocos foi necessária para garantir um tamanho mínimo dos blocos na coleção de blocagens podada.

Similarmente aos resultados obtidos com as outras três bases de dados avaliadas, o passo $Exclude$ foi realizado de maneira bastante eficiente pelos algoritmos $lBCE$ e $lECP + lBCE$ (quando comparado aos tempos de execução reportados pelos outros passos dos algoritmos). Finalmente, ao processar a base de dados FebrlData, o algoritmo SN reportou tempos de execução semelhantes aos reportados pelas heurísticas propostas.

Síntese e Discussão das Hipóteses.

Com base nos resultados apresentados nas Tabelas 8.14-8.17, é possível observar que as diferenças nos resultados de PC reportados pelos algoritmos avaliados podem podem variar em até 54%, além de claramente possuírem forte influência do esquema de pontuação de coocorrência e no valor de λ empregados. Desse modo, os resultados apresentados e a discussão relacionada apresentam suporte para as hipóteses *H7*, *H9* e *H10*, isto é, os resultados de qualidade podem variar significativamente com base no algoritmo de controle de tamanho de blocos e no conjunto de valores de parâmetros empregados. De maneira análoga, analisando os dados apresentados nas Figuras 8.16-8.23, em muitos casos os resultados de eficiência (tempos de execução) dos algoritmos avaliados podem variar em mais de 50%. Assim, os resultados apresentados e a discussão relacionada também apresentam suporte para as hipóteses *H8* e *H11*, ou seja, resultados de eficiência podem variar significativamente com base no algoritmo de controle de tamanho de blocos empregado e no esquema de pontuação de coocorrência empregado.

8.6 Ameaças à Validade

Neste seção, são apresentadas as principais ameaças à validade dos resultados e conclusões provenientes dos experimentos conduzidos neste trabalho.

Ameaças à validade do Experimento I. Primeiramente, este experimento adotou um valor aleatório de $speedup$ (entre um intervalo pré-definido) no modelo matemático adotado. Este fato, somado aos dados terem sido provenientes de um ambiente simulado, pode ter enviesado os tempos de execução dos algoritmos de qualidade de dados. Além disso, apenas duas técnicas de aprendizado de máquina foram consideradas. Uma avaliação mais abrangente, englobando outras técnicas mais avançadas de aprendizado de máquina, poderia produzir uma quantidade maior e mais diversificada de resultados, permitindo uma discussão mais ampla sobre a eficácia destas técnicas no contexto de múltiplas execuções de algoritmos de qualidade de dados disparados por um mesmo DQSLA. Além disso, a avaliação utilizou uma variação significativa de índices de classes de configuração (variando a 1 a 400). Em um cenário que apresente uma variação menor no intervalo de índices de classes de configuração disponíveis, os resultados obtidos podem ser significativamente diferentes. Tais cenários não foram explorados no Experimento I. Por fim, a grande maioria dos valores dos parâmetros globais adotados no Experimento I (ver Tabela 8.3) não foram variados no design do experimento e, por este motivo, não é possível discutir ou predizer o real impacto dos valores dos parâmetros adotados sobre os resultados obtidos com o experimento.

Ameaças à validade do Experimento II. O modelo matemático adotado pelo Experimento II para simular a execução dos algoritmos de qualidade de dados assume duas suposições básicas: i) o valor de $speedup$ associado aos tempos de execução dos algoritmos de qualidade de dados em um ambiente distribuído cresce de maneira aproximadamente linear à medida que mais nós são adicionados ao ambiente; e ii) quanto maior a quantidade de nós adicionados ao ambiente distribuído, maior tende a ser a diferença entre a quantidade de nós alocados e o valor de $speedup$ da execução distribuída da tarefa de qualidade de dados. Ainda que estas possam ser consideradas suposições bastante razoáveis (pelo fato destes fenômenos serem observados em resultados do estado da arte), tais suposições (juntamente com o fato de ter sido utilizado um ambiente simulado) podem nem sempre ser verdadeiras e/ou apresentar padrões constantes em ambientes reais, principalmente ao longo do proces-

samento de uma grande quantidade de tarefas de qualidade de dados por um SMQD, como foi avaliado no Experimento II. Outrossim, assim como no Experimento I, a grande maioria dos valores dos parâmetros globais adotados no Experimento II (ver Tabela 8.6) não foram variados no design do experimento, limitando a possibilidade de avaliar a influência destes parâmetros nos resultados obtidos. Finalmente, a avaliação utilizou uma variação significativa de índices de classes de configuração (variando a 1 a 400) e uma grande quantidade de tarefas de deduplicação de dados na carga de trabalho. Caso os experimentos tivessem adotado uma variação menor dos índices de classe de configuração disponíveis e uma menor quantidade de tarefas de qualidade de dados, os resultados poderiam ter sido diferentes dos resultados apresentados no Experimento II.

Ameaças à validade do Experimento III. A principal ameaça à validade do Experimento III diz respeito ao fato de os valores de vários parâmetros globais do experimento não terem sido variados, dentre os quais: μ, a função de similaridade adotada ($jaro\ winkler$), θ e a chave de bloco empregada na base de dados $Febrl10k$. Além disso, duas das bases de dados adotadas no experimento foram geradas artificialmente, o que limita a generalização dos resultados obtidos (em especial os resultados de eficácia) em um cenário utilizando bases de dados reais. Por fim, os incrementos nos dados gerados a partir da aplicação do processo proposto pelos autores em [30] não inclui alterações que realizam atualizações nos valores dos dados (apenas inserções e remoções de entidades), o que pode limitar a quantidade de agrupamentos presentes nos componentes de cobertura selecionados e, consequentemente, afetar negativamente a generalização dos resultados obtidos.

Ameaças à validade do Experimento IV. As principais ameaças dizem respeito ao tamanho das bases de dados avaliadas e ao fato de dois parâmetros não terem sido variados de maneira significativa nos experimentos. Em relação ao tamanho das bases de dados empregadas, caso as heurísticas fossem avaliadas utilizando bases de dados maiores, é possível que os resultados de qualidade obtidos tivessem sido significativamente diferentes. Além disso, caso as heurísticas fossem avaliadas utilizando bases de dados maiores, seria possível discutir de maneira mais embasada a escalabilidade dos resultados de eficiência obtidos. Por sua vez, os parâmetros que não tiveram valores variados de maneira significativa nos experimentos foram: os conjuntos de chaves de bloco empregados para indexar as bases de dados (foi utilizado apenas um conjunto por base de dados) e os intervalos de tamanho (foram ava-

liados apenas dois intervalos de tamanho por base de dados). Devido a esta limitação nos experimentos, não é possível afirmar a real influência dos valores de tais parâmetros sobre os resultados das heurísticas, ou seja, não é possível afirmar o nível de sensibilidade dos resultados obtidos em relação ao conjunto de funções de chave de bloco e aos intervalos de tamanho empregados.

Capítulo 9

Conclusões e Trabalhos Futuros

Neste capítulo são apresentadas as conclusões do trabalho e as principais perspectivas de trabalhos futuros.

9.1 Conclusões

O principal objetivo deste trabalho consistiu em propor e avaliar abordagens para reduzir os custos provenientes da execução de soluções para deduplicação de dados no contexto de um SMQD. Para tal, foram investigados e propostos diversos modelos, algoritmos e abordagens que empregam heurísticas, técnicas de aprendizado de máquina e a capacidade de provisionamento dinâmico de recursos computacionais provida pelo paradigma de computação em nuvem. Visando atingir o objetivo elencado, foram propostos: i) uma estrutura para representar um SLA de qualidade de dados; ii) uma arquitetura em alto nível para um SMQD; iii) um modelo de custo para um SMQD; iv) a formalização do problema de alocação de recursos computacionais para a execução de tarefas de deduplicação de dados, as quais são associadas a restrições de tempo especificadas em um SLA de qualidade de dados, em um ambiente de computação em nuvem; v) algoritmos para o tratamento do problema do item iv; vi) novas métricas e definições para o problema de DID utilizando classificação coletiva; vii) heurísticas para o problema de deduplicação incremental de dados utilizando classificação coletiva e viii) heurísticas para o problema de controle de tamanho de blocos produzidos por múltiplas chaves de bloco no contexto de deduplicação de dados. Além disso, foram realizadas quatro avaliações experimentais para validar a eficácia e eficiência dos algoritmos

propostos no trabalho.

Os resultados do Experimento I, que visou avaliar os algoritmos de alocação de recursos computacionais para execução de tarefas de qualidade de dados disparadas por um mesmo SLA de qualidade de dados, dão suporte para a hipótese H1, ou seja, a adoção de técnicas de aprendizado de máquina (baseadas em uma base de treinamento) são capazes de reduzir tanto os custos de infraestrutura quanto os custos de penalidade associados à execução de tarefas de deduplicação de dados no contexto de um SMQD, em comparação com a eficácia de heurísticas propostas para o mesmo fim. No caso do Experimento I, as técnicas de aprendizado de máquina KNN e seleção de protótipos apresentaram eficácia superior às heurísticas propostas para a estimativa de uma classe de configuração inicial para a execução de uma tarefa de qualidade de dados.

Prosseguindo, os resultados do Experimento II, que teve como objetivo avaliar algoritmos de alocação de recursos computacionais para execução de tarefas de deduplicação de dados disparadas por diferentes SLAs de qualidade de dados, suportam as hipóteses H1, H2 e H3. Desse modo, é possível concluir que i) as técnicas de aprendizado de máquina KNN e $randomforests$ apresentaram melhor eficácia (na redução de custos de infraestrutura e de quantidade de violações de SLA) em relação à técnica SVM; e ii) as heurísticas propostas (em especial, $sliced\ traning\ data$, $adaptive\ allocation$ e $probabilistic\ best\ performing\ allocation$) contribuíram positivamente para a eficácia dos algoritmos de alocação de recursos computacionais; iii) a adoção de diferentes heurísticas e/ou técnicas de aprendizado de máquina para tratar o problema de alocação dinâmica de recursos em um SMQD pode influenciar significativamente os custos de infraestrutura e penalidades do serviço.

Por sua vez, os resultados obtidos com a condução do Experimento III dão suporte para duas das hipóteses levantadas (H4 e H6) e indicam a rejeição da hipótese H5. Em resumo, a combinação de algoritmos de agrupamento automático de baixa complexidade com estratégias de minimização de função de penalidade produziu resultados ineficientes. Além disso, é também possível concluir que a aplicação de algoritmos de agrupamento automático de baixa complexidade apresentou tempos de execução menores do que o algoritmo do estado da arte ($Greedy$), mas usualmente sacrificando resultados de precisão. Por fim, as heurísticas baseadas em filtros de cobertura propostas produziram resultados mais eficientes do que os

resultados do algoritmo $Greedy$, mas os filtros usualmente produzem resultados de eficácia ligeiramente inferiores.

Finalmente, com base nos resultados obtidos a partir da condução do Experimento IV, é possível concluir que as heurísticas propostas são capazes de controlar eficientemente o tamanho dos blocos produzidos por múltiplas chaves de bloco (utilizando um intervalo de tamanho configurável) e, ao mesmo tempo, manter resultados de eficácia (*Pair Completeness* e *Reduction Ratio*) satisfatórios. A partir da metaheurística proposta ($MkPCBS$), foi possível explorar uma grande quantidade de combinações de heurísticas e, com base nos resultados obtidos, é possível concluir que os resultados são fortemente influenciados pelos valores dos parâmetros empregados (as heurísticas, o esquema de peso de coocorrência e o valor de λ). Em geral, os algoritmos de poda $lBCE$ e $lBCP$ apresentaram os melhores resultados de eficácia, especialmente quando são empregados baixos valores de λ e o esquema de pontuação $ARCS$. Por sua vez, foi também observado que os resultados de eficiência (tempo de execução) das heurísticas são fortemente influenciados pelo esquema de pontuação empregado. Neste contexto, o esquema $ARCS$ produziu tempos de execução bastante inferiores aos tempos de execução gerados pelo esquema $ECBS$.

Com base nos resultados e contribuições obtidos com as pesquisas conduzidas neste trabalho, conclui-se que a proposição e avaliação de abordagens, modelos, técnicas e algoritmos que visam reduzir custos de um SMQD é de fato uma investigação promissora e relevante por três motivos principais: i) é possível concluir a partir dos resultados dos experimentos conduzidos que os custos do SMQD são fortemente influenciados pelos algoritmos de qualidade de dados empregados; ii) os algoritmos de qualidade de dados utilizam uma grande quantidade de parâmetros que podem apresentar forte influência sobre os resultados de eficácia e eficiência dos algoritmos e, portanto, também influenciam os custos do SMQD; e iii) os custos de infraestrutura e penalidade do SMQD são os principais custos a serem considerados para calcular os custos finais repassados a um cliente do serviço e, por esta razão, são cruciais para garantir a competitividade nos preços do serviço. As pesquisas desenvolvidas neste trabalho apresentam várias avaliações experimentais, cujos resultados demonstram diversos padrões nos resultados produzidos pelos algoritmos e, portanto, servem como uma investigação inicial que elenca uma série de modelos, algoritmos e valores de parâmetros que podem ser explorados na prática visando reduzir custos operacionais de um SMQD.

9.2 Trabalhos Futuros

Nesta seção, são elencadas oito temáticas de perspectivas de extensão e trabalhos futuros no contexto do trabalho desenvolvido.

Avaliação de outras técnicas de IA para o provisionamento dinâmico de recursos computacionais. As avaliações dos algoritmos de provisionamento dinâmico de recursos computacionais no contexto de um SMQD desenvolvidas neste trabalho exploraram as seguintes técnicas de aprendizado de máquina: kNN, $randomforests$ e SVM. Ainda que tenha sido possível explorar um *design* de experimentos complexo a partir da combinação destas técnicas de aprendizado de máquina (e respectivas variações de valores de parâmetros) com as heurísticas propostas, existem outras técnicas de aprendizado de máquina disponíveis no estado da arte (tais como *Gaussian Mixture Model* e *Deep Learning*) que também podem ser empregadas no contexto de provisionamento de recursos para um SMQD. Similarmente, é também possível investigar a possibilidade de empregar técnicas de regressão neste contexto e comparar os resultados obtidos com as técnicas de aprendizado de máquina.

Além disso, como elencados na seção de ameaças à validade (Seção 8.6), diversos valores de parâmetros (associados aos algoritmos e ao ambiente de execução) não foram variados na condução dos Experimentos I e II. Ademais, os Experimentos I e II adotaram algoritmos de provisionamento dinâmico computacionais que sempre alocam *clusters* de máquinas virtuais com configuração homogênea. Desse modo, a condução de futuros experimentos que variem os valores dos parâmetros não explorados nos Experimentos I e II ou empreguem *clusters* de máquinas virtuais com configurações heterogêneas irá permitir uma avaliação e discussão associada à sensibilidade dos resultados com relação aos valores de parâmetros não investigados neste trabalho.

Avaliação de escalabilidade dos algoritmos incrementais de qualidade de dados. Outra perspectiva de trabalhos futuros consiste em avaliar algoritmos incrementais de qualidade de dados utilizando bases de dados significativamente maiores do que as bases de dados empregadas no Experimento III. Estas avaliações permitirão uma discussão a respeito da eficácia dos algoritmos de DID ao processar bases de dados maiores e da escalabilidade destes algoritmos ao processar incrementos que afetem um subgrafo muito mais denso e englobando uma grande quantidade de vértices. Desse modo, é possível avaliar principalmente a efici-

ência de técnicas baseadas em minimização de função de penalidade (que são usualmente custosas) e a eficácia (i.e., número de duplicatas detectadas) de técnicas de agrupamento automático de baixa complexidade (que no geral são bastante eficientes, mas usualmente produzem baixos resultados de precisão).

Ajuste automático nos valores de parâmetros de heurísticas. Em todos os experimentos conduzidos neste trabalho, as avaliações das heurísticas propostas para realizar DID no contexto de classificação coletiva e para controlar o tamanho de blocos utilizam uma série de valores de parâmetros de entrada, os quais podem influenciar fortemente a eficácia e a eficiência dos resultados produzidos pelas heurísticas (como observado nos resultados experimentais obtidos). Em geral, os designs experimentais empregados variam os valores destes parâmetros (o que permitiu discutir a influência dos mesmos sobre os resultados), mas na prática é difícil estimar de antemão os valores mais adequados para os parâmetros das heurísticas. Neste contexto, podem ser investigadas técnicas para ajustar automaticamente os valores de parâmetros de heurísticas para realizar DID no contexto de classificação coletiva e para controlar o tamanho de blocos com base em valores de estatísticas provenientes do perfilamento das bases de dados.

Por exemplo, as estatísticas da base de dados poderiam ser utilizadas para ajustar automaticamente o valor do parâmetro μ do algoritmo $\mathcal{Z}\text{-}Filter$ (Algoritmo 14), como também o valor do parâmetro λ da metaheurística $MkPCBS$ (Algoritmo 6).

Paralelização das heurísticas propostas. Em um cenário real, no intuito de utilizar as heurísticas propostas neste trabalho no contexto de um SMQD, é necessário implementar tais heurísticas de modo a serem executadas em um ambiente distribuído (possibilitando assim, o emprego do paradigma de computação em nuvem). Desse modo, são necessários futuros esforços visando implementar tais heurísticas utilizando arcabouços consolidados para execuções distribuídas, tais como Hadoop e Spark, levando em consideração também a resolução de problemas como o desbalanceamento de carga e a minimização da replicação dos dados enviados para as máquinas virtuais utilizadas.

Especificação detalhada do parâmetro M_{rules} do DQSLA. Na prática, os valores que compõem o parâmetro M_{rules} de um DQSLA dependem da dimensão de qualidade de dados considerada (valor do parâmetro DQ_{dim} do DQSLA). Assim, torna-se necessário desenvolver uma especificação para os diversos parâmetros relevantes para a avaliação das bases

de dados considerando dimensões de qualidade de dados específicas. Estes parâmetros são usualmente relacionados a limiares utilizados por algoritmos de qualidade de dados. Como discutido no Capítulo 4, tal especificação pode ser realizada a partir de regras BNF e representadas no SMQD por arquivos estruturados no formato XML ou JSON.

Geração automática de algoritmos de qualidade de dados. O principal objetivo do módulo *Algorithm Mapper* mostrado na arquitetura do SMQD proposta (Figura 4.1) consiste em processar os valores dos parâmetros dos DQSLAs recebidos pelo serviço e mapear, com base nestes valores, algoritmos de qualidade de dados a serem executados em um ambiente de computação em nuvem. Uma vez que o problema atribuído a este módulo não foi investigado neste trabalho, futuras investigações científicas são necessárias visando desenvolver soluções satisfatórias para este problema.

Privacidade no acesso aos dados pelo SMQD. Um problema não investigado neste trabalho consiste em como garantir a privacidade dos dados, armazenados em bancos de dados em nuvem, que são acessados pelo SMQD para executar algoritmos de qualidade de dados. Na prática, este pode ser um aspecto crucial para viabilizar a disponibilização de um serviço desta natureza em um cenário real, uma vez que muitos clientes possuem dados sigilosos ou privados (i.e., não é de interesse do usuário compartilhar os dados com usuários não autorizados). Por esta razão, futuros esforços são necessários para i) estender a arquitetura em alto nível do SMQD proposta (Figura 4.1) visando adicionar módulos responsáveis por garantir a privacidade dos dados acessados pelo serviço; ii) incorporar aos módulos adicionados no item i algoritmos disponíveis no estado da arte para execução de algoritmos de qualidade garantindo a privacidade dos dados; e iii) propor novas abordagens para executar algoritmos de qualidade mantendo a privacidade dos dados, levando em consideração o contexto específico do SMQD (por exemplo, o fato de a execução dos algoritmos estar associada a uma restrição de tempo).

Auxílio na estimativa de parâmetros de DQSLA. Diversos parâmetros presentes na estrutura de DQSLA proposta (tais como DQ_{dim}, M_{rules} e $\mathcal{P}$) requerem conhecimentos específicos sobre algoritmos de qualidade de dados e modelos de custo de penalidade para serem atribuídos corretamente e, por este motivo, necessitam do conhecimento de especialistas no domínio. Visando viabilizar a utilização de um SMQD para um usuário sem conhecimentos especializados em contratos de SLA e/ou qualidade de dados, podem ser realizadas futuras

pesquisas visando modelar módulos específicos para o SMQD no intuito de auxiliar usuários do serviço a atribuírem valores adequados para os parâmetros de DQSLA submetidos.

Bibliografia

[1] Michael Armbrust, Kristal Curtis, Tim Kraska, Armando Fox, Michael J Franklin, and David A Patterson. Piql: Success-tolerant query processing in the cloud. *Proceedings of the VLDB Endowment*, 5(3):181–192, 2011.

[2] Javed A Aslam, Ekaterina Pelekhov, and Daniela Rus. The star clustering algorithm for static and dynamic information organization. *J. Graph Algorithms Appl.*, 8:95–129, 2004.

[3] Elarbi Badidi. A cloud service broker for sla-based saas provisioning. In *Information Society (i-Society), 2013 International Conference on*, pages 61–66. IEEE, 2013.

[4] Donald P Ballou and Giri Kumar Tayi. Enhancing data quality in data warehouse environments. *Communications of the ACM*, 42(1):73–78, 1999.

[5] Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. Methodologies for data quality assessment and improvement. *ACM computing surveys (CSUR)*, 41(3):16, 2009.

[6] Carlo Batini and Monica Scannapieco. *Data Quality Dimensions*, pages 21–51. Springer International Publishing, Cham, 2016.

[7] Peter Bodík, Rean Griffith, Charles A Sutton, Armando Fox, Michael I Jordan, and David A Patterson. Statistical machine learning makes automatic control practical for internet datacenters. *HotCloud*, 9:12–12, 2009.

[8] Nicolas Bruno, Sapna Jain, and Jingren Zhou. Continuous cloud-scale query optimization and processing. *Proceedings of the VLDB Endowment*, 6(11):961–972, 2013.

[9] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment problems: revised reprint*. SIAM, 2012.

[10] Tao Chen and Rami Bahsoon. Symbiotic and sensitivity-aware architecture for globally-optimal benefit in self-adaptive cloud. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, Proceedings, Hyderabad, India, June 2-3, 2014*, pages 85–94, 2014.

[11] Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 2011.

[12] Peter Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science Business Media, 2012.

[13] Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555, 2012.

[14] William W Cohen and Jacob Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 475–480. ACM, 2002.

[15] Gianni Costa, Giuseppe Manco, and Riccardo Ortale. An incremental clustering scheme for data de-duplication. *Data Mining and Knowledge Discovery*, 20(1):152–187, 2010.

[16] Akash Das Sarma, Yeye He, and Surajit Chaudhuri. Clusterjoin: a similarity joins framework using map-reduce. *Proceedings of the VLDB Endowment*, 7(12):1059–1070, 2014.

[17] Timothy De Vries, Hui Ke, Sanjay Chawla, and Peter Christen. Robust record linkage blocking using suffix arrays. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 305–314. ACM, 2009.

[18] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 27–33. Ieee, 2010.

[19] Dimas Cassimiro do Nascimento Filho, Carlos Eduardo Santos Pires, and Campina Grande-Paraıba-PB-Brasil. A data quality-aware cloud service. In *Proceedings do Workshop de Teses e DissertaÃğÃţes do SimpÃşsio Brasileiro de Banco de Dados*, 2016.

[20] Uwe Draisbach and Felix Naumann. A generalization of blocking and windowing algorithms for duplicate detection. In *Data and Knowledge Engineering (ICDKE), 2011 International Conference on*, pages 18–24. IEEE, 2011.

[21] Uwe Draisbach, Felix Naumann, Szymon Szott, and Oliver Wonneberg. Adaptive windows for duplicate detection. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 1073–1083. IEEE, 2012.

[22] JG Enríquez, FJ Domínguez-Mayo, MJ Escalona, M Ross, and G Staples. Entity reconciliation in big data sources: a systematic mapping study. *Expert Systems with Applications*, 2017.

[23] Martin J Eppler and Peter Muenzenmayer. Measuring information quality in the web context: A survey of state-of-the-art instruments and an application methodology. In *IQ*, pages 187–196. Citeseer, 2002.

[24] Stefano Ferretti, Vittorio Ghini, Fabio Panzieri, Michele Pellegrini, and Elisa Turrini. Qos–aware clouds. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 321–328. IEEE, 2010.

[25] Jeffrey Fisher, Peter Christen, Qing Wang, and Erhard Rahm. A clustering-based framework to control block sizes for entity resolution. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 279–288. ACM, 2015.

[26] Nuwan Ganganath, Chi-Tsun Cheng, and K Tse Chi. Data clustering with cluster size constraints using a modified k-means algorithm. In *Cyber-Enabled Distributed*

Computing and Knowledge Discovery (CyberC), 2014 International Conference on, pages 158–161. IEEE, 2014.

[27] Saeed Ghanbari, Gokul Soundararajan, Jin Chen, and Cristiana Amza. Adaptive learning of metric correlations for temperature-aware database provisioning. In *Autonomic Computing, 2007. ICAC'07. Fourth International Conference on*, pages 26–26. IEEE, 2007.

[28] Demetrio Gomes Mestre and Carlos Eduardo Santos Pires. Improving load balancing for mapreduce-based entity matching. In *Computers and Communications (ISCC), 2013 IEEE Symposium on*, pages 000618–000624. IEEE, 2013.

[29] Luis Gravano, Panagiotis G Ipeirotis, Hosagrahar Visvesvaraya Jagadish, Nick Koudas, Shanmugauelayut Muthukrishnan, Divesh Srivastava, et al. Approximate string joins in a database (almost) for free. In *VLDB*, volume 1, pages 491–500, 2001.

[30] Anja Gruenheid, Xin Luna Dong, and Divesh Srivastava. Incremental record linkage. *Proceedings of the VLDB Endowment*, 7(9):697–708, 2014.

[31] Anja Gruenheid, Xin Luna Dong, and Divesh Srivastava. Incremental record linkage. *Proceedings of the VLDB Endowment*, 7(9):697–708, 2014.

[32] Anja Gruenheid, Besmira Nushi, Tim Kraska, Wolfgang Gatterbauer, and Donald Kossmann. Fault-tolerant entity resolution with the crowd. *arXiv preprint arXiv:1512.00537*, 2015.

[33] Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Renée J Miller. Framework for evaluating clustering algorithms in duplicate detection. *Proceedings of the VLDB Endowment*, 2(1):1282–1293, 2009.

[34] Taher Haveliwala, Aristides Gionis, and Piotr Indyk. Scalable techniques for clustering the web. 2000.

[35] Sue-Chen Hsueh, Ming-Yen Lin, and Yi-Chun Chiu. A load-balanced mapreduce algorithm for blocking-based entity-resolution with multiple keys. In *Proceedings of the Twelfth Australasian Symposium on Parallel and Distributed Computing-Volume 152*, pages 3–9. Australian Computer Society, Inc., 2014.

[36] Ye Hu, Johnny Wong, Gabriel Iszlai, and Marin Litoiu. Resource provisioning for cloud computing. In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, pages 101–111. IBM Corp., 2009.

[37] Ying Huo, Yi Zhuang, Jingjing Gu, Siru Ni, and Yu Xue. Discrete gbest-guided artificial bee colony algorithm for cloud service composition. *Applied Intelligence*, 42(4):661–678, 2015.

[38] Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. Autonomic resource provisioning for cloud-based software. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, Proceedings, Hyderabad, India, June 2-3, 2014*, pages 95–104, 2014.

[39] Matthew A Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[40] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th international conference on Autonomic computing*, pages 117–126. ACM, 2009.

[41] Ralph Kimball. *The data warehouse lifecycle toolkit*. John Wiley & Sons, 2008.

[42] Lars Kolb, Andreas Thor, and Erhard Rahm. Multi-pass sorted neighborhood blocking with mapreduce. *Computer Science-Research and Development*, 27(1):45–63, 2012.

[43] Hanna Kopcke and Erhard Rahm. Frameworks for entity matching: a comparison. *Data and Knowledge Engineering*, 69(2):197–210, 2010.

[44] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1-2):484–493, 2010.

[45] Yousri Kouki and Thomas Ledoux. Scaling: Sla-driven cloud auto-scaling. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 411–414. ACM, 2013.

[46] Stepan Kozak and Pavel Zezula. Efficiency and security in similarity cloud services. *Proceedings of the VLDB Endowment*, 6(12):1450–1455, 2013.

[47] Josep Ll. Berral, Ricard Gavaldà, and Jordi Torres. Empowering automatic data-center management with machine learning. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 170–172, New York, NY, USA, 2013. ACM.

[48] Michael Lones. Sean luke: essentials of metaheuristics. *Genetic Programming and Evolvable Machines*, 12(3):333–334, 2011.

[49] David Loshin. *The practitioner's guide to data quality improvement*. Elsevier, 2010.

[50] Mikko I Malinen and Pasi Fränti. Balanced k-means for clustering. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 32–41. Springer, 2014.

[51] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[52] Demetrio Gomes Mestre and Carlos Eduardo Pires. Efficient entity matching over multiple data sources with mapreduce. *Journal of Information and Data Management*, 5(1):40, 2014.

[53] Demetrio Gomes Mestre, Carlos Eduardo Pires, and Dimas C Nascimento. Adaptive sorted neighborhood blocking for entity matching with mapreduce. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 981–987. ACM, 2015.

[54] Demetrio Gomes Mestre, Carlos Eduardo Santos Pires, and Dimas Cassimiro Nascimento. Towards the efficient parallelization of multi-pass adaptive blocking for entity matching. *Journal of Parallel and Distributed Computing*, 101:27–40, 2017.

[55] Demetrio Gomes Mestre, Carlos Eduardo Santos Pires, Dimas Cassimiro Nascimento, Andreza Raquel Monteiro de Queiroz, Veruska Borges Santos, and Tiago Brasileiro Araujo. An efficient spark-based adaptive windowing for entity matching. *Journal of Systems and Software*, 128:1–10, 2017.

[56] Ryszard S Michalski, Jaime G Carbonell, and Tom M Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[57] Matthew Michelson and Craig A Knoblock. Learning blocking schemes for record linkage. In *AAAI*, pages 440–445, 2006.

[58] Dimas C Nascimento, Carlos Eduardo Pires, and Demetrio Mestre. Data quality monitoring of cloud databases based on data quality slas. In *Big-Data Analytics and Cloud Computing*, pages 3–20. Springer, 2015.

[59] Dimas C Nascimento, Carlos Eduardo Pires, and Demetrio Gomes Mestre. A data quality-aware cloud service based on metaheuristic and machine learning provisioning algorithms. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1696–1703. ACM, 2015.

[60] Dimas Cassimiro Nascimento, Carlos Eduardo Pires, and Demetrio Gomes Mestre. Applying machine learning techniques for scaling out data quality algorithms in cloud computing environments. *Applied Intelligence*, pages 1–19, 2016.

[61] Bernd Opitz, Timo Sztyler, Michael Jess, Florian Knip, Christian Bikar, Bernd Pfister, and Ansgar Scherp. On-the-fly entity resolution from distributed social media sources for mobile search and exploration. In *Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia*, pages 14–24. ACM, 2015.

[62] George Papadakis, Georgia Koutrika, Themis Palpanas, and Wolfgang Nejdl. Meta-blocking: Taking entity resolutionto the next level. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1946–1960, 2014.

[63] George Papadakis and Themis Palpanas. Blocking for large-scale entity resolution: Challenges, algorithms, and practical examples. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 1436–1439. IEEE, 2016.

[64] George Papadakis, George Papastefanatos, and Georgia Koutrika. Supervised meta-blocking. *Proceedings of the VLDB Endowment*, 7(14):1929–1940, 2014.

[65] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *Proceedings of the VLDB Endowment*, 9(9):684–695, 2016.

[66] K Hima Prasad, Tanveer A Faruquie, L Venkata Subramaniam, Mukesh Mohania, and Girish Venkatachaliah. Resource allocation and sla determination for large data processing services over cloud. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 522–529. IEEE, 2010.

[67] David Rebollo-Monedero, Marc Solé, Jordi Nin, and Jordi Forné. A modification of the k-means method for quasi-unsupervised learning. *Knowledge-Based Systems*, 37:176–185, 2013.

[68] M Brent Reynolds, Kenneth M Hopkinson, Mark E Oxley, and Barry E Mullins. Provisioning norm: An asymmetric quality measure for saas resource allocation. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 112–119. IEEE, 2011.

[69] El Kindi Rezig, Eduard C Dragut, Mourad Ouzzani, and Ahmed K Elmagarmid. Query-time record linkage and fusion over web databases. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 42–53. IEEE, 2015.

[70] Shazia Sadiq. *Handbook of data quality*. Springer, 2013.

[71] Sadiq M Sait and Kh Shahzada Shahid. Engineering simulated evolution for virtual machine assignment problem. *Applied Intelligence*, pages 1–12, 2015.

[72] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–278. ACM, 2002.

[73] Maxim Schnjakin, Rehab Alnemr, and Christoph Meinel. Contract-based cloud architecture. In *Proceedings of the second international workshop on Cloud data management*, pages 33–40. ACM, 2010.

[74] Piyush Shivam, Shivnath Babu, and Jeffrey S Chase. Learning application models for utility resource planning. In *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, pages 255–264. IEEE, 2006.

[75] Fatimah Sidi, PH Shariat Panahy, Lilly Suriani Affendey, Marzanah A Jabar, Haidi Ibrahim, and Aouache Mustapha. Data quality: A survey of data quality dimensions. In *Information Retrieval & Knowledge Management (CAMP), 2012 International Conference on*, pages 300–304. IEEE, 2012.

[76] Manoel Siqueira, José Maria Monteiro, Angelo Brayner, Flávio RC Sousa, and Javam C Machado. Interaction-aware data management in the cloud. *Journal of Information and Data Management*, 4(3):311, 2013.

[77] James Skene, D Davide Lamanna, and Wolfgang Emmerich. Precise service level agreements. In *Proceedings of the 26th International Conference on Software Engineering*, pages 179–188. IEEE Computer Society, 2004.

[78] Shan Suthaharan. Big data classification: Problems and challenges in network intrusion prediction with machine learning. *ACM SIGMETRICS Performance Evaluation Review*, 41(4):70–73, 2014.

[79] Marcello Trovati, Richard Hill, Ashiq Anjum, Shao Ying Zhu, and Lu Liu. *Big-Data Analytics and Cloud Computing: Theory, Algorithms and Applications*. Springer, 2016.

[80] Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, and Pawan Goyal. Dynamic provisioning of multi-tier internet applications. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 217–228. IEEE, 2005.

[81] Dinusha Vatsalan, Peter Christen, and Vassilios S Verykios. A taxonomy of privacy-preserving record linkage techniques. *Information Systems*, 38(6):946–969, 2013.

[82] Nele Verbiest. *Fuzzy rough and evolutionary approaches to instance selection*. PhD thesis, Ghent University, 2014.

[83] Rares Vernica, Michael J Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 495–506. ACM, 2010.

[84] Vassilios S Verykios, Alexandros Karakasidis, and Vassilios K Mitrogiannis. Privacy preserving record linkage approaches. *International Journal of Data Mining, Modelling and Management*, 1(2):206–221, 2009.

[85] Jinglian Wang, Bin Gong, Hong Liu, and Shaohui Li. Multidisciplinary approaches to artificial swarm intelligence for heterogeneous computing and cloud scheduling. *Applied Intelligence*, pages 1–14, 2015.

[86] Steven Euijong Whang and Hector Garcia-Molina. Incremental entity resolution on rules and data. *The VLDB Journal*, 23(1):77–102, 2014.

[87] Steven Euijong Whang, David Menestrina, Georgia Koutrika, Martin Theobald, and Hector Garcia-Molina. Entity resolution with iterative blocking. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 219–232. ACM, 2009.

[88] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[89] Ian H Witten, Eibe Frank, Leonard E Trigg, Mark A Hall, Geoffrey Holmes, and Sally Jo Cunningham. Weka: Practical machine learning tools and techniques with java implementations. 1999.

[90] Linlin Wu, Saurabh Kumar Garg, and Rajkumar Buyya. Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 195–204. IEEE, 2011.

[91] Linlin Wu, Saurabh Kumar Garg, and Rajkumar Buyya. Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 195–204. IEEE, 2011.

[92] Pengcheng Xiong, Yun Chi, Shenghuo Zhu, Hyun Jin Moon, Calton Pu, and Hakan Hacigumus. Intelligent management of virtualized resources for database systems in cloud

environment. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 87–98. IEEE, 2011.

[93] Mohamed ZaÃŕt and Hammou Messatfa. Data mining a comparative study of clustering methods. *Future Generation Computer Systems*, 13(2):149 – 159, 1997.

[94] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[95] Shunzhi Zhu, Dingding Wang, and Tao Li. Data clustering with size constraints. *Knowledge-Based Systems*, 23(8):883–889, 2010.

Apêndice A

Tabelas de Símbolos

Na Tabela A.1, são apresentados os símbolos e abreviações utilizados na notação associada aos conceitos de computação em nuvem adotados neste trabalho. Por sua vez, na Tabela A.2, são apresentados os símbolos e abreviações utilizados na notação associada aos conceitos de deduplicação incremental de dados no contexto de classificação coletiva utilizados neste trabalho. Por fim, na Tabela A.3, são apresentados os símbolos e abreviações utilizados na notação associada aos conceitos de controle de tamanho de blocos empregados neste trabalho.

Tabela A.1: Notação adotada (Computação em Nuvem).

Notação	Descrição
$C\langle t_i, t_f\rangle$	conjunto de clientes do SMQD durante o intervalo de tempo (t_i, t_f)
$C\$(\tau, \mathcal{C}l(\tau))$	o custo total ($VMC\$ + PenaltyC\$$) de executar a tarefa τ usando a classe de configuração $\mathcal{C}l(\tau)$
$\mathcal{C}_{conf}$	conjunto das classes de configurações disponíveis para determinar a configuração de um cluster ($\mathcal{C}_{conf} = N \times \gamma$)
$\mathcal{C}l_i$	uma classe de configuração de cluster de índice igual a i
$\mathcal{C}l(\tau)$	a classe de configuração adotada para executar a tarefa τ
$\mathcal{C}l_{opt}(\tau)$	a classe de configuração ótima para executar a tarefa τ
$\mathcal{D}$	uma base de dados
$DT(\tau)$	a diferença entre o tempo de execução ($T_{exec}(\tau)$) e o tempo de restrição ($T_{res}(\tau)$) da tarefa τ
$Alloc(\tau)$	conjunto de máquinas virtuais utilizadas para executar a tarefa τ
N	conjunto cujos elementos representam a quantidade de máquinas virtuais que podem ser alocadas
$overThr$	um limiar (positivo) que determina a classificação de alocações que produzem super provisão de recursos
$PenaltyC\$(\tau, \mathcal{C}l(\tau))$	valor total de penalidade associada à execução da tarefa τ utilizando a classe de configuração de cluster $\mathcal{C}l(\tau)$
$Price(\mathcal{C}l_i)$	preço associado à classe de configuração de cluster $\mathcal{C}l_i \in \mathcal{C}l_{conf}$
$S\langle t_i, t_f\rangle$	conjunto de SLAs de qualidade de dados que são válidos durante o intervalo de tempo $\langle t_i, t_f\rangle$
$ServC\$^{C\langle t_i,t_f\rangle}_{prov\_alg}$	custos do serviço (custos de infraestrutura + penalidades) para atender um conjunto C de clientes, durante o intervalo de tempo (t_i, t_f), usando o algoritmo de provisionamento $prov\_alg$
τ	uma tarefa de qualidade de dados
$\mathcal{T}$	um conjunto de tarefas de qualidade de dados
$T_{exec}(\tau)$	tempo de execução da tarefa τ
$T_{res}(\tau)$	tempo de restrição associado à execução da tarefa τ
TD	base de treinamento
$TD_{\lhd}$	base de treinamento filtrada ($sliced$)
$VMC\$(\tau, \mathcal{C}l(\tau))$	custo de infraestrutura associado à execução da tarefa τ utilizando a classe de configuração $\mathcal{C}l(\tau)$
ϕ	parâmetro de ajuste de classes de configuração (usado pelas heurísticas $Tunable\ Allocation$ e $Adaptive\ Allocation$)
δ	valor utilizado para adaptar o parâmetro de ajuste de classes de configuração (usado pela heurística $Adaptive\ Allocation$)
$\Lambda(\tau)$	a quantidade estimada de comparações (entre entidades) produzida pela tarefa τ
$\eta(\tau)$	o custo (tempo de execução) de comparação entre duas entidades na execução da tarefa τ
$\alpha(\tau)$	a penalidade fixa associada à tarefa τ
$\beta(\tau)$	a taxa de penalidade associada à tarefa τ
γ	conjunto de configurações de máquinas virtuais (quantidade de vCPUs)

Tabela A.2: Notação adotada (deduplicação incremental de dados no contexto de classificação coletiva).

Notação	Descrição
$C\langle t_i, t_f\rangle$	conjunto de clientes do SMQD durante o intervalo de tempo (t_i, t_f)
G	grafo de similaridade
ΔG	conjunto de incrementos (inserção, remoção ou atualização) sobre o grafo de similaridade G
δ	uma operação individual (inserção, remoção ou atualização) sobre um grafo de similaridade
θ	limiar de similar mínimo utilizado para definir quais arestas são adicionados no grafo de similaridade
$\mathcal{L}_G$	um conjunto de agrupamentos sobre os vértices de G
$CC(\mathcal{L}_G)$	valor da métrica $correlation\ clsutering$ associada ao resultado de agrupamento $\mathcal{L}_G$
$T(\Delta G)$	Componente Conectado (CONNECTED COMPONENT)
$\hat{T}(\Delta G)$	Componente Ligeiramente Conectado (MONOTONE CONNECTED COMPONENT)
$\bar{T}(\Delta G)$	Componente Diretamente Conectado (DIRECTLY CONNECTED COMPONENT)
$\dot{T}(\Delta G)$	método utilizado para selecionar o componente de cobertura de um conjunto de incrementos sobre o grafo G (i.e., o subgrafo afetado pelo conjunto de incrementos ΔG)
C	um agrupamento de vértices
f	um método de deduplicação incremental de dados
F	um método de deduplicação de dados (em $batch$)
$o(\mathcal{L}_G)$	função objetivo utilizada para calcular a penalidade associada ao resultado de agrupamento $\mathcal{L}_G$
$\mathcal{L}_G^{opt}$	agrupamento ótimo (i.e., que produz a penalidade mínima) sobre o grafo G
$\mathcal{G}$	conjunto de agrupamentos de referência (gabarito de entidades duplicadas)
CRc	cobertura do resultado de agrupamento
$wCPr$	precisão ponderada do resultado de agrupamento
$PCPr$	precisão penalizada do resultado de agrupamento
CPr	precisão do resultado de agrupamento
$accPrLoss$	perda de precisão acumulada produzida por um método de DID
$accRcLoss$	perda de cobertura acumulada produzida por um método de DID
$dgEfficacySt$	grau de estabilidade de eficácia de um método de DID
$dgEfficiencySt$	grau de estabilidade de eficiência de um método de DID
$accExecTime$	tempo de execução acumulado de um método de DID
$relEfficiency$	eficiência relativa entre dois métodos de DID
$\mathcal{F}$	filtro de componente de cobertura
μ	a porcentagem (0, 1) que determina a quantidade de agrupamentos de $\mathcal{L}_G$ a serem selecionados por um filtro de componente de cobertura
$\mathcal{C}_{alg}$	algoritmo de agrupamento automático

Tabela A.3: Notação adotada (controle do tamanho de blocos).

Notação	Descrição
λ	porcentagem que define o alcance das heurísticas de poda
WS	esquema de pontuação de coocorrência
D	base de dados
B	resultado de blocagem (i.e., conjunto de blocos)
b	bloco
$\mathcal{B}$	conjunto de blocagens (i.e., um conjunto de conjuntos de blocos)
F_{key}	conjunto de funções de chave de bloco
f	função de chave de bloco
$\|\|\mathcal{B}\|\|$	cardinalidade agregada do conjunto de blocagens $\mathcal{B}$
PC	cobertura do resultado de blocagem (*pair completeness*)
PQ	precisão do resultado de blocagem (*pair quality*)
RR	taxa de redução (*reduction ratio*)
S_{min}	tamanho mínimo dos blocos
S_{max}	tamanho máximo dos blocos
$ARCS$	esquema de pontuação de coocorrência *Aggregated Reciprocal Common Scheme*
$ECBS$	esquema de pontuação de coocorrência *Enhanced Common Blocking Scheme*
$BlockCooScore$	pontuação de coocorrência agregada do bloco
$EntityCooScore$	pontuação de coocorrência agregada da entidade
B^-	conjunto de blocos pequenos (i.e., $\forall b \in B^- : \|b\| < S_{min}$)
B^+	conjunto de blocos grandes (i.e., $\forall b \in B^+ : \|b\| > S_{max}$)
B^*	conjunto de blocos com tamanho ideal (i.e., $\forall b \in B^* : S_{min} \leq \|b\| \leq S_{max}$)

Apêndice B

Semântica das Operações do Algoritmo Greedy

Neste apêndice, são apresentadas as semânticas das operações realizadas pelo algoritmo $greedy$ [31] (Algoritmo 1).

Merge. Seja C um agrupamento em $\mathbf{Q}^c$, é avaliada a possibilidade de unir o agrupamento C com outros agrupamentos visando produzir um agrupamento de melhor qualidade (ou seja, menor valor para uma função objetivo). Para realizar esta tarefa em tempo polinomial, apenas a possibilidade de unir pares de agrupamentos é considerada. O algoritmo $Merge$ funciona da seguinte maneira:

1. Para cada agrupamento C' vizinho de C, avalie se unir C e C' produz um melhor agrupamento;

2. Caso algum agrupamento C' satisfaça a avaliação do item 1: i) una C e C'; ii) Adicione $C \cup C'$ em $\mathbf{Q}^c$; e iii) remova C' de $\mathbf{Q}^c$ se $C' \in \mathbf{Q}^c$.

Split. Seja C um agrupamento em $\mathbf{Q}^c$, é avaliado se dividir o agrupamento C em vários outros agrupamentos produz melhores agrupamentos. O algoritmo $Split$ procede da seguinte maneira:

1. Para cada nó $v \in C$, avalie se atualizar v para fora de C como um novo agrupamento produz agrupamentos melhores;

2. Caso exista um vértice v que satisfaça a condição 1, crie um novo agrupamento $C' = \{v\}$ e conduza aos passos 3-4;

3. Para cada vértice v' restante em C, avalie se mover v' de C para C' produz melhores agrupamentos. Se for o caso, mova v' para C' e repita o passo 3;

4. Adicione C e C' em $\mathbf{Q}^c$ casos estes agrupamentos estejam conectados a outros agrupamentos.

Move. Seja C um agrupamento em $\mathbf{Q}^c$, é avaliado se mover algum vértice de C para outros agrupamentos produz melhores agrupamentos. O algoritmo $Move$ procede da seguinte maneira:

1. Para cada agrupamento vizinho C' de C, realize os passos 2-3;

2. Para cada vértice $v \in C$ que está conectado a C' e para cada vértice $v \in C'$ que está conectado a C, avalie se mover v para o outro agrupamento produz agrupamentos melhores. Caso algum vértice v satisfaça essa condição, mova v para o outro agrupamento;

3. Repita o passo 2 até que não existam mais vértices para serem movidos. Então, i) adicione os dois novos agrupamentos em $\mathbf{Q}^c$ e desempilhe C' se $C' \in \mathbf{Q}^c$.

Apêndice C

Lista de Algoritmos Propostos

C.1 Algoritmos para Adaptação de Classe de Configuração

A heurística *Binary Tweak Solution*, apresentada no Algoritmo 7, realiza um ajuste baseado em uma busca binária sobre o conjunto de classes de configuração disponíveis (cujos limites variam de 1 até $\mathcal{C}_{conf}$), partindo do índice da classe de configuração de entrada (Cl_i). Uma ilustração do funcionamento da heurística *Binary Tweak Solution* é apresentada na Figura C.1.

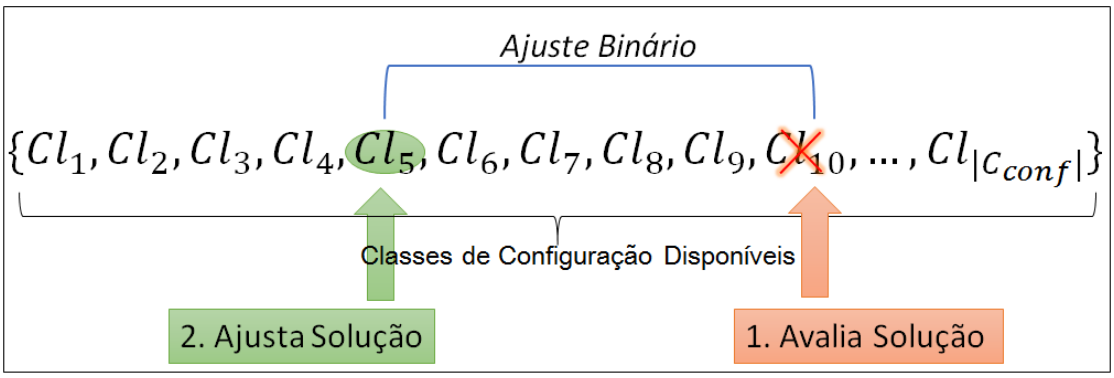

Figura C.1: Ilustração do funcionamento do algoritmo *Binary Tweak Solution*.

Por sua vez, a heurística *Tunable Tweak Solution* (Algoritmo 8) realiza um ajuste na classe de configuração de entrada (Cl_i) com base no valor de um parâmetro de entrada (F). A proposta desta heurística é ser utilizada em conjunto com técnicas baseadas em aprendizado de máquina, uma vez que é esperado que estas técnicas possam realizar boas estimativas da classe de configuração inicial (linha 4 do Algoritmo 2) para uma tarefa de deduplicação

Algoritmo 7: Binary Tweak Solution

input : Cl_i: uma classe de configuração de entrada

τ: uma tarefa de deduplicação de dados

L: o índice da última classe de configuração disponível ($L = |\mathcal{C}_{conf}|$)

output: Cl_o: uma classe de configuração de saída

```
1  begin
2      if (T_exec(τ) > T_res(τ)) then
3          // scaling up
4          o ⟵ ⌈(i + L)/2⌉
5          return Cl_o
6      else if (T_exec(τ) < T_res(τ)) then
7          // scaling down
8          o ⟵ ⌊(1 + i)/2⌋
9          return Cl_o
10     return Cl_i
```

de dados. Desse modo, é também esperado que apenas pequenos ajustes sejam necessários sobre as estimativas iniciais. Uma ilustração do funcionamento da heurística *Tunable Tweak Solution* é apresentada na Figura C.2.

C.2 Algoritmos para Estimativa de Classe de Configuração Inicial

Nesta seção são apresentados dois algoritmos de aprendizado de máquina: $KNearestNeighbors$ e *Seleção de Protótipos*. O primeiro algoritmo é apresentado no Algoritmo 9. Por sua vez, o segundo algoritmo visa gerar uma nova classe de configuração com base na agregação dos valores dos índices de um conjunto de classes de configuração de entrada. A algoritmo *Prototype Selection* é representado no Algoritmo 10. Neste trabalho, duas formas de agregação são utilizadas: i) **Median**: a mediana dos índices das K classes de configuração dos vizinhos mais próximos da tarefa τ, contidos na base de

Algoritmo 8: Tunable Tweak Solution

input : Cl_i: uma classe de configuração de entrada

τ: uma tarefa de deduplicação de dados

L: o índice da última classe de configuração disponível ($L = |\mathcal{C}_{conf}|$)

F: um fator para ajustar a classe de configuração de entrada

output: Cl_o: uma classe de configuração de saída

```
1  begin
2      if (T_exec(τ) > T_res(τ)) then
3          // scaling up
4          o ⟵ i + ⌈L/F⌉
5          return Cl_o
6      else if (T_exec(τ) < T_res(τ)) then
7          // scaling down
8          o ⟵ i − ⌊L/F⌋
9          return Cl_o
10     return Cl_i
```

treinamento; e ii) **wGeomMean**: a média geométrica ponderada, mostrada na Eq. (C.1c), dos índices das classes de configuração associadas aos K vizinhos mais próximos da tarefa τ, contidos na base de treinamento. Neste trabalho, foi utilizado o valor i^2 como parâmetro do peso da média geométrica ponderada, como mostrado nas equações (C.1a) e (C.1b).

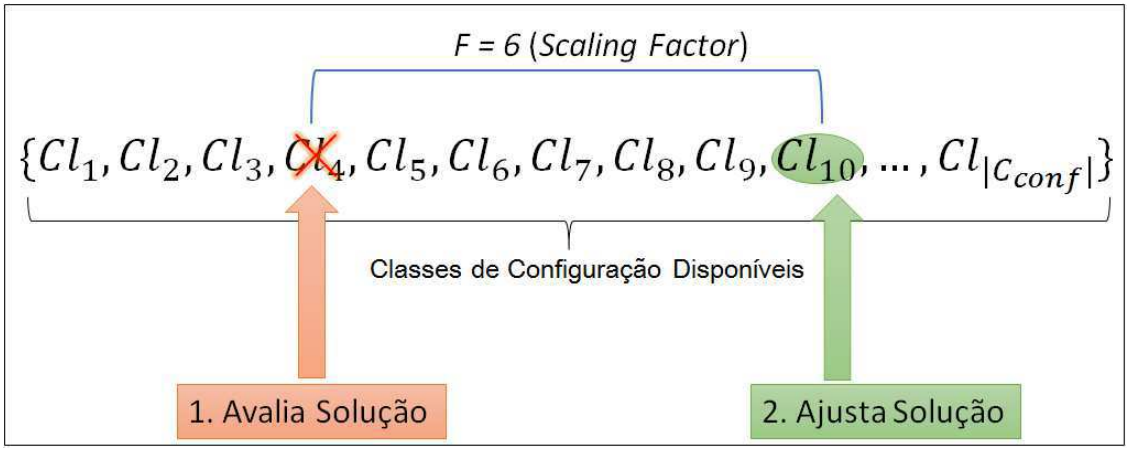

Figura C.2: Ilustração do funcionamento do algoritmo *Tunable Tweak Solution*.

Algoritmo 9: K Nearest Neighbors

input : τ: uma tarefa de deduplicação de dados

TD: uma base de treinamento

K: quantidade de vizinhos

$dist$: uma função de similaridade

output: Cl_o: uma classe de configuração de saída

begin

  return $MostFrequentClass\Big(Top\ K\Big(\arg\min_{t \in TD} \sum_{a \in \mathcal{A}_{dedup}} dist_a(t, \tau)\Big)\Big)$

$$wGeomMean(\mathbf{x}) = \left(\prod_{i=1}^{n} \mathbf{x}_i^{w_i}\right)^{\frac{1}{\sum_{i=i}^{n} w_i}} =$$

$$= \exp\left(\frac{\sum_{i=1}^{n} w_i \ln \mathbf{x}_i}{\sum_{i=1}^{n} w_i}\right) \tag{C.1a}$$

$$w_i = i^2 \tag{C.1b}$$

$$wGeomMean(\mathbf{x}) = \exp\left(\frac{\sum_{i=1}^{n} i^2 \ln \mathbf{x}_i}{\sum_{i=1}^{n} i^2}\right) \tag{C.1c}$$

Algoritmo 10: Select Prototype

input : τ: uma tarefa de deduplicação de dados

TD: uma base de treinamento

k: quantidade de vizinhos

$aggMode$: uma estratégia de agregação de valores

$dist$: uma função de similaridade

output: Cl_o: uma classe de configuração de saída

```
1  begin
2      kNearestNeighbors[] ⟵ kNearestNeighborsClasses(τ, TD, dist, k)
3      if (aggMode = Median) then
4          // ascendant sorting based on the indexes of the
               classes
5          Sort(kNearestNeighbors)
6          MedianIdx ⟵ ⌊K/2⌋
7          return kNearestNeighbors[MedianIdx]
8      else if (aggMode = wGeoMean) then
9          // ascendant sorting based on the proximity of the
               neighbors to τ
10         Sort(kNearestNeighbors)
11         NeighborsIndexes ⟵ ExtractIndexes(kNearestNeighbors)
               // using Eq. C.1c
12         o ⟵ wGeomMean(NeighborsIndexes)
13         return Cl_o
```

C.3 Heurísticas para Seleção de Algoritmos de Provisionamento e Ajuste de Classe de Configuração

C.3.1 Heurística *Probabilistic Best Performing Allocation*

A heurística *Probabilistic Best Performing Allocation*, apresentada no Algoritmo 11, tenta superar as desvantagens produzidas por ambos os valores $\{0, 1\}$ do parâmetro $optInc$ no Algoritmo 4. Para tal, ao invés de associar contadores aos algoritmos de provisionamento, o Algoritmo 11 associa probabilidades aos mesmos. A heurística *Probabilistic Best Performing Allocation* atribui inicialmente uma probabilidade uniforme, com o valor de $\frac{1}{|algSet|}$, para todos os algoritmos no conjunto $algSet$ e estas probabilidades são armazenadas no mapa $probMap$ (linhas 2-4). Em seguida, é realizada uma seleção probabilística de um algoritmo de provisionamento, entre os algoritmos no conjunto $algSet$, levando em consideração as probabilidades armazenadas no mapa $probMap$ (linha 5). Então, o algoritmo de provisionamento escolhido ($chosenAlg$) é aplicado para estimar a classe de configuração inicial da tarefa de deduplicação de dados τ (linha 6), a tarefa τ é executada (linha 7) e a base de treinamento é atualizada utilizando os dados da execução de τ (linha 8).

Prosseguindo, se a estimativa realizada pelo algoritmo de provisionamento $chosenAlg$ produzir uma alocação classificada como super provisão ou sub provisão (linha 9), então o valor $\frac{1}{|algSet|+1}$ é redistribuído entre as probabilidades atribuídas aos algoritmos pertencentes ao conjunto $(algSet \backslash chosenAlg)$ (linhas 10-11). Note que a probabilidade não é distribuída de maneira uniforme entre os algoritmos. Para cada algoritmo de provisionamento $mlAlg$ no conjunto $(algSet \backslash chosenAlg)$, sua probabilidade é aumentada de acordo com sua porção relativa do valor $\sum_{alg \in (algSet \backslash chosenAlg)} probMap[alg]$, ou seja, $\frac{probMap(mlAlg)}{1 - probMap[chosenAlg]}$. Desse modo, quanto maior a probabilidade associada a um algoritmo contido em $(algSet \backslash chosenAlg)$, maior será o aumento de sua probabilidade. Por sua vez, a probabilidade associada ao algoritmo $chosenAlg$ é diminuída na mesma quantidade $\frac{1}{|algSet|+1}$ (linha 12), uma vez que é necessário respeitar a restrição geral: $\sum_{alg \in algSet} probMap[alg] = 1$. Além disso, é importante notar que a probabilidade associada a qualquer algoritmo $mlAlg$, contido em $algSet$, só pode ser aumentada (linha 11 ou linha 14) ou diminuída (linha 12 ou linha 16) se $probMap[mlAlg] < (1 - \frac{1}{|algSet|+1})$ ou $probMap[mlAlg] \geq$

Algoritmo 11: Probabilistic Best Performing Allocation

input : τ: uma tarefa de deduplicação de dados

$algSet$: um conjunto de algoritmos de aprendizado de máquina

$probMap$: um mapa contendo uma probabilidade associada a cada algoritmo em $algSet$

$overThr$: um limiar (positivo) utilizado para avaliar alocações classificadas como super provisão

$(TD, \mathcal{A}_{dedup} \cup \{\mathcal{C}l\})$: um sistema de decisão para tarefas de deduplicação de dados

1 **begin**

2 **foreach** *mlAlg in algSet* **do**

3 **if** $(probMap[mlAlg] = Nil)$ **then**

4 $probMap[mlAlg] \leftarrow \frac{1}{|algSet|}$

5 $chosenAlg \leftarrow ProbabilisticSelection(probMap)$

6 $\mathcal{C}l(\tau) \leftarrow\ chosenAlg((TD, \mathcal{A}_{dedup} \cup \{\mathcal{C}l\}), \tau)$

7 $allocate(\tau, \mathcal{C}l(\tau))$

8 $TD \leftarrow TD \cup \langle \Lambda(\tau), \eta(\tau), T_{exec}(\tau), \mathcal{C}l(\tau) \rangle$

9 **if** $(T_{exec}(\tau) > T_{res}(\tau)$ **or** $(T_{res}(\tau) - T_{exec}(\tau)) > overThr)$ **then**

10 **foreach** *mlAlg in (algSet\chosenAlg)* **do**

11 $probMap[mlAlg] \leftarrow$
$probMap[mlAlg] + \left(\frac{probMap[mlAlg]}{1 - probMap[chosenAlg]} \times \frac{1}{(|algSet|+1)}\right)$

12 $probMap[chosenAlg] \leftarrow probMap[chosenAlg] - \frac{1}{|algSet|+1}$

13 **else**

14 **foreach** *mlAlg in (algSet\chosenAlg)* **do**

15 $probMap[mlAlg] \leftarrow$
$probMap[mlAlg] - \left(\frac{probMap[mlAlg]}{1 - probMap[chosenAlg]} \times \frac{1}{(|algSet|+1)}\right)$

16 $probMap[chosenAlg] \leftarrow probMap[chosenAlg] + \frac{1}{|algSet|+1}$

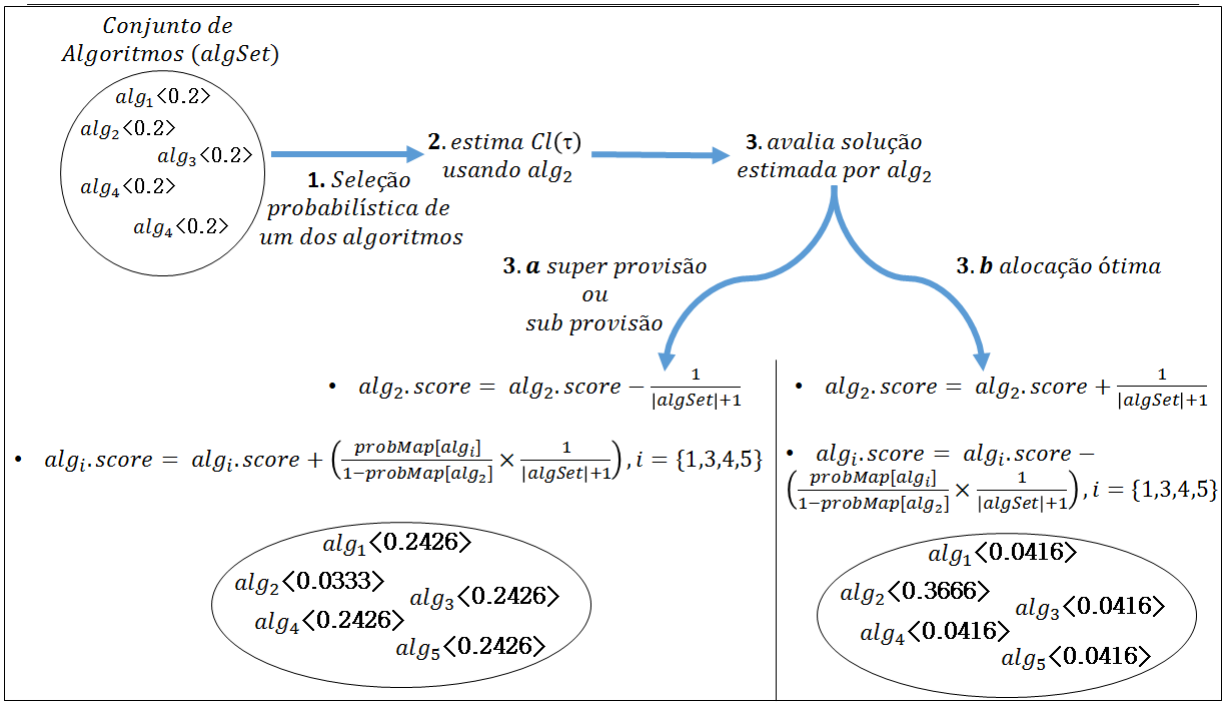

Figura C.3: Ilustração do funcionamento do algoritmo *Probabilistic Best Performing Allocation*.

$(\frac{probMap[mlAlg]}{1-probMap[chosenAlg]} \times \frac{1}{(|algSet|+1)})$, respectivamente. Estas restrições (as quais são omitidas no Algoritmo 11 para simplificar sua representação) são importantes para evitar a atribuição de valores maiores que 1 ou menores que 0 aos algoritmos de provisionamento.

Por fim, se a estimativa realizada pelo algoritmo de provisionamento $chosenAlg$ produzir uma alocação ótima (ou próxima de ótima) (linha 13), então sua probabilidade é aumentada em $\frac{1}{|algSet|+1}$ (linha 16). Por sua vez, a probabilidade associada a cada algoritmo $mlAg$ pertencente ao conjunto $(algSet \backslash chosenAlg)$ é diminuída de acordo com suas respectivas porções do valor $\sum_{alg \in algSet \backslash chosenAlg} probMap[alg]$ (linhas 14-15), como explicado anteriormente. No Exemplo C.3.1, é detalhado um cenário que explica a maneira como as probabilidades associadas aos algoritmos de provisionamento na heurística *Probabilistic Best Performing Allocation* são atribuídas e redistribuídas. Uma ilustração do funcionamento da heurística *Probabilistic Best Performing Allocation* é apresentada na Figura C.3.

Exemplo C.3.1 *Sejam* $\mathcal{T} = \{\tau_1, \tau_2\}$ *um conjunto de tarefas de deduplicação de dados a serem processadas por um SMQD e* $algSet = \{alg_1, alg_2, alg_3, alg_4, alg_5\}$ *um conjunto de algoritmos de provisionamento que estão disponíveis para estimar classes*

de configuração em um SMQD. Inicialmente, uma probabilidade uniforme ($\frac{1}{5}$) é atribuída a cada algoritmo de provisionamento pertencente ao conjunto $algSet$ e estas probabilidades são armazenadas no mapa $probMap$. O mapa $probMap$ é representado por um conjunto de tuplas na forma: $\langle alg\_name, alg\_prob \rangle$, tal que alg_name identifica o algoritmo de provisionamento e alg_prob é a probabilidade associada ao algoritmo de provisionamento. Desse modo, a configuração inicial de $probMap$ é igual a $probMap = \{\langle alg_1, 0.2\rangle, \langle alg_2, 0.2\rangle, \langle alg_3, 0.2\rangle, \langle alg_4, 0.2\rangle, \langle alg_5, 0.2\rangle\}$.

Então, a heurística Probabilistic Best Performing Allocation realiza uma seleção probabilística (linha 5) de um algoritmo de provisionamento, dentre as opções disponíveis no conjunto $probMap$, no intuito de estimar $\mathcal{C}l(\tau_1)$ (linha 6). Caso o algoritmo alg_2 seja selecionado e o mesmo produza uma alocação classificada como sub provisão ou super provisão (ou seja, a expressão na linha 9 é avaliada como verdadeira), então uma porção da probabilidade associada ao algoritmo alg_2 é redistribuída entre as probabilidades associadas aos demais algoritmos em $algset \backslash alg_2$, ou seja, ($\{alg_1, alg_3, alg_4, alg_5\}$). Esta redistribuição é realizada aumentando a probabilidade de cada algoritmo $provAlg$ contido em $algset \backslash alg_2$ em ($\frac{probMap[provAlg]}{1 - probMap[alg_2]} \times \frac{1}{(|algSet|+1)}$) e diminuindo $probMap[alg_2]$ em $\frac{1}{|algSet|+1} = \frac{1}{6}$. Logo, o mapa $probMap$ adquire os seguintes valores: $probMap = \{\langle alg_1, 0.2416\rangle, \langle alg_2, 0.0333\rangle, \langle alg_3, 0.2416\rangle, \langle alg_4, 0.2416\rangle, \langle alg_5, 0.2416\rangle\}$.

Em seguida, a heurística Probabilistic Best Performing Allocation realiza outra seleção probabilística (linha 5) de um algoritmo de provisionamento, dentre as opções disponíveis no conjunto $probMap$, no intuito de estimar $\mathcal{C}l(\tau_2)$ (linha 6). Caso o algoritmo alg_4 seja selecionado e o mesmo não produza uma alocação classificada como sub provisão ou super provisão (ou seja, a expressão na linha 9 é avaliada como falsa), então uma porção das probabilidades associadas aos algoritmos em $algset \backslash alg_4$ (ou seja, $\{alg_1, alg_2, alg_3, alg_5\}$) é redistribuída para a probabilidade associada a alg_4. Esta redistribuição é realizada reduzindo a probabilidade associada a cada algoritmo $provAlg$ em $algset \backslash alg_4$ (ou seja, $\{alg_1, alg_2, alg_3, alg_5\}$) em ($\frac{probMap[provAlg]}{1 - probMap[alg_4]} \times \frac{1}{(|algSet|+1)}$) e aumentando o valor de $probMap[alg_4]$ em $\frac{1}{|algSet|+1} = \frac{1}{6}$. Portanto, o mapa $probMap$ adquire os seguintes valores: $probMap = \{\langle alg_1, 0.1886\rangle, \langle alg_2, 0.026\rangle, \langle alg_3, 0.1886\rangle, \langle alg_4, 0.4082\rangle, \langle alg_5, 0.1886\rangle\}$.

C.3.2 Heurística *Tunable Allocation*

A principal desvantagem das heurísticas *Best Performing Allocation* e *Probabilistic Best Performing Allocation* consiste no fato de que a efetividade destas heurísticas, em relação à capacidade de reduzir custos de um SMQD, depende completamente da habilidade dos algoritmos de provisionamento em estimar classes de configuração ótimas (ou próximas de ótimas) para executar tarefas de deduplicação de dados. Uma vez que, na prática, os algoritmos de provisionamento podem produzir estimativas de classe de configuração não ideais, é também importante explorar heurísticas que possam ajustar as classes de configuração estimadas pelos algoritmos de provisionamento. Neste sentido, a heurística *Tunable Allocation*, representada no Algoritmo 12, visa ajustar as classes de configuração estimadas por um algoritmo de provisionamento dependendo da quantidade de alocações passadas realizadas pelo algoritmo que foram classificadas como sub provisão ou super provisão.

A heurística *Tunable Allocation* funciona da seguinte maneira. Dois mapas (*underMap* e *overMap*) são inicializadas no intuito de contar a quantidade de estimativas de classe de configuração já realizadas pelo algoritmo ($mlAlg$) de provisionamento que foram classificadas como sub provisão e super provisão, respectivamente. Inicialmente, é atribuído o valor 0 a ambos os mapas (linhas 2-5). Em seguida, o algoritmo ($mlAlg$) é empregado para estimar $\mathcal{C}l(\tau)$ (linha 6). Prosseguindo, se as estimativas passadas realizadas pelo algoritmo $mlAlg$ foram classificadas mais vezes como sub provisão do que como super provisão (linha 7), então o índice da classe de configuração estimada para executar τ é ajustado para ($Index(\mathcal{C}l(\tau)) + \phi$) (linhas 9-10). Caso contrário (ou seja, a expressão mostrada na linha 11 é avaliada como verdadeira), então o índice da classe de configuração estimada para executar τ é ajustado para ($Index(\mathcal{C}l(\tau)) - \phi$) (linhas 13-14).

Em seguida, a tarefa τ é executada (linha 15) (possivelmente utilizando a classe de configuração ajustada $\mathcal{C}l_d$ ou $\mathcal{C}l_u$), a base de treinamento é atualizada (linha 16) e o valor do mapa $underMap$ (linha 18) ou $overMap$ (linha 20) pode ser incrementado em 1, caso a classe de configuração estimada para executar τ seja classificada como sub provisão (linha 17) ou super provisão (linha 19). Uma ilustração do funcionamento da heurística *Tunable Allocation* é apresentada na Figura C.4.

Note que a heurística *Tunable Allocation* é aplicada para ajustar as classes de configuração estimadas por um único algoritmo de provisionamento, ao invés de escolher (e pos-

Algoritmo 12: Tunable Allocation

input : τ: uma tarefa de deduplicação de dados

$mlAlg$: um algoritmo de aprendizado de máquina

$underMap$: um mapa utilizado para contar a quantidade de alocações classificadas como sub provisão

$overMap$: um mapa utilizado para contar a quantidade de alocações classificadas como super provisão

$overThr$: um limiar (positivo) utilizado para avaliar alocações classificadas como super provisão

ϕ: um parâmetro para ajustar a classe de configuração estimada por $mlAlg$

$(TD, \mathcal{A}_{dedup} \cup \{\mathcal{C}l\})$: um sistema de decisão

begin

  if $(underMap[mlAlg] = Nil)$ **then**

    $underMap[mlAlg] \leftarrow 0$

  if $(overMap[mlAlg] = Nil$ **then**

    $overMap[mlAlg] \leftarrow 0$

  $\mathcal{C}l(\tau) \leftarrow\ mlAlg((TD, \mathcal{A}_{dedup} \cup \{\mathcal{C}l\}), \tau)$

  if $(overMap[mlAlg] < underMap[mlAlg])$ **then**

    `// scaling up`

    $u \leftarrow \Big(Index(\mathcal{C}l(\tau)) + \phi\Big)$

    $\mathcal{C}l(\tau) \leftarrow \mathcal{C}l_u$

  else if $(overMap[mlAlg] > underMap[mAlg])$ **then**

    `// scaling down`

    $d \leftarrow \Big(Index(\mathcal{C}l(\tau)) - \phi\Big)$

    $\mathcal{C}l(\tau) \leftarrow \mathcal{C}l_d$

  $allocate(\tau, \mathcal{C}l(\tau))$

  $TD \leftarrow TD \cup \langle \Lambda(\tau), \eta(\tau), T_{exec}(\tau), \mathcal{C}l(\tau) \rangle$

  if $(T_{exec}(\tau) > T_{res}(\tau))$ **then**

    $underMap[mlAlg] \leftarrow underMap[mlAlg] + 1$

  else if $((T_{res}(\tau) - T_{exec}(\tau)) > overThr)$ **then**

    $overMap[mlAlg] \leftarrow overMap[mlAlg] + 1$

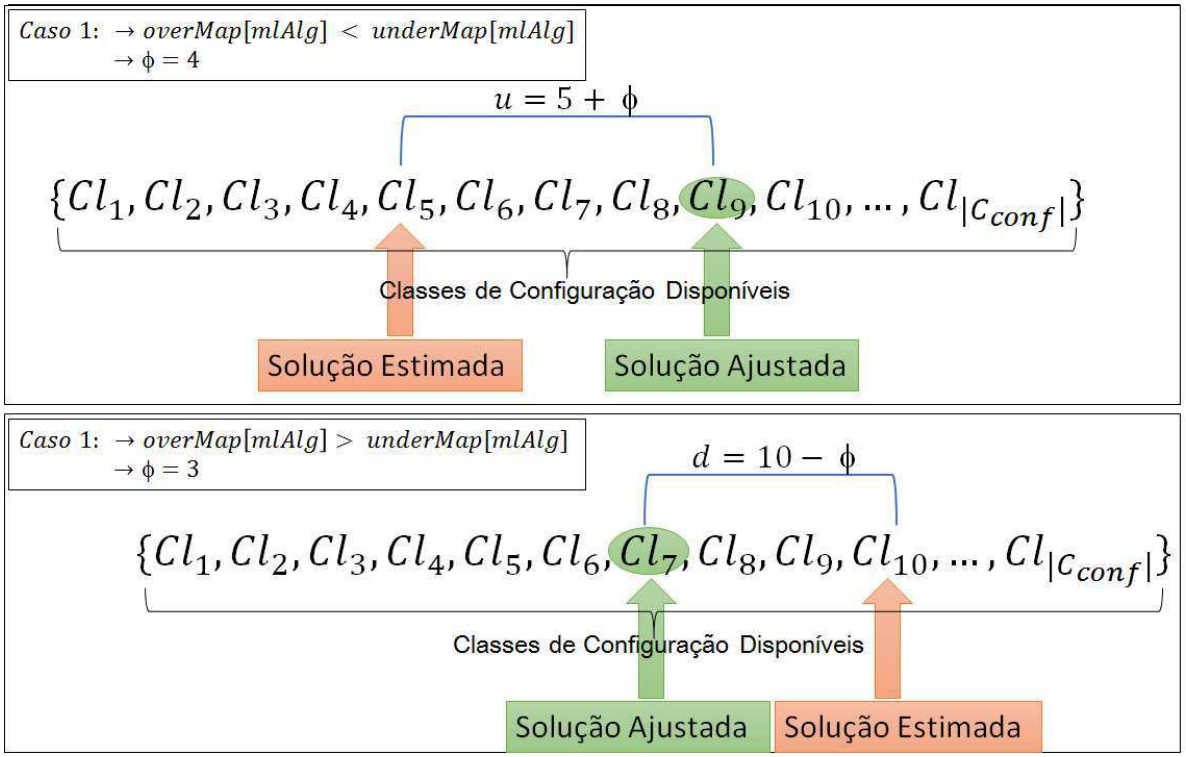

Figura C.4: Ilustração do funcionamento do algoritmo *Tunable Allocation*.

sivelmente ajustar) um algoritmo de provisionamento que é selecionado de um conjunto de entrada (como é realizado pelas heurísticas *Best Performing Allocation* e *Probabilistic Best Performing Allocation*). A tentativa de combinar heurísticas para seleção de algoritmos de provisionamento e ajuste de classes de configuração não foi explorada neste trabalho.

C.3.3 Heurística *Adaptive Allocation*

A desvantagem óbvia relacionada à heurística *Tunable Allocation* é a dificuldade de atribuir um valor apropriado para o parâmetro ϕ. Para superar esta desvantagem, nesta seção é proposta outra heurística, denominada *Adaptive Allocation*. O pseudocódigo da heurística *Adaptive Allocation* é mostrado no Algoritmo 13.

Nas linhas 2-4, a heurística *Adaptive Allocation* funciona de maneira similar ao Algoritmo 12. No entanto, a heurística *Adaptive Allocation* utiliza um mapa ($scalingMap$) para armazenar o parâmetro de ajuste associado ao algoritmo de provisionamento ($mlAlg$). Assim, após utilizar o algoritmo de provisionamento $mlAlg$ no sistema de decisão (linha 5), é verificada a necessidade de ajustar a classe de configuração estimada (linha 7 ou linha 11).

Algoritmo 13: Adaptive Allocation

input : τ: uma tarefa de deduplicação de dados, $mlAlg$: um algoritmo de aprendizado de máquina, $underMap$: um mapa para contar alocações classificadas como sub provisão, $overMap$: um mapa para contar alocações classificadas como super provisão, $scalingMap$: um mapa para armazenar o parâmetro de ajuste, $overThr$: um limiar (positivo) utilizado para avaliar alocações classificadas como super provisão, ϕ: um parâmetro de ajuste inicial, δ: um parâmetro para adaptar o parâmetro de ajuste ($scalingMap$), $(TD, \mathcal{A}_{dedup} \cup \{\mathcal{C}l\})$: um sistema de decisão

begin
- **if** $(scalingMap[mlAlg] = Nil)$ **then**
  - $underMap[mlAlg] \leftarrow overMap[mlAlg] \leftarrow 0$
  - $scalingMap[mlAlg] \leftarrow \phi$
- $\mathcal{C}l(\tau) \leftarrow mlAlg((TD, \mathcal{A}_{dedup} \cup \{\mathcal{C}l\}), \tau)$
- $tuned \leftarrow false$
- **if** $(overMap[mlAlg] > underMap[mlAlg])$ **then**
  - $u \leftarrow \Big(Index(\mathcal{C}l(\tau)) + scalingMap[mAlg]\Big)$
  - $\mathcal{C}l(\tau) \leftarrow \mathcal{C}l_u$
  - $tuned \leftarrow true$
- **else if** $(overMap[mlAlg] < underMap[mAlg])$ **then**
  - $d \leftarrow \Big(Index(\mathcal{C}l(\tau)) - scalingMap[mAlg]\Big)$
  - $\mathcal{C}l(\tau) \leftarrow \mathcal{C}l_d$
  - $tuned \leftarrow true$
- $allocate(\tau, \mathcal{C}l(\tau))$
- $TD \leftarrow TD \cup \langle \Lambda(\tau), \eta(\tau), T_{exec}(\tau), \mathcal{C}l(\tau) \rangle$
- **if** $(T_{exec}(\tau) > T_{res}(\tau))$ **then**
  - $underMap[mlAlg] \leftarrow underMap[mlAlg] + 1$
- **else if** $(T_{res}(\tau) - T_{exec}(\tau)) > overThr)$ **then**
  - $overMap[mlAlg] \leftarrow overMap[mlAlg] + 1$
- **if** $(T_{exec}(\tau) > T_{res}(\tau)$ ***or*** $(T_{res}(\tau) - T_{exec}(\tau)) > overThr)$ **then**
  - **if** $(tuned)$ **then**
    - $scalingMap[mAlg] \leftarrow \lceil (scalingMap[mAlg])^{(1+\delta)} \rceil$

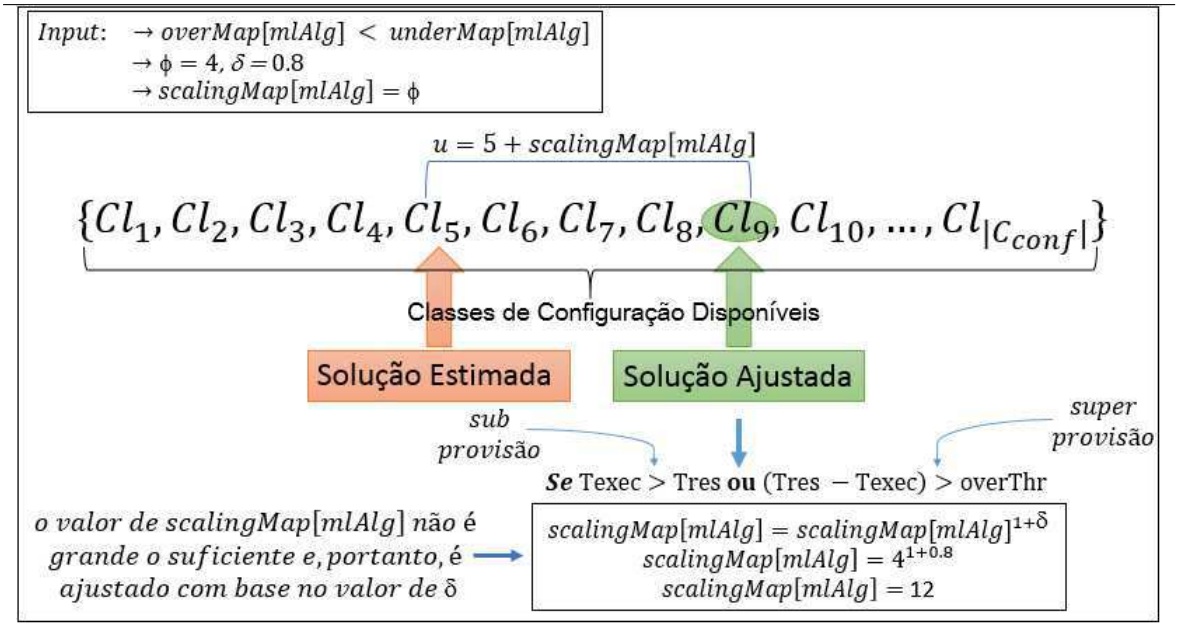

Figura C.5: Ilustração do funcionamento do algoritmo *Adaptive Allocation*.

Se for o caso, então a classe de configuração estimada é ajustada (linha 8-9 ou linha 12-13) e este fato é representado alterando o valor de uma variável lógica ($tuned = true$). Então, após executar a tarefa de deduplicação de dados (linha 15) e atualizar a base de treinamento (linha 16) do sistema de decisão, é verificado se a classe de configuração estimada produziu uma alocação classificada como sub provisão (linha 17) ou super provisão (linha 19). Em caso positivo, o mapa respectivo ($overMap$ ou $underMap$) é incrementado em 1 (linha 18 ou linha 20) e é verificado se a classe de configuração estimada foi ajustada (linha 22). Se for o caso, então significa que o parâmetro de ajuste não apresenta um valor alto o suficiente. Nestes casos, o parâmetro de ajuste (armazenado no $scalingMap$) é adaptado (linha 23) de acordo com o valor do parâmetro δ. Uma ilustração do funcionamento da heurística *Adaptive Allocation* é apresentada na Figura C.5.

C.3.4 Heurística *Sliced Training Data*

A ideia básica da heurística *Sliced Training Data* consiste em empregar um algoritmo de aprendizado de máquina, utilizado no contexto de provisionamento de recursos em um SMQD, utilizando uma porção específica da base de treinamento disponível que tende a diminuir a quantidade de estimativas de classe de configuração classificadas como sub pro-

visão. Para tal, a ideia consiste em desconsiderar da base de treinamento as instâncias cujas características representam uma complexidade menor do que as características da tarefa de deduplicação de dados a ser classificada. Esta intuição é formalizada da seguinte maneira.

Sejam τ uma tarefa de deduplicação de dados e TD a base de treinamento de um sistema de decisão para tarefas de deduplicação de dados $(TD, \mathcal{A}_{dedup} \cup \{\mathcal{C}l\})$. Assim, utilizando a notação de álgebra relacional[1], a base de treinamento filtrada $TD_{\lhd}$ é produzida como mostrada na Eq. (C.2).

$$TD_{\lhd} = \sigma_{\Lambda \geq \Lambda(\tau) \ \wedge \ \eta \geq \eta(\tau) \ \wedge \ T_{exec} \leq T_{res}(\tau)}(TD) \tag{C.2}$$

Desse modo, para um algoritmo de aprendizado de máquina $mlAlg$, uma tarefa de deduplicação de dados τ e um sistema de decisão para tarefas de deduplicação de dados $(TD, \mathcal{A}_{dedup} \cup \{\mathcal{C}l\})$, a heurística *Sliced Training Data* substitui a operação $mlAlg((TD, \mathcal{A}_{dedup} \cup \{\mathcal{C}l\}), \tau)$ por $mlAlg((TD_{\lhd}, \mathcal{A}_{dedup} \cup \{\mathcal{C}l\}), \tau)$. Nos casos em que $TD_{\lhd}$ não possui instâncias suficientes para permitir a classificação por $mlAlg$, então uma abordagem baseada em super provisão é empregada pela heurística.

C.4 Heurísticas para Deduplicação Incremental de Dados

$\mathcal{Z}$-$Filter$

Nesta seção, é proposto um filtro de componente de cobertura genérico, denominado $\mathcal{Z}$-$Filter$, o qual provê a estrutura básica dos passos realizados por um filtro de componente de cobertura. O algoritmo $\mathcal{Z}$-$Filter$ é representado no Algoritmo 14[2]. Além do conjunto de agrupamentos ($\mathcal{L}_{\dot{T}(\Delta G)}$) pertencentes ao componente de cobertura, o algoritmo $\mathcal{Z}$-$Filter$

[1] O operador de seleção é uma operação unária que retorna o subconjunto de uma relação e é representado como $\sigma_{\varphi}(R)$, tal que: i) φ é uma fórmula proposicional que consiste em átomos conectados pelos operadores lógicos $\wedge$, $\vee$ ou not; e ii) um átomo é representado na forma ($a\theta b$) ou ($a\theta v$), tal que a e b são atributos de R, v é uma constante e θ é um elemento do conjunto $\{<, \leq, =, \neq, \geq, >\}$.

[2] O pseudocódigo do Algoritmo 14 é apenas ilustrativo. Na prática, o algoritmo $\mathcal{Z}$-$Filter$ é implementado gerando um conjunto ordenado dos agrupamentos em $\mathcal{L}_{\dot{T}(\Delta G)}$, seguindo a lógica do filtro de componente de cobertura (linha 4 do Algoritmo 14) e, em seguida, retornando os agrupamentos nas $\lfloor \mu \times |\mathcal{L}_{\dot{T}(\Delta G)}| \rfloor$ primeiras posições do conjunto ordenado

também recebe como parâmetro a porcentagem $(0, 1)$ de agrupamentos do conjunto $\mathcal{L}_{\dot{T}(\Delta G)}$ que serão selecionados pelo filtro, denotado por μ.

Algoritmo 14: $\mathcal{Z}$-$Filter$

Input : $\mathcal{L}_{\dot{T}(\Delta G)}$: um agrupamento em $\mathcal{L}_{\dot{T}(\Delta G)}$
μ: a porcentagem $(0, 1)$ para determinar a quantidade de agrupamentos de $\mathcal{L}_{\dot{T}(\Delta G)}$ a serem selecioandos pelo filtro

Output: $\mathbf{Q^c}$: uma pilha de agrupamentos

```
1 begin
2     Q^c ← ∅
3     for j = 1 to ⌊μ × |L_Ṫ(ΔG)|⌋ do
4         C ← ⟨generic cluster selection⟩ | C ∉ Q^c
5         push(Q^c, C)
6     return Q^c
```

O algoritmo $\mathcal{Z}$-$Filter$ inicia criando uma pilha de agrupamentos vazia (linha 2). Em seguida, são selecionados $\lfloor \mu \times |\mathcal{L}_{\dot{T}(\Delta G)}| \rfloor$ agrupamentos do conjunto $\mathcal{L}_{\dot{T}(\Delta G)}$ (linhas 3-5), de acordo com um método específico denominado $\langle generic\ cluster\ selection \rangle$. Assim, essa chamada genérica é substituída por diferentes equações no intuito de gerar diferentes filtros de componente de cobertura.

$\mathcal{R}$-$Filter$

A heurística $\mathcal{R}$-$Filter$ simplesmente retorna um conjunto aleatório de agrupamentos (que possuem mais de um vértice) do conjunto $\mathcal{L}_{\dot{T}(\Delta G)}$. Então, a chamada genérica do algoritmo $\mathcal{Z}$-$Filter$ é substituída por:

$$C \leftarrow\ random\ C \in \mathcal{L}_{\dot{T}(\Delta G)} \mid |C| > 1 \tag{C.3}$$

Claramente, este é o filtro de componente de cobertura mais eficiente, pois o mesmo não requer nenhum processamento para selecionar os agrupamentos.

$l\mathcal{C}$-$Filter$

A heurística $l\mathcal{C}$-$Filter$ visa selecionar os agrupamentos (que possuem mais de um vértice) menos coesos contidos em $\mathcal{L}_{\dot{T}(\Delta G)}$. Portanto, a chamada genérica do algoritmo $\mathcal{Z}$-$Filter$ é substituída por:

$$C \leftarrow \underset{C \in L_{\dot{T}(\Delta G)}, |C|>1}{arg\,max} \Big(\sum_{v,v' \in C} 1 - sim(v, v') \Big) \tag{C.4}$$

Exemplo C.4.1 *Considerando os incrementos mostrados na Figura 2.5, o resultado da Eq. (C.4) sobre o conjunto de agrupamentos $\mathcal{L}_G$ é o agrupamento C_7.*

$\mathcal{M}d$-$Filter$

A heurística $\mathcal{M}d$-$Filter$ seleciona os agrupamentos (contendo mais de um vértice) do conjunto $\mathcal{L}_{\dot{T}(\Delta G)}$ que possuem os maiores valores em relação à diferença entre o grau de arestas entre agrupamentos e o grau de arestas entre vértices do próprio agrupamento. Seja $v \in G$, $dg^{\uparrow}(v)$ o conjunto de arestas entre agrupamentos do vértice v e $dg_{\downarrow}(v)$ o conjunto de arestas que conectam v a outros vértices do mesmo agrupamento de v. Desse modo, a chamada genérica do algoritmo $\mathcal{Z}$-$Filter$ é substituída por:

$$C \leftarrow \underset{C \in L_{\dot{T}(\Delta G)}, |C|>1}{arg\,max} \Big(\sum_{v \in C} dg^{\uparrow}(v) - dg_{\downarrow}(v) \Big) \tag{C.5}$$

Exemplo C.4.2 *Considerando os incrementos mostrados na Figura 2.5, o resultado da Eq. (C.5) sobre o conjunto de agrupamentos $\mathcal{L}_G$ é o agrupamento C_4.*

$\mathcal{MN}$-$Filter$

Seja $N(C)$ o conjunto de agrupamentos conectados ao agrupamento C em $\mathcal{L}_G$, ou seja, $\forall \langle C' \in N(C) \rangle \exists \langle e(v, v') \in G \rangle (v \in C \wedge v' \in C')$. A heurística $\mathcal{MN}$-$Filter$ visa selecionar os agrupamentos (contendo mais de um vértice) do conjunto $\mathcal{L}_{\dot{T}(\Delta G)}$ que possuem a maior quantidade de vizinhos. Para tal, a chamada genérica do algoritmo $\mathcal{Z}$-$Filter$ é substituída por:

$$C \leftarrow \underset{C \in L_{\dot{T}(\Delta G)}, |C|>1}{arg\,max} \Big(|N(C)| \Big) \tag{C.6}$$

Exemplo C.4.3 *Considerando os incrementos mostrados na Figura 2.5, o resultado da Eq. (C.6) sobre o conjunto de agrupamentos $\mathcal{L}_G$ é o agrupamento C_7.*

$m\mathcal{C}$-$Filter$

A heurística $m\mathcal{C}$-$Filter$ visa selecionar os agrupamentos (contendo mais de um vértice) do conjunto $\mathcal{L}_{\dot{T}(\Delta G)}$ que possuem os menores valores de conectividade. Logo, a chamada genérica do algoritmo $\mathcal{Z}$-$Filter$ é substituída por:

$$C \leftarrow \underset{C \in L_{\dot{T}(\Delta G)}, |C|>1}{arg\,min} \left(\frac{\sum\limits_{v,v' \in C} sim(v, v')}{\frac{|C| \times (|C|-1)}{2}} \right) \tag{C.7}$$

Exemplo C.4.4 *Considerando os incrementos mostrados na Figura 2.5, o resultado da Eq. (C.7) sobre o conjunto de agrupamentos $\mathcal{L}_G$ é o agrupamento C_7.*

$\mathcal{M}w$-$Filter$

A heurística $\mathcal{M}w$-$Filter$ tem como objetivo selecionar agrupamentos (contendo mais de um vértice) do conjunto $\mathcal{L}_{\dot{T}(\Delta G)}$ que possuem os maiores valores em relação à diferença entre a soma das arestas entre agrupamentos e a soma das arestas entre vértices do próprio agrupamento. Portanto, a chamada genérica do algoritmo $\mathcal{Z}$-$Filter$ é substituída por:

$$C \leftarrow \underset{C \in L_{\dot{T}(\Delta G)}, |C|>1}{arg\,max} \Big(\sum_{v \in C, v' \in N(C)} sim(v, v') - \sum_{v,v' \in C} sim(v, v') \Big) \tag{C.8}$$

Exemplo C.4.5 *Considerando os incrementos mostrados na Figura 2.5, o resultado da Eq. (C.8) sobre o conjunto de agrupamentos $\mathcal{L}_G$ é o agrupamento C_4.*

Metaheurística CFG

Nesta seção é apresentada uma metaheurística denominada $Clustered\text{-}based\ Filtered$ **Q**-$Greedy$ (CFG). A metaheurística CFG combina algoritmos de agrupamento automático, filtros de componente de cobertura e o algoritmo **Q**-$Greedy$ (Algoritmo 5) para gerar diferentes métodos de DID. A metaheurística CFG é representada no Algoritmo 15.

A metaheurística CFG inicialmente atualiza o grafo de similaridade (G) de acordo com os incrementos nos dados pertencentes ao conjunto ΔG (linha 2). Em seguida, se um algoritmo de agrupamento automático ($\mathcal{C}l_{alg}$) é recebido como parâmetro (ou seja, a expressão na linha 3 é avaliada como verdadeira), então o algoritmo $\mathcal{C}l_{alg}$ é utilizado para agrupar os vértices do componente de cobertura $\dot{T}(\Delta G)$ (linha 4). Prosseguindo, os resultados dos agrupamentos em $\mathcal{L}_{G+\Delta G}$ são atualizados (linha 5) considerando os agrupamentos adicionados e/ou atualizados por $\mathcal{C}l_{alg}$.

Então, independente se a expressão na linha 3 foi avaliada como verdadeiro ou falso, é verificado se a metaheurística recebeu um filtro ($\mathcal{F}$) de componente de cobertura como parâmetro de entrada. Se for o caso, a heurística $\mathcal{F}$ é utilizada para filtrar os agrupamentos em $\mathcal{L}_{\dot{T}(\Delta G)}$ (linha 7), levando em consideração o valor do parâmetro μ, e os agrupamentos filtrados são processados pelo algoritmo **Q**-$Greedy$ (linha 8) para gerar os agrupamentos resultantes. Caso contrário, é verificado se o valor do parâmetro $greedy\_flag$ é verdadeiro (linha 9). Se for o caso, então cada agrupamento em $\dot{T}(\Delta G)$ é adicionado (linhas 10-12) na pilha $\mathbf{Q^c}$ (note que os agrupamentos em $\dot{T}(\Delta G)$ podem ter sido modificados por $\mathcal{C}l_{alg}$) e a pilha $\mathbf{Q^c}$ é passada como parâmetro para o algoritmo **Q**-$Greedy$ para gerar os agrupamentos resultantes.

Por fim, se nem o parâmetro $\mathcal{F}$ é diferente de nil nem o parâmetro $greedy\_flag$ é igual $true$, então os agrupamentos gerados por $\mathcal{C}l_{alg}(\dot{T}(\Delta G))$ (linha 4) são retornados (linha 14) como os agrupamentos resultantes.

Algoritmo 15: Metaheurística CFG

Input : $G(V, E)$: um grafo de similaridade

ΔG: um conjunto de incrementos sobre G

$\mathcal{L}_G$: um agrupamento em G

$\mathcal{C}l_{alg}$: um algoritmo de agrupamento automático

$\mathcal{F}$: um filtro de componente de cobertura

μ: a porcentagem $(0, 1)$ de agrupamentos do conjunto

$\mathcal{L}_{\dot{T}(\Delta G)}$ a ser selecionada por $\mathcal{F}$

$\dot{T}$: um método de seleção de componentes de cobertura (T, $\hat{T}$ ou $\bar{T}$)

greedy_flag: um parâmetro lógico para disparar a execução do algoritmo Greedy

Output: $\mathcal{L}_{G+\Delta G}$: a clustering on $G + \Delta G$

begin
  $updateSimilarityGraph(G(V, E), \Delta G)$
  if $(\mathcal{C}l_{alg} <> Nil)$ **then**
    $\mathcal{L}_{\dot{T}(\Delta G)} \leftarrow \mathcal{C}l_{alg}(\dot{T}(\Delta G))$
    $\mathcal{L}_{G+\Delta G} \leftarrow update(\mathcal{L}_G, \mathcal{L}_{\dot{T}(\Delta G)})$
  if $(\mathcal{F} <> Nil)$ **then**
    $\mathcal{L}^{\mathcal{F}}_{\dot{T}(\Delta G)} \leftarrow \mathcal{F}(\mathcal{L}_{\dot{T}(\Delta G)}, \mu)$
    return Q-Greedy($\mathcal{L}^{\mathcal{F}}_{\dot{T}(\Delta G)}$)
  else if *(*$greedy\_flag = true$*)* **then**
    $\mathbf{Q^c} \leftarrow \emptyset$
    $G' \leftarrow \dot{T}(\Delta G)$
    $adiciona\ cada\ agrupamento\ de\ G'\ em\ \mathbf{Q^c}$
    return Q-Greedy($G + \Delta G$, $\mathbf{Q^c}$, $\mathcal{L}_{G+\Delta G}$)
  return $\mathcal{L}_{G+\Delta G}$

C.5 Heurísticas para Controlar o Tamanho de Blocos

C.5.1 Heurísticas para Reduzir o Tamanho de Blocos ($Shrink$)

As heurísticas propostas nesta seção possuem dois objetivos. Primeiro, quanto maior a redução de entidades nos blocos da coleção de blocagens podada (i.e., $\sum_{B\in\mathcal{B}'}\sum_{b\in B}|b|$), maior será a taxa de redução entre a coleção de blocagens recebida como entrada e a coleção de blocagens podada, i.e., $RR(\mathcal{B},\mathcal{B}')$. Segundo, ao reduzir a cardinalidade agregada da coleção de blocos podada (i.e., $\sum_{B\in\mathcal{B}'}\sum_{b\in B}|b|$), os tempos de execução dos passos seguintes da Metaheurística $MkPCBS$ também tendem a diminuir.

Neste sentido, o principal desafio das heurísticas propostas nesta seção consiste em remover entidades dos blocos sem comprometer significativamente os resultados de qualidade (PQ e PC) associados à coleção de blocos podada. No intuito de tratar este desafio, são propostas heurísticas que visam explorar os pesos de coocorrência médios das entidades nos blocos para selecionar entidades a serem removidas dos blocos.

low Entity Co Occurrence Pruning (lECP)

A heurística $low\ Entity\ Co\ Occurrence\ Pruning$ visa reduzir o tamanho de um bloco removendo as entidades que apresentam os menores pontuações de coocorrência de entidade (Definição 7.4.3) no bloco. Em outras palavras, o algoritmo visa reduzir o tamanho dos blocos removendo cada entidade e dos blocos em que é menos provável que seja encontrada duplicatas entre e e as demais entidades do bloco.

O pseudocódigo da heurística $lECP$ é apresentado no Algoritmo 16. Para cada bloco b na coleção de blocagens (linhas 2-3), o algoritmo realiza as seguintes tarefas: i) é criada uma pilha de prioridade ordenada pela pontuação de coocorrência (Definição 7.4.3) agregada das entidades (linha 5); ii) é adicionada cada entidade $e \in b$ na pilha (linhas 7-9); e iii) são removidas $\lfloor|b|*\lambda\rfloor$ entidades de b que possuem as menores pontuações de coocorrência de entidade em b (linhas 11-12). Finalmente, a coleção de blocagens podada é retornada[3] na linha 13. Uma ilustração[4] do funcionamento da heurística $lECP$ é apresentada no Exemplo

[3]É assumido que as alterações realizadas pelo algoritmo são refletidas na coleção de blocos de entrada $\mathcal{B}$. A mesma suposição é feita nos Algoritmos 16-21

[4]para ilustrar o funcionamento das heurísticas propostas para controlar o tamanho de blocos, é empregado

C.5.1.

Exemplo C.5.1 (low Entity Co Occurrence Pruning (lECP)) *Considerando a coleção de blocagens $\mathcal{B}_{D_1}$ mostrada no Exemplo 7.2.5, caso sejam escolhidos os valores de parâmetros $WS = CBS$ e $\lambda = 0.4$, então o bloco $b = R\{d_7, d_8, d_{12}\}$ é podado pelo algoritmo $lECP$ da seguinte maneira: i) as pontuações de coocorrência agregada (WS = CBS) entre os pares de entidades em b são calculadas como: $CBS(d_7, d_8) = 2$, $CBS(d_7, d_{12}) = 1$ e $CBS(d_8, d_{12}) = 1$; ii) a pilha de prioridade $Q = \{(d_7, \frac{2+1}{2}), (d_8, \frac{2+1}{2}), (d_{12}, \frac{1+1}{2})\}$ (ver Definição 7.4.3) é gerada (linhas 5-9); iii) o valor de $limit$ é calculado como $\lfloor |b| * \lambda \rfloor = \lfloor 3 * 0.4 \rfloor = 1$ (linha 10); e iv) portanto, uma única entidade ($Q.pop() = d_{12}$) é excluída do bloco b (linhas 11-12). Note que esta operação de poda não reduz o resultado de PC da coleção de blocagens podada.*

low Block Co Occurrence Pruning (lBCP)

A heurística *low Block Co Occurrence Pruning* visa podar a coleção de blocagens recebida como entrada ao remover cada entidade e dos blocos em que a mesma apresenta as menores pontuações de coocorrência agregada. O pseudocódigo da heurística $lBCP$ é apresentado no Algoritmo 17. Para cada entidade e na coleção de blocagens $\mathcal{B}$ (linha 2), o algoritmo realiza as seguintes tarefas: i) é criada uma pilha de prioridade ordenada pela pontuação de coocorrência agregada de e (linha 4); ii) é criado um conjunto ($\mathcal{B}_e$) contendo cada bloco b, tal que $e \in b$ (linha 6); iii) cada bloco $b \in \mathcal{B}_e$ é adicionado na pilha de prioridade; e iv) a entidade e é removida de $\lfloor |\mathcal{B}_e| * \lambda \rfloor$ blocos nos quais e apresenta as menores pontuações de coocorrência agregada (linhas 12-14). Finalmente, a coleção de blocos podada é retornada na linha 15. Uma ilustração de funcionamento da heurística $lBCP$ é apresentado no Exemplo C.5.2.

Exemplo C.5.2 (low Block Co Occurrence Pruning (lBCP)) *Considerando a coleção de blocagens $\mathcal{B}_{D_1}$ apresentada no Exemplo 7.2.5, caso sejam definidos os valores de parâmetros $WS = CBS$ e $\lambda = 0.4$, então os blocos da entidade d_{10} são podados pelo algoritmo $lBCP$ da seguinte maneira: i) a pontuação de coocorrência agregada (WS = CBS) entre*

um esquema de pontuação [62] que simplesmente conta a quantidade de blocos compartilhados por duas entidades, i.e., $CBS(e_1, e_2) = |B_{e_1,e_2}|$, s.t. $\forall b \in B_{e_1,e_2} : e_1 \in b \wedge e_2 \in b$

Algoritmo 16: $lECP$

Input : $\mathcal{B}$: coleção de blocagens

WS: esquema de pontuação de coocorrência

λ: porcentagem (0,1) que determina o alcance da heurística

Output: uma coleção de blocagens podada

```
1  begin
2      foreach B ∈ 𝓑 do
3          foreach b ∈ B do
4              // cria uma pilha de prioridade ordenada pela
                  pontuação de coocorrência agregada das
                  entidades
5              Q ← GeneratePriorityQueue()
6              // adicionar cada entidade e ∈ b na pilha
7              foreach e ∈ b do
8                  // usando Definição 7.4.3
9                  Q.push(e, EntityCooScore(e, b, 𝓑, WS))
10             limit ← ⌊|b|∗λ⌋
11             for i = 1 to limit do
12                 b ← b\Q.pop()
13     return 𝓑
```

Algoritmo 17: $lBCP$

Input : $\mathcal{B}$: coleção de blocos de entrada

WS: esquema de pontuação de coocorrência

λ: porcentagem (0,1) que determina o alcance da heurística

Output: a pruned blocking collection

```
1  begin
2      foreach e in the blocking collection B do
3          // cria uma pilha de prioridade ordenada pela
               pontuação de coocorrência agregada da entidade
4          Q ← GeneratePriorityQueue()
5          // seleciona os blocos da entidade e
6          B_e ← select every block b, s.t. e ∈ b
7          // adiciona cada bloco de e na pilha
8          foreach b ∈ B_e do
9              // usando a Definição 7.4.3
10             Q.push(b, EntityCooScore(e, b, B, WS))
11         limit ← ⌊|B_e| * λ⌋
12         for i = 1 to limit do
13             b ← Q.pop()
14             b ← b\e
15     return B
```

a entidade d_{10} *e as entidades* $\{d_2, d_3, d_5, d_6, d_8, d_9, d_{14}\}$ *(i.e., entidades que compartilham pelo menos um bloco com* d_{14}*) é calculada como:* $CBS(d_2, d_{10}) = 1$*,* $CBS(d_3, d_{10}) = 1$*,* $CBS(d_5, d_{10}) = 1$*,* $CBS(d_6, d_{10}) = 1$*,* $CBS(d_8, d_{10}) = 1$*,* $CBS(d_9, d_{10}) = 2$ *e* $CBS(d_{14}, d_{10}) = 1$*; ii) a pilha de prioridade* $Q = \{(G\{d_5, d_6, d_9, d_{10}\}, \frac{4}{3}), (Spain\{d_2, d_3, d_8, d_9, d_{10}\}, \frac{5}{4}), (L\{d_{10}, d_{14}\}, \frac{1}{1})\}$ *(ver Definição 7.4.3) é gerada (linhas 4-10); iii) o valor de* $limit$ *é calculado como* $\lfloor |\mathcal{B}_e| * \lambda \rfloor = \lfloor 3 * 0.4 \rfloor = 1$ *(linha 11); iv) assim, a entidade* d_{10} *é removida do bloco* $L\{d_{10}, d_{14}\}$ *(linhs 12-14). Note que esta operação de poda não reduz o resultado de* PC *da coleção de blocagens podada.*

Large Block Size Pruning (LBSP)

A heurística *Large Block Size Pruning* visa podar apenas os blocos grandes da coleção de blocagens recebida como entrada. A intuição desta ideia consiste em dois princípios: i) quanto maior é o tamanho do bloco, mais atenuante se torna as diferenças entre as pontuações de coocorrência agregadas das entidades do bloco e, portanto, a poda se torna mais acurada (i.e., a poda tende a remover entidades sem sacrificar resultados de qualidade); e ii) ao restringir o processo de poda sobre blocos grandes, o tempo de execução total para executar o primeiro passo da metaheurístics MkPCBS é reduzido e o foco do passo se torna os blocos que irão produzir mais impacto sobre a taxa de redução (RR).

O pseudocódigo da heurística $LBSP$ é apresentado no Algoritmo 18. Primeiro, o tamanho médio μ dos blocos é calculado (linha 2). Então, para cada bloco b na coleção de blocagens $\mathcal{B}$, tal que $|b| \geq \mu$, o algoritmo realiza as seguintes operações: i) é criada uma pilha de prioridade ordenada pela pontuação de coocorrência agregada das entidades (linha 7); e ii) são removidas $\lfloor |b| * \lambda \rfloor$ entidades do bloco b que possuem as menores pontuações de coocorrência agregada (linhas 13-14). Por fim, a coleção de blocagens podada é retornada na linha 15. É apresentada uma ilustração do funcionamento do algoritmo LBSP no Exemplo C.5.3.

Exemplo C.5.3 (Large Block Size Pruning (LBSP)) *Considerando a coleção de blocagens* $\mathcal{B}_{D_1}$ *mostrada no Exemplo 7.2.5, note que* $\mu = \frac{49}{16} = 3.06$*. Assim, apenas os blocos* $A\{d_1, d_2, d_5, d_7, d_8, d_{16}\}$*,* $Salamanca\{d_1, d_4, d_5, d_6, d_7, d_{16}\}$*,* $Spain\{d_2, d_3, d_8, d_9, d_{10}\}$*,* $F\{d_2, d_4, d_{14}, d_{13}\}$*,* $G\{d_5, d_6, d_9, d_{10}\}$*,* $C\{d_2, d_3, d_4\}$*,* $\{N\{d_1,\ d_3,\ d_{16}\}$*,* $R\{d_7, d_8, d_{12}\}$ *e* $Italy$ $\{d_{11},\ d_{14},\ d_{15}\}$ *são podados pelo algoritmo LBSP. Caso os valores de parâmetros*

Algoritmo 18: $LBSP$

Input : $\mathcal{B}$: coleção de blocagens

WS: esquema de pontuação de coocorrência

λ: porcentagem (0,1) que determina o alcance da heurística

Output: coleção de blocagens podada

```
1  begin
2      μ ← (∑_{B∈ℬ} ∑_{b∈B} |b|) / ∑_{B∈ℬ} |B|
3      foreach B ∈ ℬ do
4          foreach b ∈ B do
5              if (|b| ≥ μ) then
6                  // cria uma pilha de prioridade ordenada pela
                      pontuação de coocorrência agregada das
                      entidades
7                  Q ← GeneratePriorityQueue()
8                  // adiciona cada entidade e ∈ b na pilha
9                  foreach e ∈ b do
10                     // usando Definição 7.4.3
11                     Q.push(b, EntityCooScore(e, b, ℬ, WS))
12                 limit ← ⌊|b| * λ⌋
13                 for i = 1 to limit do
14                     b ← b \ Q.pop()
15     return ℬ
```

$WS = CBS$ *e* $\lambda = 0.2$ *sejam definidos, então o bloco* $b = Spain\{d_2, d_3, d_8, d_9, d_{10}\}$ *é podado pelo algoritmo* $LBSP$ *da seguinte maneira: i) as pontuações de coocorrência agregada (WS = CBS) entre os pares de entidades em* b *são calculadas como:* $CBS(d_2, d_3) = 2$*,* $CBS(d_2, d_8) = 1$*,* $CBS(d_2, d_9) = 1$*,* $CBS(d_2, d_{10}) = 1$*,* $CBS(d_3, d_8) = 1$*,* $CBS(d_3, d_9) = 1$*,* $CBS(d_3, d_{10}) = 1$*,* $CBS(d_8, d_9) = 1$*,* $CBS(d_8, d_{10}) = 1$ *e* $CBS(d_9, d_{10}) = 2$*; ii) é gerada a pilha de prioridade* $Q = \{(d_2, \frac{5}{4}), (d_3, \frac{5}{4}), (d_9, \frac{5}{4}), (d_{10}, \frac{5}{4}), (d_8, \frac{4}{4})\}$ *(ver Definição 7.4.3) (linhas 7-11); iii) o valor de* $limit$ *é calculado como* $\lfloor |b| * \lambda \rfloor = \lfloor 5 * 0.2 \rfloor = 1$ *(linha 12); e iv) portanto, uma única entidade (*$Q.pop() = d_8$*) é removida do bloco* b *(linhas 13-14). Note que esta operação de poda não reduz o resultado de* PC *da coleção de blocagens podada.*

C.5.2 Heurística para Dividir Blocos

Nesta seção, é proposta uma heurística que visa executar o segundo passo da metaheurística proposta ($MkPCBS$), i.e., garantir um tamanho máximo nos blocos de uma coleção de blocagens podada. A heurística proposta ($CooSlicer$) divide blocos grandes ($|b| > S_{max}$) com base nos valores de coocorrência agregada (Definição 7.4.3) de suas entidades. Desse modo, os sub blocos são iterativamente produzidos a partir do agrupamento de entidades que apresentam as maiores pontuações de coocorrência agregada dentre as entidades restantes no bloco. Para tal, para cada bloco b na coleção de blocagens, tal que $|b| > S_{max}$, b é dividido em $\lceil \frac{|b|}{S_{max}} \rceil$ sub blocos.

O pseudocódigo da heurística $CooSlicer$ é apresentado no Algoritmo 19. Inicialmente, o algoritmo cria o conjunto B^+, o qual contém cada bloco b da coleção de blocagens, tal que $|b| > S_{max}$ (linha 3). Então, para cada bloco $b \in B^+$ (linha 4), o algoritmo realiza as seguintes tarefas: i) é criada uma pilha de prioridade, ordenada pela pontuação de coocorrência agregada das entidades (linha 6); ii) cada entidade $e \in b$ é adicionada na pilha (linhas 8-10); iii) é calculada a quantidade de blocos ($\#blocks$) e o tamanho máximo ($bSize$) dos sub blocos que serão produzidos dividindo o bloco b (linhas 11-12); iv) são criados $\#blocks$ sub blocos contendo (note que o último bloco gerado pode conter $c < bSize$ entidades) no máximo $bSize$ entidades (linhas 14-19); e v) o bloco b é removido da coleção de blocagens. Finalmente, a coleção de blocagens podada é retornada na linha 21. É apresentada uma ilustração do funcionamento do algoritmo $CooSlicer$ no Exemplo C.5.4.

Algoritmo 19: $CooSlicer$

Input : $\mathcal{B}$: coleção de blocagens

WS: esquema de pontuação de coocorrência

λ: porcentagem (0,1) que determina o alcance da heurística

Output: coleção de blocos podada contendo blocos de tamanho máximo igual a S_{max}

```
1  begin
2      // seleciona blocos grandes
3      B+ ← select every block b, s.t. |b|> S_max
4      foreach b ∈ B+ do
5          // cria uma pilha de prioridade ordenada pela
               pontuação de coocorrência agregada das entidades
6          Q ← GeneratePriorityQueue()
7          // adiciona cada entidade e ∈ b na pilha
8          foreach e ∈ b do
9              // usando Definição 7.4.3
10             Q.push(e, EnittyCooScore(e, b, WS, B))
11         #blocks ← ⌈|b| / S_max⌉
12         bSize ← ⌈|b| / #blocks⌉
13         B_b ← B ∈ B, s.t. b ∈ B
14         while (Q is not empty) do
15             b' ← ∅
16             for i = 1 to bSize do
17                 if (Q is not empty) then
18                     b' ← b' ∪ Q.pop()
19             B_b ← B_b ∪ b'
20         B_b ← B_b\b
21     return B
```

Exemplo C.5.4 (CooSlicer) *Considerando a coleção de blocagens $\mathcal{B}_{D_1}$ apresentada no Exemplo 7.2.5, caso sejam definidos os valores de parâmetros $WS = CBS$ e $S_{max} = 2$, o bloco $b = R\{d_7, d_8, d_{12}\}$ é dividido pelo algoritmo $cooSlicer$ da seguinte maneira: i) as pontuações de coocorrência agregada (WS = CBS) entre os pares de entidades em b são calculadas como: $CBS(d_7, d_8) = 2$, $CBS(d_7, d_{12}) = 1$ e $CBS(d_8, d_{12}) = 1$; ii) a pilha de prioridade $Q = \{(d_7, \frac{3}{2}), (d_8, \frac{3}{2}), (d_{12}, \frac{2}{2})\}$ (ver Definição 7.4.3) é gerada (linhas 6-10); iii) $\#blocks = \lceil \frac{|b|}{S_{min}} \rceil = \lceil \frac{3}{2} \rceil = 2$ (linha 11) e, portanto, $bSize = \lceil \frac{|b|}{\#blocks} \rceil = \lceil \frac{3}{2} \rceil = 2$ (linha 12); iv) os blocos $\{d_7, d_8\}$ e $\{d_{12}\}$ são gerados e inseridos na coleção de blocagens (linhas 14-19); e v) o bloco $R\{d_7, d_8, d_{12}\}$ é excluído da coleção de blocagens (linha 20). Por fim, a coleção de blocagens podada é retornada na linha 21. Note que esta operação de divisão de blocos não reduz o resultado de PC da coleção de blocagens podada.*

C.5.3 Heurística para Unir Blocos

Nesta seção, é proposta uma heurística que visa unir blocos de uma coleção de blocagens, a qual visa executar o terceiro passo da metaheurística proposta ($MkPCBS$). Este passo visa garantir que todos os blocos da coleção de blocagens apresentem um tamanho mínimo (S_{min}). Uma possível abordagem para realizar este passo consiste em calcular uma medida de similaridade entre um bloco $b \in B^-$ (i.e., o conjunto de blocos na coleção de blocagens, tal que $|b| < S_{min}$) com todos os outros blocos $b' \in B^- \backslash b$. Visando otimizar este processo, é empregada uma abordagem mais eficiente do que calcular a pontuação de coocorrência agregada entre todos os pares de entidades em $b \times b'$ (para todo $b \in B^-$ e $b' \in B^- \backslash b$)

Para tal, ao invés de calcular a pontuação de coocorrência entre pares de entidades, é proposta uma heurística que explora a quantidade de entidades pertencentes ao conjunto de interseção entre blocos pequenos ($|b| < S_{min}$). A intuição desta ideia é que quanto mais entidades dois blocos compartilham, maior é a probabilidade do conjunto formado pela interseção entre estes dois blocos incluir pares de entidades duplicadas que não compartilhavam blocos nos blocos originais. Desse modo, a heurística proposta ($MaxIntersectionMerge$) visa unir blocos pequenos com base no tamanho da interseção entre os blocos.

O pseudocódigo da heurística $MaxIntersectionMerge$ é apresentado no Algoritmo 20. Inicialmente, o algoritmo cria os conjuntos B^- e B^*, os quais contém cada bloco b na coleção de blocagens, tal que $|b| < S_{min}$ e $S_{min} \leq |b| \leq S_{max}$, respectivamente (linhas 3-5). Então,

enquanto o conjunto B^- não for vazio (linha 6), o algoritmo realiza as seguintes tarefas: i) o maior bloco (b_1) é selecionado do conjunto B^- (linha 7); ii) o bloco b_2 é selecionado de B^- (se $|B^-|> 1$) ou B^+ (se $|B^-|= 1$) (linha 9 ou linha 11); iii) é realizada a interseção entre os blocos b_1 e b_2 para formar o bloco b' (linha 13); iv) b_1 e b_2 são removidos de B^- (linhas 15-16); e v) b' é inserido no conjunto B^- (linha 19) ou B^* (linha 21), dependendo do seu tamanho. Por fim, a coleção de blocos podada é retornada na linha 22.

Um caso específico pode ocorrer[5] quando nenhum dos blocos do conjunto $B^-\backslash b_1$ (linha 9) ou B^* (linha 11) são elegíveis para serem unidos com o maior bloco $b_1 \in B^-$ (linha 7), i.e., $\forall b_2 \in B^-\backslash b_1 : (|b_1|+|b_2|) > S_{max}$ e $\forall b_2 \in B^* : (|b_1|+|b_2|) > S_{max}$. Neste caso, b_2 é selecionado como mostrado na linha 11 (removendo a restrição associada ao tamanho do bloco produzido a partir da operação de união, i.e., $(|b_1|+|b_2|) > S_{max}$) e são removidas as $|b'|-S_{max}$ entidades do bloco produzido b' (linha 13) que apresentem menores pontuações de coocorrência agregada em b'. Ou seja, é executada a heurística $lECP$ (Algoritmo 16) utilizando o valor de parâmetro $\lambda = \frac{|b'|-S_{max}}{|b'|}$. Esta redução no tamanho de b' visa visa garantir que $|b'|\leq S_{max}$. Uma ilustração da heurística $MaxIntersectionMerge$ é apresentada no Exemplo C.5.5.

Exemplo C.5.5 (MaxIntersectionMerge) *Considerando a coleção de blocagens $\mathcal{B}_{D_1}$ mostrada no Exemplo 7.2.5, caso sejam definidos os valores de parâmetro $WS = CBS$, $S_{min} = 7$ ed $S_{max} = 8$, então o bloco $b_1 = Salamanca\{d_1, d_4, d_5, d_6, d_7, d_{16}\}$ (linha 7) é processado pelo algoritmo $MaxIntesectionMerge$ da seguinte maneira: i) o bloco $b_2 = N\{d_1, d_3, d_{16}\}$ é selecionado porque apresenta a maior interseção com o bloco $Salamanca\{d_1, d_4, d_5, d_6, d_7, d_{16}\}$ e $|Salamanca\{d_1, d_4, d_5, d_6, d_7, d_{16}\} \cap N\{d_1, d_3, d_{16}\}|\leq 8$ (linha 9); ii) o bloco $b' = (N \cap Salamanca)\{d_1, d_3, d_4, d_5, d_6, d_7, d_{16}\}$ é criado (linha 13); iii) os blocos $Salamanca\{d_1, d_4, d_5, d_6, d_7, d_{16}\}$ e $N\{d_1, d_3, d_{16}\}$ são excluídos da coleção de blocagens (linhas 15-16); e iv) finalmente, o bloco $(N \cap Salamanca)\{d_1, d_3, d_4, d_5, d_6, d_7, d_{16}\}$ é inserido em B^* porque $7 \leq |(N \cap Salamanca)\{d_1, d_3, d_4, d_5, d_6, d_7, d_{16}\}|\leq 8$ (linha 21).*

[5]este caso especial não é mostrado no algoritmo $MarxIntersectionMerge$ no intuito de simplificar sua representação

Algoritmo 20: $MaxIntersectionMerge$

Input : $\mathcal{B}$: coleção de blocagens

S_{min}: tamanho mínimo dos blocos

S_{max}: tamanho máximo dos blocos

Output: coleção de blocagens podada contendo blocos de tamanho maior ou igual a S_{min}

begin

  `// seleciona blocos pequenos`

  $B^- \leftarrow$ select every block b, $s.t.\ |b| < S_{min}$

  `// seleciona blocos de tamanho correto`

  $B^* \leftarrow$ select every block b, $s.t.\ S_{min} \leq |b| \leq S_{max}$

  while $(|B^-| > 0)$ **do**

    $b_1 \leftarrow \underset{b \in B^-}{arg\,max} |b|$

    if $(|B^-| > 1)$ **then**

      $b_2 \leftarrow \underset{b \in B^- \setminus b_1,\ s.t.(|b1|+|b|) \leq S_{max}}{arg\,max} |b \cap b1|$

    else

      $b_2 \leftarrow \underset{b \in B^*,\ s.t.(|b1|+|b|) \leq S_{max}}{arg\,max} |b \cap b1|$

    `// une os blocos` b_1 `e` b_2

    $b' \leftarrow b_1 \cup b_2$

    `// remove os blocos` b_1 `e` b_2 `da coleção de blocagens`

    $B^- \leftarrow B^- \setminus b1$

    $B^- \leftarrow B^- \setminus b2$

    `// insere` b' `em` B^- `ou em` B^*`, dependendo do seu tamanho`

    if $(|b'| < S_{min})$ **then**

      $B^- \leftarrow B^- \cup b'$

    else

      $B^* \leftarrow B^* \cup b'$

  return $\mathcal{B}$

C.5.4 Heurística para Excluir Blocos

Nesta seção, é proposta uma heurística que exclui blocos de uma coleção de blocagens, o que corresponde ao quarto passo da metaheurística proposta ($MkPCBS$). A ideia da heurística proposta consiste em excluir da coleção de blocagens os blocos que apresentam as menores pontuações de coocorrência agregada entre as entidades (Definição 7.4.4).

O pseudocódigo da heurística $low\ Block\ Co\ Occurrence\ Excluder$ (lBCE) é apresentado no Algoritmo 21. Inicialmente, o algoritmo cria uma pilha de prioridade vazia ordenada pela pontuações de coocorrência agregada dos blocos (linha 2). Então, cada bloco b na coleção de blocagens (linhas 4-5) é adicionado na pilha (linha 7). Em seguida, a heurística $lBCE$ exclui $\lfloor(\sum_{B\in\mathcal{B}}|B|) * \lambda\rfloor$ blocos da coleção de blocagens que apresentam as menores pontuações de coocorrência agregada (linhas 9-12). Uma ilustração do funcionamento do algoritmo $lBCE$ é apresentada no Exemplo C.5.6.

Exemplo C.5.6 (low Block Co Occurrence Excluder (lBCE)) *Considerando a coleção de blocagens $\mathcal{B}_{D_1}$ mostrada no Exemplo 7.2.5, caso sejam definidos os valores de parâmetros $WS = CBS$ e $\lambda = 0.25$, então os blocos $R\{d_7, d_8, d_{12}\}$, $Spain\{d_2, d_3, d_8, d_9, d_{10}\}$, $C\{d_2, d_3, d_4\}$ e $L\{d_{10}, d_{14}\}$[6] são processados pela heurística $lBCE$ da seguinte maneira: i) a pilha de prioridade $Q = \{(C\{d_2, d_3, d_4\}, \frac{5.5}{3})^7, (R\{d_7, d_8, d_{12}\}, \frac{4}{3})^8, (Spain\{d_2, d_3, d_8, d_9, d_{10}\}, \frac{6}{5})^9, (L\{d_{10}, d_{14}\}, \frac{1}{1})^{10}$ (ver Definição 7.4.4) é gerada (linhas 3-7); ii) o valor de $limit$ é calculado como $\lfloor|\{R\{d_7, d_8, d_{12}\}, Spain\{d_2, d_3, d_8, d_9, d_{10}\},$*

[6]para simplificar o Exemplo C.5.6, apenas os blocos $R\{d_7, d_8, d_{12}\}$, $Spain\{d_2, d_3, d_8, d_9, d_{10}\}$, $C\{d_2, d_3, d_4\}$ e $L\{d_{10}, d_{14}\}$ são processados pela heurística $lBCE$. No entanto, todos os blocos na coleção de blocagens $\mathcal{B}_{D_1}$ são considerados para calcular as pontuações de coocorrência agregada dos blocos)

[7]uma vez que $CBS(d_2, d_3) = 2$, $CBS(d_2, d_4) = 2$ e $CBS(d_3, d_4) = 1$, então $EntityCooScore(d_2) = \frac{4}{2} = 2$, $EntityCooScore(d3) = \frac{4}{2} = 2$ e $EntityCooScore(d_4) = \frac{3}{2} = 1.5$. Portanto, $BlockCooScore(C\{d_2, d_3, d_4\}) = \frac{5.5}{3}$

[8]uma vez que $CBS(d_7, d_8) = 2$, $CBS(d_7, d_{12}) = 1$ e $CBS(d_8, d_{12}) = 1$, então $EntityCooScore(d_7) = \frac{2+1}{2} = 1.5$, $EntityCooScore(d_8) = \frac{2+1}{2} = 1.5$ e $EntityCooScore(d_{12}) = \frac{1+1}{2} = 1$. Assim, $BlockCooScore(R\{d_7, d_8, d_{12}\}) = \frac{4}{3}$

[9]uma vez que $CBS(d_2, d_3) = 2$, $CBS(d_2, d_8) = 1$, $CBS(d_2, d_9) = 1$, $CBS(d_2, d_{10}) = 1$, $CBS(d_3, d_8) = 1$, $CBS(d_3, d_9) = 1$, $CBS(d_3, d_{10}) = 1$, $CBS(d_8, d_9) = 1$, $CBS(d_8, d_{10}) = 1$ e $CBS(d_9, d_{10}) = 2$, então $EntityCooScore(d2) = \frac{5}{4}$, $EntityCooScore(d3) = \frac{5}{4}$, $EntityCooScore(d8) = \frac{4}{4}$, $EntityCooScore(d9) = \frac{5}{4}$ e $EntityCooScore(d10) = \frac{5}{4}$. Desse modo, $BlockCooScore(Spain\{d_2, d_3, d_8, d_9, d_{10}\}) = \frac{6}{5}$

[10]Uma vez que $CBS(d_{10}, d_{14}) = 1$, então $EntityCooScore(d_{10}) = \frac{1}{1} = 1$ e $EntityCooScore(d_{14}) =$

Algoritmo 21: $lBCE$

Input : $\mathcal{B}$: coleção de blocagens

WS: esquema de pontuação de coocorrência

λ: porcentagem (0,1) que determina o alcance da heurística

Output: coleção de blocagens podada

```
1  begin
2      // cria uma pilha de prioridade ordenada pela pontuação
           de coocorrência agregada dos blocos
3      Q ← GeneratePriorityQueue()
4      foreach B ∈ 𝓑 do
5          foreach b ∈ B do
6              // usando Definição 7.4.4
7              Q.push(b, BlockCooScore(b, 𝓑, WS))
8      limit ← ⌊(∑_{B∈𝓑} |B|) * λ⌋
9      for i = 1 to limit do
10         b ← Q.pop()
11         B_b ← B ∈ 𝓑, s.t. b ∈ B
12         B_b ← B_b \ b
13     return 𝓑
```

$C\{d_2, d_3, d_4\}, L\{d_{10}, d_{14}\}\}|*\lambda\rfloor = \lfloor 4 * 0.25\rfloor = 1$ *(linha 8); e iii) portanto, um único bloco* ($Q.pop() = L\{d_{10}, d_{14}\}$) *é excluído da coleção de blocagens(linhas 9-12). Note que a exclusão deste bloco não reduz o resultado de* PC *produzido pela coleção de blocagens podada.*

$\frac{1}{1} = 1$. Portanto, $BlockCooScore(L\{d_{10}, d_{14}\}) = \frac{1}{1}\}$

Apêndice D

Prova de Lemas, Proposições e Teoremas

D.1 Desbalanceamento de Carga Associado à Alocação de Máquinas Virtuais com Configuração Heterogênea

Seja $\mathcal{M} = \{m_1, m_2, \cdots, m_M\}$ um conjunto de máquinas virtuais. A capacidade de processamento e o preço[1] de uma máquina virtual $m \in \mathcal{M}$ é denotada por $Proc(m)$ e $Price(m)$, respectivamente. Sejam $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_{|\mathcal{T}|}\}$ um conjunto de sub tarefas[2] a serem executadas em um ambiente distribuído, $WLoad(\tau)$ a carga de trabalho associada a uma tarefa $\tau \in \mathcal{T}$ e $T_{exec}(\tau, m)$ o tempo de execução de uma tarefa τ utilizando a máquina virtual m. Assim, é assumido o seguinte cenário que emprega máquinas virtuais com configurações heterogêneas: i) Seja $\mathcal{M}_{het} = \{m_1, m_2, \cdots, m_M\}$ um conjunto de máquinas virtuais com configurações heterogêneas, tal que $\forall\langle m_i, m_j \in \mathcal{M}_{het}\rangle(i > j \Rightarrow Proc(m_i) > Proc(m_j))$, i.e., cada máquina virtual em $\mathcal{M}_{het}$ possui uma capacidade de processamento diferente das demais e a capacidade de processamento das máquinas virtuais em $\mathcal{M}_{het}$ é ordenada de maneira crescente pelo índice das máquinas virtuais; e ii) Similarmente, $\forall\langle \tau_i, \tau_j \in \mathcal{T}\rangle(i > j \Rightarrow WLoad(\tau_i) > WLoad(\tau_j))$, i.e., cada tarefa em $\mathcal{T}$ apresenta uma carga de trabalho diferente das demais e as cargas de trabalho associadas às tarefas em $\mathcal{T}$ são ordenada de maneira crescente pelo índice das tarefas.

Considerando a adoção de um algoritmo de alocação guloso (ou seja, as sub tarefas de

[1] Em $ por unidade de tempo

[2] É assumido que a tarefa principal foi subdividida em $|\mathcal{T}|$ sub tarefas.

maior complexidade são alocadas primeiro[3]), no instante de tempo t_1 é assumido que o subconjunto $\mathcal{T}'$ contendo as últimas M tarefas em $\mathcal{T}$ (i.e., $\mathcal{T}' = \{\tau_{k+1}, \tau_{k+2}, \cdots, \tau_{|\mathcal{T}|}\}$, tal que $k = |\mathcal{T}| - M$) ainda precisem ser alocadas. No pior caso, no instante de tempo t_1 apenas a máquina virtual $m_1 = \underset{m \in M_{het}}{arg\,min}\ Proc(m)$ está disponível e, portanto, o par (τ_{k+1}, m_M) é alocado pelo algoritmo de alocação (note que a tarefa de maior complexidade foi alocada à máquina virtual de menor capacidade de processamento). No instante de tempo t_2, apenas a máquina virtual m_2 está disponível e, portanto, o par (τ_{k+2}, m_{M-1}) é alocado pelo algoritmo de alocação. Caso este cenário se repita M vezes, o algoritmo de alocação irá produzir a alocação $\mathcal{A}_{worst} = \bigcup_{i=1}^{M} (\tau_{k+i}, m_{M+1-i})$.

Uma vez que o tempo de execução da tarefa principal é definido pela finalização da execução de todas as sub tarefas (ou seja, o tarefa principal será completada no instante de tempo em que a última máquina virtual finalizar sua execução), então a alocação produzida no pior cenário irá gerar uma perda causada pelo desbalanceamento de carga como mostrado na Eq. (D.1).

$$Loss(\mathcal{A}_{worst}, \mathcal{M}_{het}, \mathcal{T}) = \sum_{i=1}^{M} \Bigg(\overbrace{\big(\underbrace{T_{exec}(\tau_{|\mathcal{T}|}, m_1)}_{\substack{\text{sub tarefa que produz} \\ \text{necessariamente o maior} \\ \text{tempo de execução}}} - T_{exec}(\tau_{k+i}, m_{M+1-i})\big)}^{\substack{\text{parte do desbalanceamento de carga é causado pela} \\ \textbf{diferença nas configurações} \text{ das máquinas virtuais}}} \times Price(m_{M+1-i}) \Bigg) \tag{D.1}$$

Por outro lado, caso o algoritmo de alocação produza uma alocação ótima (i.e., $\mathcal{A}_{opt} = \bigcup_{i=1}^{M} (\tau_{k+i}, m_i)$), não é possível predizer qual máquina virtual irá produzir o maior[4] tempo de execução. Desse modo, seja o par $(\tau', m') = \underset{(\tau', m') \in A_{opt}}{arg\,max}\ T_{exec}(\tau', m')$, então o desbalanceamento de carga gerado pela alocação ótima será calculado com base na diferença entre $T_{exec}(\tau', m')$ e os demais tempos de execução produzidos pelos pares do conjunto $\mathcal{A}_{opt} \backslash (\tau', m')$, como mostrado na Eq. (D.2).

[3]Esta estratégia visa minimizar o desbalanceamento de carga causado pela execução das sub tarefas finais.

[4]Isto porque, se $i > j$, então necessariamente $T_{exec}(\tau_i, m_j) > T_{exec}(\tau_j, m_i)$ (mas não é possível afirmar se $T_{exec}(\tau_i, m_i) < T_{exec}(\tau_j, m_j)$ ou $T_{exec}(\tau_j, m_j) > T_{exec}(\tau_i, m_i)$).

$$Loss(\mathcal{A}_{opt}, \mathcal{M}_{het}, \mathcal{T}) = \sum_{(\tau,m)\in\mathcal{A}_{opt}} \Big(\overbrace{\underbrace{(\underset{(\tau',m')\in A_{opt}}{arg\,max}\ T_{exec}(\tau', m') - T_{exec}(\tau, m))}_{\text{sub tarefa que produz o maior tempo de execução}}}^{\text{parte do desbalanceamento de carga é causado pela } \textbf{diferença nas configurações} \text{ das máquinas virtuais}} \times Price(m) \Big) \tag{D.2}$$

Note que ao empregar um conjunto de máquinas virtuais com configuração heterogênea ($\mathcal{M}_{het}$), os valores de $Loss(\mathcal{A}_{worst}, \mathcal{M}_{het}, \mathcal{T})$ e $Loss(\mathcal{A}_{opt}, \mathcal{M}_{het}, \mathcal{T})$ são ambos influenciados pelas diferenças nos tempos de execução causados pelas diferentes configurações de máquina virtual adotadas. Além disso, a adoção de $\mathcal{M}_{het}$ implica necessariamente que $Loss(\mathcal{A}_{worst}, \mathcal{M}_{het}, \mathcal{T}) > Loss(\mathcal{A}_{opt}, \mathcal{M}_{het}, \mathcal{T})$. Ou seja, quanto pior a qualidade do conjunto de alocação produzido, maior será a perda gerada pelo desbalanceamento de carga.

Por sua vez, assumindo o seguinte cenário que emprega máquinas virtuais com configurações homogêneas: seja $\mathcal{M}_{hom} = \{m_1, m_2, \cdots, m_M\}$ um conjunto de máquinas virtuais com configurações heterogêneas, tal que $\forall\langle m_i, m_j \in \mathcal{M}_{hom}\rangle(Proc(m_i) = Proc(m_j))$, ou seja, todas as máquinas virtuais em M_{hom} possuem a mesma configuração. Desse modo, independente do conjunto de alocação produzido pelo algoritmo de alocação, será gerada uma perda causada pelo desbalanceamento de carga como mostrado na Eq. (D.3).

$$Loss(\mathcal{A}, \mathcal{M}_{hom}, \mathcal{T}) = \sum_{(\tau,m)\in\mathcal{A}} \Big(\overbrace{\underbrace{(\underset{(\tau',m')\in A}{arg\,max}\ T_{exec}(\tau', m') - T_{exec}(\tau, m))}_{\text{sub tarefa que produz o maior tempo de execução}}}^{\text{o desbalanceamento de carga é causado } \textbf{unicamente} \text{ pelas } \textbf{diferentes cargas de trabalho} \text{ das sub tarefas} } \times Price(m) \Big) \tag{D.3}$$

Portanto, ao empregar um conjunto de máquinas virtuais com configuração homogênea ($\mathcal{M}_{hom}$), para qualquer alocação $\mathcal{A}$ produzida pelo algoritmo de alocação, a perda produzida pelo desbalanceamento de carga será causado unicamente pelas diferentes cargas de trabalho das sub tarefas do conjunto $\mathcal{T}$, como mostrado na Eq. (D.3).

D.2 Convergência do Algoritmo Hill Climbing Estimation

Sejam τ uma tarefa de deduplicação de dados, thr um limiar (positivo) mínimo utilizado para avaliar alocações classificadas como super provisão, e $L = |\mathcal{C}_{conf}|$ o índice da última classe de configuração disponível (parâmetros de entrada do algoritmo). O algoritmo *Hill Climbing Estimation* (Algoritmo 2) visa encontrar a classe de configuração ótima ($\mathcal{C}l_{opt}(\tau)$) para a tarefa τ, i.e. $T_{exec}^{\mathcal{C}l_{opt}(\tau)} \leq T_{res}(\tau) \wedge \forall\langle \mathcal{C}l' \in \mathcal{C}_{conf} \backslash \mathcal{C}_{opt}(\tau)\rangle(|T_{exec}^{\mathcal{C}l_{opt}(\tau)} - T_{res}(\tau)| \leq |T_{exec}^{\mathcal{C}l'(\tau)} - T_{res}(\tau)|)$, aplicando a técnica de busca Hill Climbing.

Desse modo, o algoritmo estima uma classe de configuração inicial ($\mathcal{C}l_i$) para executar a tarefa τ (linha 4). Caso a alocação da classe de configuração inicial para executar a tarefa τ produza sub provisão ou super provisão de recursos (i.e., a expressão da linha 8 é avaliada como verdadeiro), então a classe de configuração é ajustada para $\mathcal{C}l_T$ (linha 10) caso o DQSLA que disparou a tarefa τ dispare uma nova tarefa.

O ajuste para calcular o índice da classe de configuração $\mathcal{C}l_T$ é sempre realizado da seguinte forma: i) se a execução de τ produziu sub provisão de recursos (i.e., $T_{exec}^{\mathcal{C}l_i(\tau)} > T_{res}(\tau)$), então a classe de configuração ótima de τ está necessariamente entre os índices $i+1$ e $|\mathcal{C}_{conf}|$ e, portanto, o índice da classe $\mathcal{C}l_T$ é ajustada para algum valor $T \in \{i+1, 2, \cdots, |\mathcal{C}_{conf}|\}$; caso contrário, se ii) se a execução de τ produziu super provisão de recursos (i.e., $(T_{res}(\tau) - T_{exec}^{\mathcal{C}l_i(\tau)}) > thr$), então a classe de configuração ótima de τ está necessariamente entre os índices 1 e $i-1$ e, portanto,o índice da classe $\mathcal{C}l_T$ é ajustada para algum valor $T \in \{1, 2, \cdots, i-1\}$.

Seguindo o funcionamento do algoritmo, se $\mathcal{C}l_i(\tau) \neq \mathcal{C}l_{opt}(\tau)$, então caso o DQSLA que disparou a tarefa τ dispare uma nova execução, o algoritmo irá utilizar um índice de configuração ajustado $\mathcal{C}l_{T_1}(\tau)$, tal que $T_1 \in \{1, 2, \cdots, i-1\}$ ou $T_1 \in \{i+1, i+2, \cdots, |\mathcal{C}_{conf}|\}$. Prosseguindo, se $\mathcal{C}l_{T_1}(\tau) \neq \mathcal{C}l_{opt}(\tau)$, então caso o DQSLA que disparou a tarefa τ dispare uma nova execução, o algoritmo irá utilizar um índice de configuração ajustado $\mathcal{C}l_{T_2}(\tau)$, tal que: i) se $T_1 \in \{i+1, i+2, \cdots, |\mathcal{C}_{conf}|\}$, então $T_2 \in \{i+1, 2, \cdots, T_1 - 1\}$ ou $T_2 \in \{T_1+1, T_1+2, \cdots, |\mathcal{C}_{conf}|\}$; caso contrário, ii) se $T_1 \in \{1, 2, \cdots, i-1\}$, então $T_2 \in \{1, 2, \cdots, T_1 - 1\}$ ou $T_2 \in \{T_1+1, T_1+2, \cdots, i-1\}$. Esse processo é repetido k vezes para ajustar as classes de configuração subsequentes até a convergência do algoritmo, i.e., $\mathcal{C}l_{T_k}(\tau) = \mathcal{C}l_{opt}(\tau)$.

Note que após cada ajuste, no pior caso, o algoritmo diminui em um o espaço de busca em que se encontra a classe de configuração ótima $\mathcal{C}l_{opt}(\tau)$. Portanto, se a classe de configuração inicial estimada não for a classe de configuração ótima, i.e. $\mathcal{C}l_i(\tau) \neq \mathcal{C}l_{opt}(\tau)$, então no pior caso (($i = 1$ e $opt = |\mathcal{C}_{conf}|$) ou ($i = |\mathcal{C}_{conf}|$ e $opt = 1$)) o algoritmo irá encontrar necessariamente a classe de configuração ótima após $|\mathcal{C}_{conf}|-1$ ajustes na classe de configuração.

D.3 Prova de Lemas e Teoremas Relacionados ao Problema de Provisionamento de Recursos Computacionais em um SMQD

Lema D.3.1 *Sejam* $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_{|\mathcal{T}|}\}$ *um conjunto de tarefas de qualidade de dados,* $\mathcal{C}_{conf} = (N \times \gamma) = \{\mathcal{C}l_1, \mathcal{C}l_2, \cdots, \mathcal{C}l_{m\times k}\}$ *um conjunto de classes de configuração e* alg' *um algoritmo de provisionamento. Então:*

$\exists\tau \in \mathcal{T}(alg'(\tau) \in (\mathcal{C}_{conf} \backslash \mathcal{C}l_{opt}(\tau)) \ \wedge \ T_{exec}^{alg'(\tau)}(\tau) < T_{res}(\tau)) \Rightarrow$
$\sum_{\tau\in\mathcal{T}}(C\$(\tau, alg'(\tau))) > \sum_{\tau\in\mathcal{T}}(C\$(\tau, \mathcal{C}l_{opt}(\tau)))$.

Prova D.3.1 $(alg'(\tau) \in (\mathcal{C}_{conf} \backslash \mathcal{C}l_{opt}(\tau)) \wedge T_{exec}^{alg'(\tau)}(\tau) < T_{res}(\tau)) \Rightarrow T_{exec}^{\mathcal{C}l_{opt}(\tau)}(\tau) > T_{exec}^{alg'(\tau)}(\tau)$*, caso contrário teríamos* $alg'(\tau) = \mathcal{C}l_{opt}(\tau)$*. Uma vez que* $T_{exec}^{\mathcal{C}l_{opt}}(\tau) > T_{exec}^{alg'(\tau)}(\tau)$*, então* $Pr(alg'(\tau)) > Pr(\mathcal{C}l_{opt}(\tau))$*. Assim,* $VMC\$(\tau, alg'(\tau))$ *tende a ser maior que* $VMC\$(\tau, \mathcal{C}l_{opt}(\tau))$ *(porque ainda que* $T_{exec}^{\mathcal{C}l_{opt}}(\tau) > T_{exec}^{alg'(\tau)}(\tau)$*, quanto maior a capacidade de processamento do cluster, maior tende a ser a diferença entre o valor de* $speedup$ *e a quantidade de máquinas virtuais alocadas).*

Além disso, note que $PenaltyC\$(\tau, alg'(\tau)) = 0$ *(porque* $T_{exec}^{alg'(\tau)}(\tau) < T_{res}(\tau)$*) e* $PenaltyC\$(\tau, \mathcal{C}l_{opt}(\tau)) = 0$ *(por definição). Como mostrado na Eq. (5.2), uma vez* $C\$(\tau, \mathcal{C}l(\tau)) = VMC\$(\tau, \mathcal{C}l(\tau)) + PenaltyC\$(\tau, \mathcal{C}l(\tau))$*, é possível concluir que:*

$\forall\tau \in \mathcal{T}[(alg'(\tau) \in (\mathcal{C}_{conf} \backslash \mathcal{C}l_{opt}(\tau)) \ \wedge \ T_{exec}^{alg'(\tau)}(\tau) < T_{res}(\tau)) \Rightarrow$
$C\$(\tau, alg'(\tau)) > C\$(\tau, \mathcal{C}l_{opt}(\tau))]$.

Como consequência,

$\sum_{\tau\in\mathcal{T}}(C\$(\tau, alg'(\tau))) > \sum_{\tau\in\mathcal{T}}(C\$(\tau, \mathcal{C}l_{opt}(\tau)))$.

Lema D.3.2 *Sejam* $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_{|\mathcal{T}|}\}$ *um conjunto de tarefas de qualidade de dados,* $\mathcal{C}_{conf} = (N \times \gamma) = \{\mathcal{C}l_1, \mathcal{C}l_2, \cdots, \mathcal{C}l_{m \times k}\}$ *um conjunto de classes de configuração e* alg' *um algoritmo de provisionamento. Então:*

$\exists \tau \in \mathcal{T}(alg'(\tau) \in (\mathcal{C}_{conf} \backslash \mathcal{C}l_{opt}(\tau)) \ \wedge \ T_{exec}^{alg'(\tau)}(\tau) > T_{res}(\tau)) \Rightarrow$
$\sum_{\tau \in \mathcal{T}}(C\$(\tau, alg'(\tau))) > \sum_{\tau \in \mathcal{T}}(C\$(\tau, \mathcal{C}l_{opt}(\tau)))$.

Prova D.3.2 $(alg'(\tau) \in (\mathcal{C}_{conf} \backslash \mathcal{C}l_{opt}(\tau)) \ \wedge \ T_{res}(\tau) < T_{exec}^{alg'(\tau)}(\tau)) \Rightarrow$
$PenaltyC\$(\tau, alg'(\tau)) > 0$.

Uma vez que $PenaltyC\$(\tau, \mathcal{C}l_{opt}(\tau)) = 0$ *(por definição), então* $PenaltyC\$(\tau, alg'(\tau)) > PenaltyC\$(\tau, \mathcal{C}l_{opt}(\tau))$. *Outrossim, ainda que* $VMC\$(\tau, alg'(\tau)) < VMC\$(\tau, \mathcal{C}l_{opt}(\tau))$ *(porque* $T_{exec}^{alg'(\tau)}(\tau) > T_{exec}^{\mathcal{C}l_{opt}}(\tau)$ *e, portanto,* $Pr(\mathcal{C}l_{opt}(\tau)) > Pr(alg'(\tau))$*), o valor da soma* $(VMC\$(\tau, alg'(\tau)) + PenaltyC\$(\tau, alg'(\tau)))$ *tende a ser maior que o valor de* $C\$(\tau, \mathcal{C}l_{opt}(\tau))$ *porque* $\mathcal{C}l_{opt}(\tau)$ *produz* $(T_{res}(\tau) \approx T_{exec}^{\mathcal{C}l_{opt}}(\tau))$ *e* $PenaltyC\$(\tau, \mathcal{C}l_{opt}(\tau)) = 0$. *Estes resultados são especialmente potencializados quando* $\alpha(\tau)$ *(usado na Eq. (5.3)) possui um valor significante ou* $T_{exec}^{alg'(\tau)}(\tau) \gg T_{exec}^{\mathcal{C}l_{opt}(\tau)}(\tau)$. *Uma vez que* $C\$(\tau, \mathcal{C}l(\tau)) = PenaltyC\$(\tau, \mathcal{C}l(\tau)) + VMC\$(\tau, \mathcal{C}l(\tau))$*, é possível concluir que:*

$\forall \tau \in \mathcal{T}[(alg'(\tau) \in (\mathcal{C}_{conf} \backslash \mathcal{C}l_{opt}(\tau)) \ \wedge \ T_{res}(\tau) < T_{exec}^{alg'(\tau)}(\tau)) \Rightarrow$
$C\$(\tau, alg'(\tau)) > C\$(\tau, \mathcal{C}l_{opt}(\tau))]$.

Como consequência,

$\sum_{\tau \in \mathcal{T}}(C\$(\tau, alg'(\tau))) > \sum_{\tau \in \mathcal{T}}(C\$(\tau, \mathcal{C}l_{opt}(\tau)))$.

Teorema D.3.1 *Sejam* $\mathcal{T}\langle t_i, t_f\rangle = \{\tau_1, \tau_2, \cdots, \tau_{|\mathcal{T}\langle t_i, t_f\rangle|}\}$ *um conjunto de tarefas de qualidade de dados disparadas por DQSLAs de clientes representados pelo conjunto* $C\langle t_i, t_f\rangle$, $\mathcal{C}_{conf} = (N \times \gamma) = \{\mathcal{C}l_1, \mathcal{C}l_2, \cdots, \mathcal{C}l_{m \times k}\}$ *um conjunto de classes de configuração, e* alg' *e* alg_{opt} *algoritmos de provisionamento. Então:*

$[\forall \tau \in \mathcal{T}\langle t_i, t_f\rangle(alg_{opt}(\tau) = \mathcal{C}l_{opt}(\tau)) \ \wedge \ \exists \tau \in \mathcal{T}(alg'(\tau) \in \mathcal{C}_{conf} \backslash \mathcal{C}l_{opt}(\tau))] \Rightarrow ServC\$_{alg'}^{C\langle t_i, t_f\rangle} > ServC\$_{alg_{opt}}^{C\langle t_i, t_f\rangle}$.

Prova D.3.3 $\forall \tau \in \mathcal{T}\langle t_i, t_f\rangle[(alg'(\tau) \in (\mathcal{C}_{conf} \backslash \mathcal{C}l_{opt}(\tau))) \Rightarrow (T_{exec}^{alg'(\tau)}(\tau) < T_{res}(\tau) \vee T_{exec}^{alg'(\tau)}(\tau) > T_{res}(\tau))]$*, caso contrário teríamos* $alg'(\tau) = \mathcal{C}l_{opt}(\tau)$.

Então, $\exists \tau \in \mathcal{T}\langle t_i, t_f\rangle((alg'(\tau) \in (\mathcal{C}_{conf} \backslash \mathcal{C}l_{opt}(\tau)) \ \wedge \ T_{exec}^{alg'(\tau)}(\tau) < T_{res}(\tau)) \Rightarrow ServC\$_{alg'}^{C\langle t_i, t_f\rangle} > ServC\$_{alg_{opt}}^{C\langle t_i, t_f\rangle}$ *(pelo lema D.2.1).*

Outrossim, $\exists \tau \in \mathcal{T}\langle t_i, t_f\rangle((alg'(\tau) \in (\mathcal{C}_{conf} \backslash \mathcal{C}l_{opt}(\tau)) \ \wedge \ T_{exec}^{alg'(\tau)}(\tau) > T_{res}(\tau)) \Rightarrow$

$ServC\$_{alg'}^{C\langle t_i,t_f\rangle} > ServC\$_{alg_{opt}}^{C\langle t_i,t_f\rangle}$ *(pelo Lema D.2.2).*

Portanto (por Silogismo Hipotético),

$[\forall\tau \in \mathcal{T}\langle t_i,t_f\rangle(alg_{opt}(\tau) = \mathcal{C}l_{opt}(\tau)) \wedge \exists\tau \in \mathcal{T}(alg'(\tau) \in (\mathcal{C}_{conf}\backslash\mathcal{C}l_{opt}(\tau)))] \Rightarrow$
$ServC\$_{alg'}^{C\langle t_i,t_f\rangle} > ServC\$_{alg_{opt}}^{C\langle t_i,t_f\rangle}$.

D.4 Desigualdade Relacionada à Precisão de Agrupamentos

Proposição D.4.1 *Sejam* $a \in \mathbf{N}^*$ *e* $B = \{b_1, b_2, \ldots, b_{|B|}\}$, *tal que* $\forall\langle b \in B\rangle(b \in \mathbf{N}^*)$, $|B| > 1$ *e* $\sum_{b\in B} b = a$. *Então:* $\forall\langle a \in \mathbf{N}^*\rangle(a(a-1) > \sum_{b\in B} b(b-1))$.

Prova D.4.1 *Primeiramente, note que* $\forall\langle a \in \mathbf{N}^*\rangle(a > \frac{\sum_{b\in B} b(b-1)}{a} + 1 \Rightarrow a(a-1) > \sum_{b\in B} b(b-1))$.
Além disso, $\frac{\sum_{b\in B} b(b-1)}{a} + 1 = \frac{\sum_{b\in B} b^2-b}{a} + 1 = \frac{\sum_{b\in B} b^2}{a} - \frac{\sum_{b\in B} b}{a} + 1 = \frac{\sum_{b\in B} b^2}{a} - 1 + 1 = \frac{\sum_{b\in B} b\times b}{a} = \sum_{b\in B} \frac{b}{a} \times b$.
Uma vez que $|B| > 1$ *e* $\sum_{b\in B} b = a$, *então* $\forall\langle b \in B\rangle(a > b)$, *logo,* $\frac{b}{a} < 1$. *Assim,* $\forall\langle a \in \mathbf{N}^*\rangle(\sum_{b\in B} \frac{b}{a} \times b < a)$.
Portanto, uma vez que $\forall\langle a \in \mathbf{N}^*\rangle(a > \sum_{b\in B} \frac{b}{a} \times b)$ *e* $\frac{\sum_{b\in B} b(b-1)}{a} + 1 = \sum_{b\in B} \frac{b}{a} \times b$, *então* $\forall\langle a \in \mathbf{N}^*\rangle(a(a-1) > \sum_{b\in B} b(b-1))$.

Apêndice E

Regras de Similaridade e Funções de Chave de Bloco

E.1 Experimento III

Sejam Jw a função de similaridade $Jaro\ Winkler$ e $avg\_jw\_cora(r_1, r_2)$ uma função calculada como ($Jw(title(r_1)$, $title(r_2))$ + $Jw(authors(r_1)$, $authors(r_2))$ + $Jw(location(r_1)$, $location(r_2))$)/3. Para a base de dados $Cora$, foi empregada as seguintes regras para computar o valor de similaridade entre duas entidades distintas (r_1,r_2):

$$sim\_cora(r_1, r_2) = \begin{cases} 0 & \textbf{se } year(r_1) \neq year(r_2) \\ avg\_jw\_cora(r_1, r_2) & \textbf{se } year(r_1) = year(r_2) \end{cases}$$

Por sua vez, para a base de dados $DBLPM4$, foi empregada a seguinte regra de similaridade para computar o valor de similaridade entre entidades distintas (r_1,r_2): $Jw(title(r_1), title(r_2))$.

Por fim, na base de dados $Febrl10k$ foi empregada a técnica de indexação blocagem padrão [14, 15] utilizando o valor do atributo $state$ de cada entidade como o valor de chave de bloco. Além disso, os blocos que foram gerados com apenas uma entidade foram eliminados do experimento. Sejam $avg\_nm\_Jw(r_1, r_2)$ uma função calculada como $(Jw(name(r_1), name(r_2)) + Jw(surname(r_1), surname(r_2)))/2$ e $avg\_nm\_addr\_Jw(r_1, r_2)$ uma função calculada como $(Jw(name(r_1), name(r_2)) +$

Tabela E.1: Funções de chave de bloco empregadas no Experimento IV (F_{key}).

Base de Dados	**Funções de Chave de Bloco**
CoraATDV	$\{4B(SplitLine[0]), 4B(SplitLine[1]), 2B(SplitLine[2]), 2B(SplitLine[3])\}$
DBLPM4	$\{F_1(title), 4B(title)\}$
DS3	$\{F_1(title), 2T(title), 2T(authors), 2T(venue), year\}$
FebrlData	$\{1T(name), 1T(surname), 1T(address_1)\}$

$Jw(surname(r_1), surname(r_2)) + Jw(street(r_1), street(r_2)))/3$. Desse modo, foram empregadas as seguintes regras de similaridade para computar o valor similaridade entre entidades distintas (r_1,r_2):

$$sim\_febrl10k(r_1, r_2) = \begin{cases} 0 & \textbf{se } name(r_1) = nil \vee name(r_2) = nil \\ avg\_nm\_Jw(r_1, r_2) & \textbf{se } street(r_1) = nil \vee street(r_2) = nil \\ avg\_nm\_addr\_Jw(r_1, r_2) & \text{caso contrário} \end{cases}$$

E.2 Experimento IV

Sejam S uma sequência de caracteres, $F_n(S)$ os n primeiros caracteres de S, $nB(S)$ os n primeiros bigramas [12] de S e $nT(S)$ os n primeiro trigramas [12] de S. Usando esta notação, na Tabela E.1 são apresentadas as funções de chave de bloco que foram empregadas para indexar as bases de dados utilizadas no no Experimento IV (Capítulo 8). Uma vez que as entidades na base de dados Cora CoraATDV não seguem um esquema bem definido, os valores das entidades foram divididos por espaços em branco. Cada parte do resultado é denotado como $SplitLine[k]$, tal que k é o índice do vetor resultante.

Apêndice F

Tamanho da Janela Fixa Empregado pelo Algoritmo *Sorted Neighborhood*

Sejam S_{min} ed S_{max} dois números inteiros (representando um intervalo de tamanho de blocos), $\mathcal{F}$ um conjunto de chaves de bloco e $\mathcal{B}$ uma coleção de blocagens produzida pela indexação das entidades na base de dados D com base no conjunto $\mathcal{F}$. Levando em consideração o problema apresentado na Definição 7.3.6, a coleção de blocos podada $\mathcal{B}'_{max}$ que produz a cardinalidade agregada máximo é composta por ($|D|\cdot|\mathcal{F}|$ div S_{max}) blocos contendo S_{max} entidades e um bloco contendo ($|D|\cdot|\mathcal{F}|$ mod S_{max}) entidades. Seguindo a Definição 7.3.1, é possível calcular a cardinalidade agregada ($||\mathcal{B}'_{max}||$) produzida por $\mathcal{B}'_{max}$ como mostrado na Eq. (F.1).

$$||\mathcal{B}'_{max}|| = |\mathcal{F}|\cdot(|D| \text{ div } S_{max}) \cdot S_{max} \cdot (S_{max} - 1) + |\mathcal{F}|\cdot(|D| \text{ mod } S_{max}) \cdot ((|D| \text{ mod } S_{max}) - 1) \tag{F.1}$$

Desse modo, ao empregar o algoritmo *Sorted Neighborhood* (SN) [12] utilizando uma janela fixa de tamanho w sobre os blocos em $\mathcal{B}$, a quantidade de comparações produzida pelo algoritmo SN não pode exceder o valor da cardinalidade agregada máxima $||\mathcal{B}'_{max}||$ (Eq. (F.1)). Uma vez que o algoritmo SN é executado $|\mathcal{F}|$ vezes (i.e., para cada função de chave de bloco empregada), o número de comparações produzido pelo algoritmo SN para cada função de chave de bloco não pode exceder $\frac{||\mathcal{B}'_{max}||}{|\mathcal{F}|}$. Portanto, para cada execução do algoritmo SN, é preciso configurar um tamanho da janela fixa (w) que produza no máximo

$\frac{||\mathcal{B}'_{max}||}{|\mathcal{F}|}$ comparações entre entidades.

Uma vez que a quantidade de comparações produzida pelo algoritmo SN pode ser estimada teoricamente [13], é possível calcular o valor máximo permitido (w_{max}) para o tamanho da janela fixa como mostrado na Eq. (F.2).

$$\begin{aligned}
(w-1)\cdot(|D|-\frac{w}{2}) &= \frac{||\mathcal{B}'_{max}||}{|\mathcal{F}|} \\
w\cdot|D|-\frac{w^2}{2}-|D|+\frac{w}{2} &= \frac{||\mathcal{B}'_{max}||}{|\mathcal{F}|} \\
-\frac{w^2}{2}+\frac{(2\cdot w\cdot|D|)+w}{2}-\frac{||\mathcal{B}'_{max}||}{|\mathcal{F}|}-|D| &= 0 \\
w^2-(2\cdot|D|+1)\cdot w+2\cdot\left(\frac{||\mathcal{B}'_{max}||}{|\mathcal{F}|}+|D|\right) &= 0
\end{aligned} \tag{F.2}$$

A partir da Eq. (F.2), é possível concluir que o valor máximo da janela fixa (w_{max}) pode ser calculado como mostrado na Eq. (F.3), i.e., o valor máximo de w que pode ser empregado pelo algoritmo SN para produzir até $\frac{||\mathcal{B}'_{max}||}{|\mathcal{F}|}$ comparações entre entidades.

$$w_{max}=\frac{(2\cdot|D|+1)+\sqrt{(-2\cdot|D|+1)^2-4\cdot2\cdot\left(\frac{||\mathcal{B}'_{max}||}{|\mathcal{F}|}+|D|\right)}}{2} \tag{F.3}$$

Note que, uma vez que a abordagem proposta no Capítulo 7 para controlar o tamanho dos blocos pode executar operações de $Shrink$ e $Exclude$ sobre os blocos de entrada, não é possível calcular a quantidade mínima de comparações produzida pela coleção de blocagens podada ($\mathcal{B}'$). Por esta razão, para cada base de dados, são empregadas duas variações do valor máximo do tamanho de janela w_{max} (com base na Eq. (F.3)).