GILBERTO VIANA DE OLIVEIRA

ESTUDO E DESENVOLVIMENTO DE META-HEURÍSTICAS EVOLUTIVAS ESCALÁVEIS PARA AGRUPAMENTO DE DADOS

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA
MINAS GERAIS – BRASIL
2016

Ficha catalográfica preparada pela Biblioteca Central da Universidade Federal de Viçosa - Câmpus Viçosa

T
O48e
2016

Oliveira, Gilberto Viana de, 1988-
Estudo e desenvolvimento de meta heurística evolutivas escaláveis para agrupamentos de dados / Gilberto Viana de Oliveira. – Viçosa, MG, 2016.
x, 56f. : il. ; 29 cm.

Orientador: Murilo Coelho Naldi.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f.52-56.

1. Algoritmos. 2. Heurística. 3. Otimização combinatória.
I. Universidade Federal de Viçosa. Departamento de Informática.
Programa de Pós-graduação em Ciência da Computação.
II. Título.

CDD 22. ed. 005.1

GILBERTO VIANA DE OLIVEIRA

ESTUDO E DESENVOLVIMENTO DE META-HEURÍSTICAS EVOLUTIVAS ESCALÁVEIS PARA AGRUPAMENTO DE DADOS

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 26 de fevereiro de 2016.

_____	_____
Ricardo José G. Barreto Campello	Fábio Ribeiro Cerqueira

_____
Murilo Coelho Naldi
Orientador

À minha família e aos meus amigos por me fazerem acreditar que sou capaz.

"A persistência é o menor caminho do êxito."
(Charles Chaplin)

Agradecimentos

Agradeço primeiramente aos meus pais, pelo apoio e incentivo incondicional em todos os momentos da minha vida, e pela força durante essa jornada que vivi.

Aos meus amigos Marques Moreira, Jefferson Lopo e Rafael Sampaio por toda a confiança e ajuda ao longo deste longo caminho.

À minha namorada Marina Alvarenga ofereço um agradecimento especial, por ter me dado todo o apoio que necessitava nos momentos difíceis, todo carinho, respeito e amor.

Aos professores, que se dedicaram tanto para passar o conhecimento adiante e formar uma base sólida para cada um de nós.

Ao professor Fábio Cerqueira pela ajuda e orientação durante o tempo que estive em Viçosa.

Agradeço ao meu orientador Murilo Naldi, por sempre me auxiliar na solução dos problemas e por acreditar que seríamos capazes de realizar esta pesquisa mesmo em meio a tantas dificuldades.

Agradeço ao financiamento fornecido pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), ao qual foi essencial para que pudesse me dedicar exclusivamente a pesquisa e a escrita dessa dissertação.

Agradeço a Universidade Federal de Viçosa e ao Departamento de Informática por fornecerem o suporte necessário para adquirir o título de mestre e o conhecimento para atuar como educador.

E finalmente, a todos aqueles que, direta ou indiretamente, contribuíram para a execução deste trabalho, os meus sinceros agradecimentos.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

OLIVEIRA, Gilberto Viana de, M.Sc., Universidade Federal de Viçosa, Fevereiro de 2016. **Estudo e Desenvolvimento de Meta-heurísticas Evolutivas Escaláveis para Agrupamento de Dados**. Orientador: Murilo Coelho Naldi.

A cada dia mais dados são gerados das mais diversas fontes. A extração de conhecimento das bases de dados torna-se cada vez mais desafiadora, visto que os processos utilizados não são triviais. O agrupamento de dados usa técnicas que são capazes de trabalhar com dados pouco conhecidos de forma não supervisionada. Essas técnicas dividem os dados em grupos tentando capturar a estrutura presente nos dados para obter um conhecimento que servirá de ponto inicial para seu estudo. Poucos algoritmos de agrupamentos conseguem trabalhar em um contexto escalável. Um dos algoritmos mais influentes no agrupamento é o k-médias, que possui complexidade linear e duas fases bem distintas, facilmente adaptada para modelos escaláveis. Porém, k-médias possui limitações, como sensibilidade à inicialização e especificação do número de grupos k, que geralmente é desconhecido. O objetivo desta pesquisa é estudar e desenvolver algoritmos de agrupamento para este contexto escalável. Especificamente, procura-se trabalhar com meta-heurísticas que proporcionem o agrupamento escalável sem a necessidade de especificação do número de grupos k. Essa dissertação propõe dois novos algoritmos de agrupamento que encontram um valor para k automaticamente em um modelo escalável chamado *MapReduce*. Adicionalmente, foi estudado um algoritmo com o mesmo propósito encontrado na literatura. Todos os algoritmos foram desenvolvidos e comparados de duas maneiras: pela sua complexidade assintótica e através de experimentos em bases artificiais e reais. Com base em testes estatísticos, foi possível verificar as principais diferenças entre a performance dos algoritmos.

Abstract

OLIVEIRA, Gilberto Viana de, M.Sc., Universidade Federal de Viçosa, February of 2016. **Study and Development of Scalable Evolutionary Metaheuristics for Data Clustering.** Adviser: Murilo Coelho Naldi.

Everyday more data are generated from several sources. The knowledge extraction from datasets becomes more and more challenging as the applied techniques are not trivial. Data clustering techniques are able to work with little knowledge about the data in a totally unsupervised manner. These techniques divide data into clusters trying to capture the structure of the data to obtain knowledge that will serve as a starting point for further studies. Few clustering algorithms are able to work in a scalable scenario. One of the most influential clustering algorithms is k-means, which has linear asymptotic complexity and two distinct phases, which can be easily adapted for scalable models. However, k-means has limitations such as sensitivity to initialization and previous specification of the numbers of clusters k, which is generally unknown, specially for real world scenarios. The objective of this research is to study and develop scalable clustering algorithms. Specifically, the use of meta-heuristics for scalable clustering to automatically determine the number of k clusters. This dissertation proposes two new clustering algorithms that are able to automatically find the value k in a scalable programing model called MapReduce. Additionally, an state-of-art algorithm from the literature has been studied and compared. All algorithms were developed and compared in two ways: based on their asymptotic complexity and through experiments in artificial and real datasets. Based on statistical tests, is was possible to find the main differences among quality and performance of all compared algorithms.

Capítulo 1

INTRODUÇÃO

A tecnologia se mantém em constante evolução. Isto traz mudanças no dia a dia das pessoas, direta ou indiretamente. Tal evolução traz consigo uma produção massiva de dados, o que acarreta diversos desafios, já que estes dados não devem apenas ficar armazenados, mas devem ser interpretados de forma útil. Com isso, surge a necessidade de desenvolver e aplicar técnicas que possam transformar esses dados em informação e conhecimento [Chen et al., 1996].

Segundo Chen et al. [1996] o processo de extração não trivial de informações implícitas, previamente desconhecidas e potencialmente úteis denomina o termo que conhecemos como mineração de dados. Uma das técnicas de extração de informação em mineração de dados é o agrupamento de dados ou clusterização (termo advindo do inglês *clustering*) que tem como objetivo classificar objetos de um conjunto de dados de maneira não supervisionada.

Métodos não supervisionados usam técnicas para dividir os dados em grupos sem a necessidade de conhecimento prévio sobre os objetos de estudo [Jain & Dubes, 1988]. Esse tipo de classificação é útil quando não se tem conhecimento sobre os possíveis grupos existentes nos dados, como podemos encontrar em aplicações reais. No agrupamento, os dados são divididos entre grupos que devem capturar a estrutura dos dados, ou seja, os grupos são formados usando apenas as informações encontradas nos dados que descrevem os objetos e seus relacionamentos [Tan et al., 2009].

O agrupamento de dados pode ser usado em diversas áreas como: bioinformática [Suarez et al., 2013], [O'Driscoll et al., 2013], [Müller et al., 2008], processamento de imagens [Ducournau et al., 2012], análise de redes sociais [Narasimhamurthy et al., 2010], estudos meteorológicos [Strauss et al., 2013], entre outras aplicações [Jain et al., 1999; Xu & Wunsch, 2005]. Além disso, existem projetos que estu-

dam técnicas para adaptação e criação de algoritmos de aprendizado de máquina (incluindo agrupamento) para grandes quantidades de dados, como por exemplo Apache Mahout [Owen et al., 2011] e *Machine Learning Library* (MLlib) presente no Apache Spark [Hamstra et al., 2015].

1.1 O problema e sua importância

O volume de dados tem crescido bastante a cada ano acompanhando os avanços da tecnologia. Isso ocorre devido à criação de fontes de dados e ao aumento da complexidade desses dados. Dados são gerados em todos os lugares, setores, economias, organizações e por todas as pessoas que são usuários de tecnologia [Manyika et al., 2011]. Em 2008, o *New York Stock Exchange* gerou em torno de um *terabyte* de dados comerciais por dia. No mesmo ano, o *Facebook* hospedou mais de 10 bilhões de fotos, ocupando um *pettabyte* de espaço dedicado às imagens [White, 2012]. Em 2010, apenas cerca de 10% da população possuía *smartphones*, número que só tende a crescer a cada ano [Manyika et al., 2011].

Além do aumento da quantidade de dados, eles também possuem diferentes formatos, o que aumenta a complexidade do problema em sua captura, armazenagem e gerenciamento. Nesse novo contexto surgem termos como *big data* e *data science*.

Por uma falta de definição concreta, o termo *big data* não tem sido muito usado na academia, mas para entendimento geral, ele pode ser definido para conceituar grandes quantidades de dados, análises de mídias sociais, gerenciamento de recursos para próxima geração, análises em tempo real [Schroeck et al., 2012].

Já o termo *data science*, ou ciência dos dados, pode ser definido como um campo multidisciplinar (que envolve conhecimentos de matemática, estatística, computação e aprendizado de máquina) com foco no estudo da extração de conhecimento dos dados [Schutt & O'Neil, 2013].

Devido ao grande aumento na quantidade de dados gerados, um modelo de programação voltado para lidar com estes volumes foi proposto por Dean & Ghemawat [2004]. Tal modelo, chamado *MapReduce*, é uma abstração inspirada nas primitivas *map* e *reduce* das linguagens funcionais e permite uma programação distribuída e escalável de maneira simplificada, retirando do programador a preocupação com distribuição das operações na rede, sincronização, manipulação e controle de falhas, por exemplo. O modelo foi projetado para funcionar em milhares de máquinas através de uma rede. Entretanto, o modelo restringe os algoritmos a dividirem suas tarefas entre duas funções principais, chamadas *map* e *reduce*, e impõe um fluxo

de execução para as mesmas. O modelo executa funções *map* sobre pares de valores *chave/valor* para o processamento dos dados. Estes dados são combinados em conjuntos de *chave/valor* intermediários para posteriormente serem resolvidos por funções *reduce* através de suas *chaves* [White, 2012].

A Figura 1.1 exemplifica o uso de um contador de palavras no modelo *MapReduce*. Primeiro, temos como entrada um conjunto de dados que é dividido entre os computadores da rede. Cada computador recebe um subconjunto de dados. Durante a etapa *map*, cada palavra é processada individualmente, tendo como saída um par de valores *palavra/contador* (inicialmente cada *map* só conta uma única palavra, e o contador será sempre igual a 1). Durante a próxima etapa, *shuffle*, os valores com mesma *chave* são combinados em listas intermediárias, que posteriormente são processadas pelas funções *reduce*[1]. Cada *chave* então é enviada a uma função *reduce* que emite o resultado final, sendo então armazenado em um Sistema de Arquivos Distribuídos (DFS) [White, 2012].

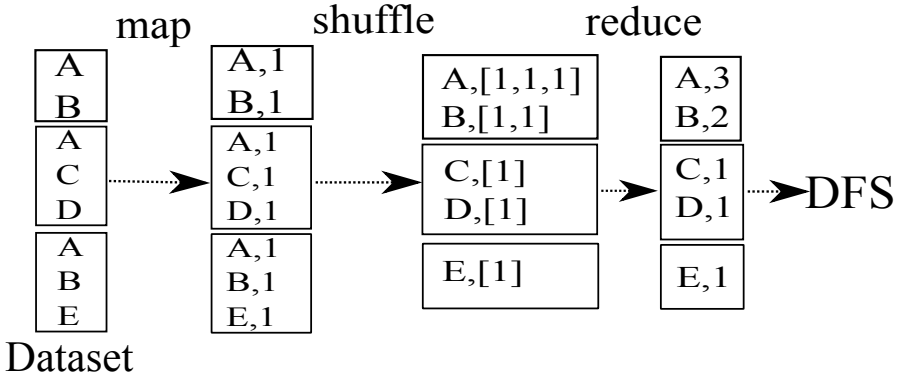

Figura 1.1. Fluxo de execução simplificado do modelo *MapReduce*.

Nem todas as técnicas tradicionais de agrupamento de dados se aplicam ao modelo *MapReduce* ou ao paradigma de programação paralela. Isso resulta em um pequeno número de algoritmos disponíveis em plataformas distribuídas [Hamstra et al., 2015]. Outro fator que influência na adaptação ou criação de novos algoritmos para este contexto é a complexidade computacional, que pode tornar o algoritmo impraticável, visto que no agrupamento a complexidade geralmente se baseia no número de objetos do conjunto de dados.

O k-médias é um dos mais conhecidos algoritmos de agrupamento, eleito um dos dez algoritmos mais influentes na mineração de dados [Wu & Kumar, 2009]. Isso

[1]Antes da etapa *reduce* pode ser implementada uma etapa com funções do tipo *combiner*, responsáveis por combinar os dados localmente antes de enviá-los pela rede.

porque é um algoritmo simples, escalável e de fácil adaptação a diferentes domínios de aplicação.

A complexidade do algoritmo é $O(t.n.k)$, onde n é a quantidade de objetos do conjunto de dados, t o número de iterações e k o número de grupos. Em muitas aplicações, como a abordada neste trabalho, o número de objetos é muito maior que os outros valores e a complexidade final pode ser dada como $O(n)$. O k-médias possui duas fases bem separadas, facilmente adaptáveis ao modelo *MapReduce*, e por este motivo tem sido usado neste modelo em diversas plataformas escaláveis, como por exemplo no Apache Hadoop [White, 2012] e Apache Spark [Hamstra et al., 2015].

Embora k-médias apresente essas vantagens descritas acima, ele também possui limitações. O algoritmo é sensível a escolha dos protótipos iniciais [Wu & Kumar, 2009]. Podem ser usadas técnicas para contornarem este problema, como por exemplo, escolher melhores protótipos iniciais [Bahmani et al., 2012]. Além do problema de inicialização, assim como a maioria dos algoritmos de agrupamentos, k-médias possui como um dos parâmetros de entrada o número de grupos (k), que pode ser desconhecido caso não se tenha informações prévias sobre o conjunto de dados, o que geralmente ocorre para aplicações práticas [Naldi et al., 2011].

O foco desta pesquisa foi abordar técnicas que atenuam o problema de encontrar um bom valor para o número de grupos k. Para isso, foram estudadas meta-heurísticas evolutivas que apresentaram bons resultados no passado [Alves et al., 2006; Naldi et al., 2011; Naldi & Campello, 2014]. Foram desenvolvidas duas variantes do algoritmo *Fast Evolutionary Algorithm for Clustering* (F-EAC) [Alves et al., 2006], ambas adaptadas para o modelo *MapReduce*. O objetivo principal foi tornar os algoritmos escaláveis, possibilitando-os executar sobre grandes quantidades de dados em redes com grandes quantidades de máquinas. Em adição, também foi estudado um algoritmo que também resolve o problema de encontrar o número de k. Este algoritmo se chama *G-means* [Hamerly & Elkan, 2003] e teve uma versão adaptada para *MapReduce* pelos autores Debatty et al. [2014]. Foram realizados experimentos para cada um dos algoritmos trabalhados com a finalidade de testá-los, tanto do ponto de vista de tempo computacional quanto da qualidade das soluções finais.

1.2 Objetivos

Nesta seção são apresentados os objetivos geral e específicos que foram desenvolvidos ao longo desta pesquisa.

1.2.1 Objetivo Geral

O objetivo geral desta pesquisa foi adaptar e desenvolver algoritmos de agrupamentos para o modelo escalável *MapReduce*, buscando a aplicação do agrupamento a grandes quantidades de dados sem a necessidade de especificação do número de grupos k.

1.2.2 Objetivos Específicos

Nesta pesquisa foram considerados os seguintes objetivos específicos:

- Utilizar o modelo MapReduce para garantia de escalabilidade dos algoritmos.
- Utilizar o Apache Spark como plataforma por se mostrar superior ao Hadoop.
- Encontrar os pontos fortes e fracos de cada um dos algoritmos estudados e desenvolvidos.

1.3 Organização da dissertação

Esta dissertação foi elaborada no formato de uma coletânea de artigos produzidos durante a pesquisa, seguindo o formato recomendado pela Comissão do Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Viçosa. Ao todo foram escritos dois artigos. Sendo assim, a dissertação está organizada como se segue:

O Capítulo 1 apresenta a introdução do conteúdo abordado e descreve os objetivos desta pesquisa.

O Capítulo 2 contém os dois artigos produzidos. O primeiro artigo propôs um novo algoritmo evolutivo em *MapReduce*. O algoritmo é uma variante do F-EAC [Naldi et al., 2011], que adapta as principais operações do algoritmo para o modelo *MapReduce*. Além disso, a cada acesso a um objeto do conjunto de dados, o algoritmo utiliza esse acesso para mapear o objeto para todas as partições presentes em uma população de tamanho $|P|$, predefinida pelo usuário. O algoritmo também modifica os operadores de mutação para diminuir o acesso ao conjunto de dados.

Ao final, foram realizados experimentos para verificar se o algoritmo era capaz de encontrar resultados com qualidade igual ou superior ao resultado encontrado em um algoritmo que executava múltiplas partições de k-médias *MapReduce* em paralelo [Garcia & Naldi, 2014]. O segundo artigo propôs um novo algoritmo evolutivo, também adaptado para *MapReduce*, porém baseado na ideia de Combinação de Agrupamentos Distribuídos (CAD) [Naldi & Campello, 2014]. Neste algoritmo, ao invés de cada objeto ser processado por uma função *map*, um subconjunto de dados é processado. O agrupamento é realizado sobre cada um desses subconjuntos que gera uma solução parcial. Ao final, essas soluções são combinadas para a obtenção de uma solução final. Adicionalmente este novo algoritmo foi comparado ao *Scalable Fast Evolutionary Algorithm for Clustering* (SF-EAC), proposto no primeiro artigo e também ao *MapReduce G-means* [Debatty et al., 2014]. Os algoritmos foram comparados de duas maneiras: através da análise assintótica e de experimentos computacionais práticos. Ao final, foram realizados testes estatísticos para a avaliação final dos algoritmos.

O Capítulo 3 apresenta as conclusões gerais do trabalho de pesquisa e sugestões para trabalhos futuros.

A descrição dos artigos que compõem esta dissertação é apresentada a seguir:

- OLIVEIRA, G. V.; NALDI, M. C. . **Scalable Fast Evolutionary k-means Clustering**. *In: 2015 Brazilian Conference on Intelligent Systems (BRACIS)*, 2015, Natal. 2015 Brazilian Conference on Intelligent Systems, 2015. v. 1. p. 74-79.

- OLIVEIRA, G. V.; COUTINHO, F. P.; CAMPELLO R. J. G. B.; NALDI, M. C. . **Meta-heurísticas para aprimoramento de k-médias por meio de modelos distribuídos e escaláveis.** Convidado para submissão na revista *Neurocomputing*, em fase de tradução.

Capítulo 2

ARTIGOS

Este capítulo contém os resultados da pesquisa desenvolvida para esta dissertação. Ao longo da pesquisa, foram estudados algoritmos que pudessem ser adaptados para o modelo *MapReduce* cujo objetivo fosse encontrar automaticamente um bom valor para o número de grupos k. Neste modelo, foi observado que o k-médias tem sido o principal algoritmo adaptado. Isso porque possui uma complexidade linear, ideal para se trabalhar com grandes entradas de dados e também por se adaptar às restrições que o modelo *MapReduce* impõem. Entretanto, as implementações existentes não se focam no problema de encontrar um valor para k e sim no agrupamento propriamente dito. Uma das formas para se contornar este problema é o uso de k-médias por repetidas vezes com diferentes valores para k. Foi observado o uso de múltiplos agrupamentos em paralelo no modelo *MapReduce* [Garcia & Naldi, 2014]. Também foi observado o uso do algoritmo *G-means* [Debatty et al., 2014] para *MapReduce*, que inicia um agrupamento com um valor preestabelecido de k e aumenta esse valor, caso o algoritmo conclua que seja necessário. Adicionalmente, foi observado que o uso de algoritmos que usam meta-heurísticas evolutivas teve bons resultados para a solução deste tipo de problema no passado [Alves et al., 2006; Naldi et al., 2011; Naldi & Campello, 2014], porém estes algoritmos não haviam sido adaptados para este paradigma.

As Seções 2.1 e 2.2 apresentam os dois artigos resultantes desta pesquisa. O primeiro artigo "*Scalable Fast Evolutionary k-means*" apresentou um algoritmo evolutivo para *MapReduce* que executa k-médias para múltiplos agrupamentos paralelos para cada acesso do objeto de conjunto de dados. Além disso o algoritmo modifica os operadores de mutação para diminuir ainda mais esse acesso ao conjunto. O algoritmo foi comparado a múltiplas execuções de k-médias em *MapReduce* paralelos e conseguiu encontrar agrupamentos com solução igual ou superior em menor

tempo computacional. O segundo artigo "Meta-heurísticas para aprimoramento de k-médias por meio de modelos distribuídos e escaláveis" também apresenta um novo algoritmo em *MapReduce* para agrupamento. O algoritmo é baseado no conceito de Combinação de Agrupamentos Distribuídos. Este conceito agrupa subconjuntos de dados em soluções parciais e as utiliza para um agrupamento posterior. O algoritmo utilizou funções do modelo para mapear subconjuntos de dados ao invés de objetos individuais, com o objetivo de diminuir o trafego de informações da rede. Adicionalmente, o segundo artigo apresenta um algoritmo que foi adaptado para *MapReduce* que também objetiva encontrar automaticamente o número de grupos k. Ao final, os algoritmos são comparados de duas maneiras: através de sua complexidade assintótica e de experimentos em conjuntos de dados artificiais e reais. Os resultados são analisados através de testes estatísticos para determinar a diferença da performance dos algoritmos.

2.1 Scalable Fast Evolutionary *k*-means

Gilberto Viana de Oliveira, Murilo Coelho Naldi

In: 2015 Brazilian Conference on Intelligent Systems (BRACIS), 2015, Natal. 2015 Brazilian Conference on Intelligent Systems, 2015. v. 1. p. 74-79.

RESUMO

O aumento da quantidade de dados gerados exige uma maior escalabilidade pelos algoritmos de agrupamento. O paralelismo intrínseco do modelo *MapReduce* atribui gestão e confiabilidade para operações em grande escala distribuídas. No entanto, suas restrições dificultam a aplicação direta em vários algoritmos de agrupamento tradicionais. O k-médias é um dos poucos algoritmos que satisfazem as restrições *MapReduce*, porém necessita da especificação do número de grupos e também é sensível a inicialização. Este artigo propõe um algoritmo *MapReduce* capaz de evoluir grupos sem a necessidade de especificação dos parâmetros de k-médias. Através de operadores evolutivos, os grupos obtidos são utilizados para procurar por melhores soluções, permitindo que o algoritmo encontre soluções de qualidade rapidamente. O algoritmo é comparado com versões sistemáticas em *MapReduce* que são capazes de encontrar o número de grupos de k-médias. Experimentos computacionais e análises estatísticas dos resultados indicam que o algoritmo proposto é capaz de obter soluções com qualidade igual ou superior em tempo computacional menor se comparado à busca sistemática.

Palavras-chave: agrupamento evolutivo, k-médias, *MapReduce*.

ABSTRACT

The increasing amount of data requires greater scalability for clustering algorithms. The intrinsic parallelism of the MapReduce model confers management and reliability to large-scale distributed operations. However, its restrictions hinder the direct application of several traditional clustering algorithms. k-means is one of the few clustering algorithms that satisfy the MapReduce constraints, but it requires the prior specification of the number of clusters and is sensitive to their initialization. This paper proposes a MapReduce algorithm able to evolve clusters with no need to specify k-means' parameters. Through evolutive operators, obtained clusters are used to search for better solutions, allowing the algorithm to find quality solutions quickly. The algorithm is compared with state-of-the-art MapReduce versions of a systematic algorithm which is able to find the number

of k-means clusters and initializations. Computational experiments and statistical analyses of the results indicate that the proposed algorithm is able to obtain clusters with quality equal or superior to clusters of the compared algorithm, but faster.

Keywords: evolutionary clustering; k-means; MapReduce.

2.1.1 Introduction

The term big data has been used to convey several concepts such as massive amounts of data, social media analyses, data-management resources for the next generation, real-time data, etc. Given these large amounts of data, management frameworks such as Apache Hadoop [White, 2012] and Apache Spark [Hamstra et al., 2015] have been developed. The former is based on the powerful distributed and scalable programming model MapReduce [Dean & Ghemawat, 2004], which was shown to be restrictive by hindering comparisons between distributed objects in a dataset. In order to overcome these obstacles, Spark extends the MapReduce model with further functionalities, which include the use of main memory through an abstraction called Resilient Distributed Data (RDD). Frameworks like these directly support new tools and algorithms able to manage this massive amount of data and have been used by companies such as IBM, Microsoft, Dell, Yahoo, and Amazon in projects that enable processing in distributed commodity servers.

In essence, data science consists in extracting knowledge from data. One of the main tasks in this field is clustering, whose aim is to divide the dataset into a finite number of categories. The applicability of these algorithms comprehends areas ranging from image processing and market segmentation to document categorization and bioinformatics (see [Jain et al., 1999; de Vega & Cantú-Paz, 2010]), among others. Amidst the clustering algorithms, k-means stands out for being considered one of the ten most influent algorithms in data mining. Such influence is due to its simplicity, scalability, and easy adaptation to different domains [Wu & Kumar, 2009]. Nevertheless, k-means is sensitive to the initial choice of prototypes that will represent the clusters. It also requires the number of k clusters as an input, which makes it quite restrictive in practice when this value is unknown, mainly in real-world applications.

In order to extract knowledge from the increasing amount of data, clustering algorithms may be adapted to the MapReduce model. In face of the restrictions

imposed by MapReduce, k-means is one of the few clustering algorithms that may be adapted exactly. Nonetheless, due to the algorithm's limitations, quality solutions must be sought among the multiple runs of the algorithm with different numbers of clusters and initial prototypes. At the end, the resulting clusters must be assessed and selected. The need for multiple runs of the algorithm allows techniques that aim to increase the computational performance, such as parallelism, to be explored. For instance, in [Garcia & Naldi, 2014] the authors propose the multiple parallel and distributed execution of the k-means algorithm in MapReduce and its validation using a relative index [Vendramin et al., 2010]. The technique proposed by those authors was shown to be able to overcome, in terms of computational performance, multiple k-means runs in the Mahout library [Owen et al., 2011]. However, the proposed technique limits the algorithm's to an interval of values of k and does not use any information in the clustering process to guide the search.

The present study proposes the scalable, parallel, and distributed use of an evolutionary clustering algorithm based on k-means for scenarios with large amounts of data and unknown number of clusters. The proposed algorithm is able to perform and evolve several clustering solutions with a single access to the data for each k-means iteration and a single extra access to calculate a validation index. It uses the MapReduce programming model, which provides it with scalability, parallelism, and high distribution capacity. The algorithm is compared to one of the state-of-the-art in k-means MapReduce clustering algorithms which is able to automatically find the number of clusters. The results show that the algorithm proposed is able to find a solution with quality equal to or better than the compared algorithms while using less computational time. Besides, the algorithm is able to keep evolving partitions during search to seek better solutions.

The remaining of this paper is organized as follows: Section 2.1.2 presents a brief description of the field covered by this research. Section 2.1.3 details the proposed algorithm. Section 2.1.4 shows the results of the experiments performed. Finally, the conclusions are presented in Section 2.1.5.

2.1.2 Related Work

The MapReduce programming model [Dean & Ghemawat, 2004] enables the management, parallelization, and distribution of the processing of large amounts of data. The model abstracts the way the data are treated and allows calculations to be performed using a set of $< key/value >$ pairs. The processed set of pairs yields new intermediate $< key/value >$ pairs using *map* functions. Finally, the values

with the same intermediate *key* are merged by the *reduce* functions. All pairs that execute the *map* task are run in parallel. Despite the advantages inherent to its use, MapReduce is quite restrictive in practice. The calculations applied to each *map* function occur independently and no information can be exchanged during the execution of each of these functions. Due to this type of restriction, simple clustering algorithms are usually chosen to work with this model. Another important factor is the complexity of the algorithm chosen, which may render the processing impracticable for large amounts of data.

In face of these restrictions, several techniques developed in MapReduce have been based on some sort of implementation of the k-means algorithm [Zhao et al., 2009; Liu & Cheng, 2012; Xu et al., 2012; Bahmani et al., 2012; Yang et al., 2013]. The choice of this algorithm occurs because it is one of the most influent algorithms in data mining [Wu & Kumar, 2009], besides the possibility of easily adapting its steps to the model's functions. Its popularity is responsible for its implementation in important projects such as Mahout for Hadoop [Owen et al., 2011] and the machine-learning algorithm library MLlib for Spark [Hamstra et al., 2015]. However, these implementations suffer from the known limitations of the algorithm: its sensitivity in choosing initial prototypes and the need to previously define the number of k clusters. In practical scenarios, where this number is unknown, several runs may be required with different k and different initial prototypes for each k value. By the end of all runs, the best clustering must be selected as the final solution according to some validation criterion [Jain et al., 1999].

Recently, Garcia & Naldi [2014] has proposed an algorithm based on multiple k-means runs in MapReduce called MRMk-means. The algorithm's differential is its ability to overcome the limitations of k-means by being able to automatically select the initial prototypes and the best number of clusters in the dataset within a k-value range provided by the user. The algorithm consists on executing in distributed and parallel manner the p number of partitions for each k value in the range informed while accessing the data a single time for each k-means iteration. Next, the algorithm applies the simplified silhouette validation criterion [Hruschka et al., 2004] using the same parallelization and distribution strategy employed to generate the clusters. The final partition is chosen from the assessment performed by the criterion. MRMk-means was able to obtain better performance compared to multiple runs of the k-means implementation in the Mahout library, which is used worldwide and supported by the Apache foundation.

In Naldi et al. [2011], the authors show that evolutionary clustering algorithms tend to find clusters with quality equal to or better than those found through sys-

temic (repetitive) methods of k-means runs. This occurs because evolutionary algorithms use information on previously obtained clusters to enhance and guide the search for better solutions. This information is used in evolutionary operators that seek the most appropriate number of clusters for the data, even if this number is not within a user-defined range. An exact distributed version of the algorithm, DF-EAC, is proposed in Naldi & Campello [2014]. However, the DF-EAC still lacks the scalability inherent to the applications that use MapReduce and, therefore, lacks the safety and resilience of the frameworks that use this programming model.

2.1.3 Scalable Fast Evolutionary Algorithm for Clustering

A partition of a data set $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, composed of an a-dimensional feature or attribute vectors $\mathbf{x}_j$, is a collection $C = \{C_1, C_2, ..., C_k\}$ of k non-overlapping data subsets C_i (clusters) such that $C_1 \cup C_2 \cup ... \cup C_k = X$, $C_i \neq \emptyset$ and $C_i \cap C_l = \emptyset$ for $i \neq l$. Essentially, the Scalable Fast Evolutionary Algorithm for Clustering (SF-EAC) is the MapReduce version of the Fast Evolutionary Algorithm for Clustering (F-EAC) [Alves et al., 2006; Naldi et al., 2011]. It was designed to efficiently evolve partitions obtained from the k-means algorithm. In SF-EAC, partitions are represented as *genotypes*, evolved through mutation and selection operators. These operators use probabilistic rules to process partitions sampled from the search space. Roughly speaking, better partitions have higher probabilities of being sampled. In other words, the evolutionary search is biased towards more promising clustering solutions. To determine the relative quality of each partition, here called *fitness*, a relative validation index [Vendramin et al., 2010] is used to assess the appropriateness of each partition obtained. When used for this purpose, the relative index is considered a *fitness function* $f(\cdot)$.

In the F-EAC, each partition is represented as an individual of a population. The encoding scheme used is based on an integer vector $\mathbf{g}$ that has n positions, each of which corresponding to an object of the dataset. A cluster label is assigned to each vector position. Thus, $\mathbf{g}(i)$ stores the label (from the set $\{1, 2, 3, ..., k\}$) of the cluster that contains the i-th object. The algorithm requires the number of genotypes $|P|$ to be processed during each iteration, which is known as *population size*. In this study, the genotypes are initialized by randomly drawing both the number of clusters (k) and the prototypes for each partition. This procedure somehow favors diversity in the initial population by creating initial partitions that represent different numbers of clusters. Thus, the algorithm requires a maximum and minimum *initial* number of clusters, k_{max} and k_{min} respectively. It is important to note that algorithm is robust

to the choice of values for these parameters, since its evolutionary search operators are able to properly increase or decrease the number of clusters of partitions above or under the initial limits during execution [Alves et al., 2006; Naldi et al., 2011].

For each SF-EAC iteration (also known as *generation*), every partition is fine-tuned with the k-means algorithm, as detailed in Section 2.1.3.1. Thus, SF-EAC requires at least one stopping criterion for k-means. In this study, a maximum number of iterations t is used. Another suitable criterion is convergence, which is attained when no significant difference is observed between the same clusters in two consecutive iterations. Last but not least, SF-EAC requires stopping criteria $\mathsf{s_C}$ for itself. In practical applications, imposing a maximum number of generations and/or a minimum threshold for population diversity are suitable choices. Once stopped, the algorithm returns the best evaluated partition $\mathbf{g}_S$ and its number of clusters k^*.

A description of the SF-EAC algorithm is presented in Algorithm 1. The difference between the original F-EAC and SF-EAC is how the steps 4, 5 and 14 in Algorithm 1 are implemented. These steps are described in Sections 2.1.3.1, 2.1.3.2, and 2.1.3.3 respectively.

2.1.3.1 Local Search

At each algorithm generation, the algorithm applies a local search for better partitions. One of the main differences between SF-EAC and the previous versions of F-EAC is the parallel simultaneous application of k-means to all individuals. This is performed with a single MapReduce iteration per k-means iteration, which is split into two main functions: *map* (Algorithm 2) and *reduce* (Algorithm 3).

Each *map* function consists in calculating the dissimilarities between an object (vector $\mathbf{x}_i$) in the dataset and the k centroids of an individual stored in a matrix $\mathbf{C}$. Hence, $n \times |P|$ functions are executed in parallel, one for each combination between one vector of the dataset and one individual. The output of each *map* function is an $<id/object+>$ pair, where id identifies the cluster to which the vector $\mathbf{x}_i$ belongs and $object+$ is a data structure that contains the vector $\mathbf{x}_i$ and the total n_S vectors stored in the structure [1].

The pairs $<id/object+>$ with the same id are merged and sent to a *reduce* task, which executes the *reduce* function described in Algorithm 3. This function calculates the centroid identified by id, returning an $<id/centroid+>$ pair, where id is the cluster identification and $centroid+$ is a data structure that contains the

[1] After the *map* function, each $object+$ stores a single object. However, this value changes during the execution of SF-EAC.

Algoritmo 1: *Scalable Fast Evolutionary Algorithm for Clustering (SF-EAC).*

Require: k_{min}, k_{max}, t, $\mathsf{S_C}$ and $|P|$;
1: $i \leftarrow 1$;
2: initialize a population P_i with $|P|$ genotypes encoding $k \in \{k_{min}, ..., k_{max}\}$ random clusters each;
3: **repeat**
4: apply multiple MapReduce k-means to all genotypes in P_i;
5: evaluate every genotype in P_i in parallel according to the fitness function;
6: compute the best fitness value v_{VC}, store its corresponding genotype $\mathbf{g}_S$;
7: $k^* \leftarrow$ the number of clusters of the partition encoded into $\mathbf{g}_S$;
8: **if** $\mathsf{S_C}$ is not met **then**
9: apply elitist strategy;
10: select genotypes from P_i;
11: calculate which mutation operator will be applied to each genotype;
12: **for all** selected genotypes **do**
13: select the clusters to be mutated;
14: apply mutation operators to the selected clusters to create new genotypes;
15: **end for**;
16: copy the new genotypes to the next population P_{i+1};
17: $i \leftarrow i + 1$;
18: **end if**;
19: **until** $\mathsf{S_C}$ is met;
20: **return** k^* and the corresponding genotype $\mathbf{g}_S$;

Algoritmo 2: SF-EAC k-means *map* function.

Require: $\mathbf{x}_i$, k, and $\mathbf{C}$;
1: **for all** k centroids **do**
2: calculate the distance between the $\mathbf{x}_i$ and the current centroid (stored at $\mathbf{C}$);
3: **end for**
4: store in *id* the position of $\mathbf{C}$ where the nearest centroid is;
5: instantiate an *object+* data structure with $\mathbf{x}_i$;
6: **return** the <*id/object+*> pair;

cluster centroid and the number of objects in the *id*-th cluster. All *reduce* tasks run in parallel, which characterizes the *reduce* job.

After the *reduce* job, the <*id/centroid+*> pairs are stored in the distributed file system (DFS). If the stopping criterion is not satisfied, a new iteration of the algorithm is run and the <*id/centroid+*> pairs are used to update the $\mathbf{C}$ matrix, which is required for the next k-means *map* job. Otherwise, the algorithm stops.

Algoritmo 3: SF-EAC k-means *reduce* function.

Require: $<id/object+>$ merged by id pairs;
1: initialize variable *count* and vector **sum** with zeros;
2: **for all** $<id/object+>$ pairs **do**
3: $\quad$ **sum** $\leftarrow$ **sum** $+ \mathbf{x}_i$;
4: $\quad$ $count \leftarrow count + n_s$;
5: **end for**
6: calculate and store the centroid by **sum**/*count*;
7: instantiate a *centroid+* data structure with the centroid;
8: **return** the $<id/centroid+>$ pair;

2.1.3.2 Fitness Calculation

After the local search, the fitness of each individual must be assessed. In SF-EAC, the fitness value is the result of the simplified silhouette value index [Hruschka et al., 2004], which obtained the best result in a recent comparative study [Vendramin et al., 2010] and fulfills the restrictions imposed by the MapReduce model. To calculate the silhouette, once again the MapReduce model is used. Each *map* function consists in calculating the dissimilarities between one vector $\mathbf{x}_i$ of the dataset and the k centroids of an individual stored in a matrix $\mathbf{C}$ at the end of the local search. The output of each *map* function is an $<id/silhouette+>$ pair, where id identifies the cluster to which the vector $\mathbf{x}_i$ belongs and *silhouette+* is a data structure that contains the vector's silhouette value and the total n_{SS} silhouette values stored in the structure. All *map* tasks run in parallel, which characterizes the map job.

Algoritmo 4: SF-EAC simplified silhouette *map* function.

Require: $\mathbf{x}_i$, k, $\mathbf{C}$;
1: **for all** k centroids **do**
2: $\quad$ calculate the distance between $\mathbf{x}_i$ and the current centroid (stored at $\mathbf{C}$);
3: $\quad$ store in a the shortest distance;
4: $\quad$ store in id the position in $\mathbf{C}$ of the nearest centroid;
5: $\quad$ store in b the second shortest distance;
6: **end for**
7: calculate $silhouette \leftarrow (b - a)/max\{a, b\}$;
8: instantiate a *silhouette+* data structure with the *silhouette* value;
9: **return** all $<id/silhouette+>$ pairs;

All $<id/silhouette+>$ pairs with the same id are merged and sent to a *reduce* task, which executes the *reduce* function described in Algorithm 5. This function calculates the mean simplified silhouette of each cluster and returns a $<id/silhouette+>$ pair, where id is the cluster identification and *silhouette+* is a

data structure that contains the cluster silhouette and the number of objects in the *id*-th cluster. Based on the information in *silhouette+*, the mean simplified silhouette can be calculated for the whole partition, which is the mean silhouette of the clusters pondered by the number of objects they contain. All *reduce* tasks run in parallel, which characterizes the *reduce* job.

Algoritmo 5: SF-EAC simplified silhouette *reduce* function.

Require: <*id/silhouette+*> merged pairs;
1: initialize variables *sum* and *count* with zeros;
2: **for all** <*id/silhouette+*> pairs **do**
3: $sum \leftarrow sum + silhouette$;
4: $count \leftarrow count + n_{ss}$;
5: **end for**
6: **if** $count = 1$ **then**
7: $silhouette_{id} \leftarrow 0$;
8: **else**
9: $silhouette_{id} \leftarrow sum/count$;
10: **end if**
11: instantiate a *silhouette+* data structure with $silhouette_{id}$ as *silhouette* and *count* as n_{ss};
12: **return** the <*id/silhouette+*> pair;

2.1.3.3 Mutation Operators

SF-EAC has two mutation operators. The first mutation operator (MO_1) eliminates one or more selected clusters, assigning each of their objects to the cluster in which the centroid is the least dissimilar to this object. MO_1 can only be applied to genotypes that encode more than two clusters and at least two clusters must remain after the application. The second mutation operator (MO_2) is only applied to clusters formed by at least two objects. It splits one or more selected clusters into two new clusters each.

In order to reduce the number of MapReduce tasks and, consequently, data access, the method of implementing SF-EAC mutation operators somewhat differs from the original methods [Naldi et al., 2011]. MO_1 only eliminates the selected clusters so that the objects that belonged to these clusters are reallocated in other clusters during the next local search step (Section 2.1.3.1). In turn, MO_2 empties the selected clusters and duplicates their centroids, thus creating two new identical clusters. In the next local search step, if there is a tie in an object's shortest dissimilarity for two or more clusters, then this object is randomly drawn among

these clusters. Although SF-EAC mutation operators depend on the local search step, no direct access to the data is required.

Both SF-EAC operators adopt a (probabilistic) informed search strategy for mutation purposes. The informed search is guided by the fitness of the clusters and probabilistic selection may be applied to choose the clusters to be mutated, e.g., the roulette wheel strategy [Davis, 1991]. In this case, a linear normalization [Davis, 1991] is recommended before selection to avoid premature convergence and balance the evolutionary pressure.

Apart from the genotypes chosen by the elitist strategy [Mitchell, 1998], every genotype selected in Step 10 is modified by one of the mutation operators. In SF-EAC, the mutation operator to be applied is proportional to its performance during the evolutionary search. One simple way of accomplishing this is to consider the performance of the operators individually for each genotype. If the use of an operator generated a genotype with fitness higher than its predecessor, this operator will be chosen to mutate the generated genotype afterwards. Otherwise, the other operator will be chosen. If the genotype belongs to the initial population or was selected by elitism, both operators have the same chance of being chosen.

2.1.4 Experiments

In order to assess SF-EAC's performance, the algorithm was compared to MRMk-means [Garcia & Naldi, 2014], since it is also able to automatically determine the number of clusters and their initial prototypes for the k-means algorithm. To the best of the authors' knowledge, no other MapReduce algorithm has been proposed with such characteristics. To that end, we used the original version of MRMk-means, which was implemented for Hadoop. However, the Spark framework offers advantages in relation to Hadoop for iterative algorithms [Hamstra et al., 2015]. Thus, besides the original MRMk-means version, the Spark versions of MRMk-means and of SF-EAC are also compared.

For the experiments presented here, three data sets were randomly generated with overlapping mixtures of Gaussian distributions by the MixSim R Package [Melnykov et al., 2012]. All data sets have objects with ten attributes, divided into three mixtures of Gaussian distributions. The main differences among them are their sizes, with 10^7, 2×10^7, and 5×10^7 objects respectively. Additionally, a dataset with 5×10^5 articles from Medline database (Pubmed) [Roberts, 2001] was created. Although they may not be considered "big data" in every scenario, they are big enough to stress our small eleven node Apache Hadoop 2.4.0 and Apache Spark

1.1.0 cluster. Each node runs in a computer with one I5 3.20 GHz processor and 12 GB of RAM. All algorithms ran in parallel managed by the distributed system.

For all algorithms, the k-means stopping criterion adopted is $t = 5$ iterations. For MRMk-means, we ran the clustering of 29 partitions with k values between $k_{min} = 2$ and $k_{max} = 30$. This range was chosen for being close to the limit allowed by the cluster used. The algorithm is run ten times with randomly selected initial prototypes for each k value for a total of 290 partitions. Here, this set of runs will be called procedure. In order to increase the experiment's reliability, given the algorithm's probabilistic nature, we repeated the indicated procedure ten times. Two MRMk-means versions were compared: one in the Hadoop framework (original) and the other in the Spark framework. For each procedure, the best partition according to the simplified silhouette criterion was selected as the final solution and the execution time was stored. In an attempt to achieve a fair comparison between MRMk-means and SF-EAC, we determined that $|P| = 29$, $k_{min} = 2$, $k_{max} = 30$ and the algorithm was executed for ten generations (s_C), which totals 290 partitions and is equivalent to one MRMk-means procedure. Since SF-EAC is probabilistic, it was also executed ten times. The mean and standard variation (in parenthesis) values over the computational time (in seconds) for both versions of MRMk-means (Hadoop and Spark) and SF-EAC are presented in Table 2.1. The SF-EAC execution time was measured at two moments: when the algorithm obtains a partition with quality equal to or better than the *best quality* among all MRMk-means solutions (presented in Table 2.1 as "SF-EAC (best)") and at the total execution time (presented as "SF-EAC (total)").

Tabela 2.1. Mean and standard deviation of the computational time (in seconds) of the compared algorithms.

Dataset	MRMk(Hadoop)	MRMk(Spark)	SF-EAC (best)	SF-EAC (total)
10^7	54920.6 (2118.4)	4720.4 (325.3)	750.5 (196.9)	2207.5 (372.3)
2×10^7	77884.0 (1292.8)	5194.5 (271.4)	887.1 (322.8)	2965.7 (207.0)
5×10^7	100611.3(1659.7)	11089.6(363.6)	1573.3 (482.8)	7738.0 (584.0)
PubMed	17103.9 (455.1)	713.7 (20.9)	205.3 (54.5)	325.6 (15.6)

In order to assess the significance of the experimental results, the analysis of variance (ANOVA) test [Walpole et al., 2006] was adopted over the executions of the compared algorithms. When the null hypothesis was rejected, indicating that there is statistical evidence to support that the compared results are different, the Bonferroni procedure [Hochberg & Tamhane., 1987] was applied to the critical values to compensate for the multiple comparisons and maintain the current level of

statistical confidence at 95%. The test results showed that the differences among all compared algorithms are significant, except between SF-EAC's "best" and "total" times for PubMed dataset.

2.1.5 Conclusions and future work

The values presented in Table 2.1 enable the conclusion that SF-EAC obtained results with quality equal to or better than MRMk-means while using less computational time for all datasets. Such differences are significant at 95% confidence according to the statistical tests employed. Since SF-EAC's partitions evolves, the computational time required to process them is reduced with the generations. This is advantageous when the minimum quality desired is known (expressed by a validation value) or little time is available for clustering. Moreover, the SF-EAC's evolutionary search is not restricted to a pre-defined range of values for the number of clusters. The MapReduce model guarantees the algorithm's scalability. As a secondary outcome, it can be seen that the Spark framework obtains better results than Hadoop for iterative procedures.

Future works must investigate other initialization types for k-means that allow the algorithm to seek better solutions with a smaller number of generations. The adaptation of clustering algorithms for local use with the MapReduce model and their combination to form a global data partition will also be investigated. We expect that this approximate approach may pose an advantage in terms of computational time.

Acknowledgment

The authors acknowledge the Brazilian agencies CNPq, CAPES and FAPEMIG for the financial support.

2.2 Meta-heurísticas para aprimoramento de *k*-médias por meio de modelos distribuídos e escaláveis

Gilberto Viana de Oliveira, Felipe Provezano Coutinho, Ricardo José Gabrielli Barreto Campello, Murilo Coelho Naldi

Convidado para submissão na revista *Neurocomputing*.

RESUMO

O aumento dos conjuntos de dados gera uma necessidade cada vez maior de escalabilidade por parte dos algoritmos de agrupamento. O modelo de programação *MapReduce* permite segurança, gerenciamento e escalabilidade. Entretanto, ele possui algumas restrições que podem dificultar ou até impossibilitar a adaptação de diversos algoritmos de agrupamentos tradicionais. O k-médias é um dos dez algoritmos mais influentes em Mineração de Dados e pode ser facilmente adaptado para o modelo *MapReduce*. Contudo, o algoritmo possui algumas limitações, como a necessidade de definição prévia de um número de grupos (k) e sensibilidade a inicialização dos mesmos, o que pode ser um problema em aplicações reais em que tais informações são desconhecidas. Neste trabalho, são propostas e apresentadas duas meta-heurísticas evolutivas e escaláveis em *MapReduce* que buscam automaticamente o agrupamento com melhor número de grupos para grandes conjuntos de dados. A primeira consiste em um algoritmo capaz de aprimorar agrupamentos obtidos por meio do algoritmo k-médias e operadores evolutivos. A segunda consiste em aplicar o k-médias para obter agrupamentos em subdivisões do conjunto de dados e, posteriormente, combinar os resultados obtidos. Todas as técnicas abordadas são comparadas assintótica e experimentalmente com um algoritmo estado-da-arte desenvolvido em *MapReduce*. Os resultados foram analisados por meio de testes estatísticos e indicam que a primeira obteve os resultados de melhor qualidade, enquanto que a segunda obteve os melhores tempos computacionais.

Palavras-chave: k-médias, meta-heurísticas, algoritmos evolutivos, otimização, algoritmos distribuídos, *MapReduce*, escalabilidade.

ABSTRACT

The increase of datasets generates the growing necessity of scalability by the clustering algorithms. The MapReduce programming model provides safety,

management, and scalability. However, it has some restrictions that can hamper or even make the adaptation of several traditional clustering algorithms impossible. k-means is one of the ten most influential Data Mining algorithms and can be easily adapted to the MapReduce model. Nevertheless, the algorithm has some limitations, such as the requirement of the number of clusters and the sensibility to clusters' initialization, which can be a problem in real applications where such information is unknown. In this paper, we propose and present two evolutionary metaheuristics in MapReduce that are scalable and automatically searche for the best number of clusters. The first one consists of an algorithm that is able to improve clusters obtained from k-means and evolutionary operators. The second one consists of applying k-means to obtain clusters from subdivisions of the dataset and, posteriorly, combining the results. All covered techniques are asymptotically and experimentally compared with a state-of-the-art algorithm developed in MapReduce. The results were analyzed using statistical hypothesis tests and indicate that the first algorithm obtained better quality results, while the second one obtained better computational time.

Keywords: k-means, metaheuristics, evolutionary algorithms, optimization, distributed algorithms, MapReduce, scalability.

2.2.1 Introdução

Com o aumento significativo da quantidade de dados gerados a cada dia, surgem desafios relacionados à compreensão e exploração dos dados. Trabalhar com esses grandes conjuntos de dados objetiva a geração de valor ou extração de conhecimento útil destes. Para alcançar tal objetivo, é preciso utilizar modelos de programação que possibilitem a análise de grandes quantidades de dados em tempo hábil, garantindo segurança contra falhas e escalabilidade à medida que a demanda pela análise e o tamanho do conjunto de dados aumenta.

Um poderoso modelo de programação, batizado de *MapReduce*, foi apresentado em Dean & Ghemawat [2004]. Este modelo é uma abstração que permite uma programação distribuída e escalável de maneira simplificada. Porém, restringe os algoritmos a dividirem suas tarefas entre duas funções principais, chamadas *map* e *reduce*, e impõe um fluxo de execução para as mesmas. Algoritmos que seguem este modelo têm sido usados em empresas como IBM, Microsoft, Dell, Yahoo! Amazon,

dentre outras [2]. Algumas plataformas foram criadas para atenderem a esse modelo e facilitarem sua implantação. Uma delas é o *Apache Hadoop* [White, 2012], que se difundiu rapidamente desde sua criação. Outra plataforma é o *Apache Spark* Hamstra et al. [2015], que além de usar o *MapReduce* como base apresentou algumas vantagens, como por exemplo o uso de uma abstração chamada Dados Distribuídos Resilientes (DDR ou do inglês, RDD). Essa abstração permite que programas escaláveis realizem operações na memória principal com grande tolerância a falhas, sendo assim indicada para o uso em algoritmos iterativos [Hamstra et al., 2015], como é o caso de diversos algoritmos de Mineração de Dados.

Existem bibliotecas criadas com algoritmos para a solução de problemas em larga escala. Dentre elas, o projeto *Apache* Mahout [Owen et al., 2011] e a *Machine Learning Library* (MLlib) [Hamstra et al., 2015] possuem um conjunto de algoritmos de aprendizado de máquina adaptados para *MapReduce*. Existem poucos algoritmos para agrupamento de dados nessas bibliotecas. Isso acontece porque a maioria dos algoritmos tradicionais não foram originalmente projetados para trabalharem de forma distribuída e paralela e sua adaptação muitas vezes não é possível devido às restrições impostas pelo modelo ou pelo contexto escalável.

A Ciência dos Dados é um campo multidisciplinar, que envolve conhecimentos de estatística, matemática, ciência da computação, aprendizado de máquina, visualização de dados, dentre outros. É um campo que necessita de pessoas com conhecimentos práticos de ferramentas e materiais, juntamente com compreensão teórica do que é possível ser feito para uma boa utilização dos dados [Schutt & O'Neil, 2013]. Uma das formas tradicionais para se extrair conhecimento de dados consiste na aplicação de técnicas de agrupamento. O agrupamento de dados possui como objetivo dividir o conjunto de dados em um número finito de categorias de acordo com a similaridade presente entre os objetos do conjunto estudado [Jain et al., 1999]. O agrupamento é uma técnica não supervisionada, ou seja, o resultado (grupos) é obtido unicamente a partir do conjunto de dados. A aplicabilidade desses algoritmos engloba áreas como: processamento de imagens, segmentação de mercado, categorização de documentos e bioinformática [Jain et al., 1999; Xu & Wunsch, 2005], dentre outras.

Dentre os algoritmos de agrupamento, merece destaque o k-médias, considerado um dos dez algoritmos mais influentes em mineração de dados [Wu & Kumar, 2009]. Essa influência se deu por sua simplicidade, escalabilidade e sua fácil adaptação aos mais diversos domínios. Entretanto, k-médias possui algumas limitações.

[2]Disponível em: http://cwiki.apache.org/confluence/display/SPARK/Powered+By+Spark e http://wiki.apache.org/hadoop/PoweredBy

Como a maioria dos algoritmos de agrupamento, ele requer como parâmetro de entrada o número de grupos k. Encontrar esse valor é um grande desafio para a aplicação de agrupamento de dados, visto que sem esse valor os algoritmos se tornam bem restritivos na prática quando k é desconhecido, principalmente em aplicações do mundo real. Adicionalmente, o algoritmo também é sensível à escolha inicial dos protótipos que representarão os grupos.

O k-médias é o principal algoritmo adaptado para o modelo *MapReduce*. Uma das formas de superar suas limitações consiste em buscar soluções de qualidade dentre múltiplas execuções do algoritmo com diferentes números de grupos e protótipos iniciais e, ao final, avaliar e selecionar os agrupamentos resultantes. A necessidade de múltiplas execuções do algoritmo permite que técnicas que visem o aumento de desempenho computacional, como o paralelismo, sejam exploradas. Em Garcia & Naldi [2014], os autores propõem a execução múltipla paralela e distribuída do algoritmo k-médias em *MapReduce* (chamado MRMk-means) com validação feita por meio de índice de validação relativo [Vendramin et al., 2010].

Outro algoritmo que tem como objetivo encontrar o número de grupos k é o *G-means*, apresentado em Hamerly & Elkan [2003], que usa o teste estatístico de Anderson-Darling para determinar se é necessário ou não aumentar o valor de k. *G-means* teve uma versão adaptada para *MapReduce* [Debatty et al., 2014].

Em Oliveira & Naldi [2015], um algoritmo escalável que utiliza o modelo *MapReduce* foi proposto para evoluir agrupamentos resultantes de k-médias, de forma a encontrar o melhor número de grupos e sua inicialização. Nomeado como *Scalable Fast Evolutinary Algorithm for Clustering* (SF-EAC), o algoritmo trabalha utilizando informações de agrupamentos obtidos para guiar sua busca por melhores soluções. Esse algoritmo foi capaz de superar o MRMk-means [Garcia & Naldi, 2014], conseguindo encontrar uma solução de qualidade igual ou superior com menor tempo computacional.

Este trabalho consiste em uma extensão do trabalho apresentado em Oliveira & Naldi [2015]. No trabalho atual, o SF-EAC é apresentado em maiores detalhes e analisado assintoticamente. Também é proposto um novo algoritmo evolutivo que objetiva encontrar agrupamentos de qualidade próxima ao SF-EAC, porém em menor tempo computacional. O algoritmo é chamado de *Combination of Fast Evolutionary Algorithm for Clustering* (CF-EAC) e utiliza o *MapReduce* para dividir o conjunto de dados em subconjuntos, denominados blocos. Após a divisão, é aplicado o agrupamento em cada um dos blocos, formando soluções parciais. O agrupamento dessas soluções parciais é dado como a solução final para o agrupamento do conjunto de dados.

Adicionalmente neste trabalho, os algoritmos SF-EAC, CF-EAC e *G-means* são comparados de duas formas: assintoticamente, por meio da análise de custo computacional e empiricamente por meio de experimentos sobre conjuntos de dados artificiais e reais. A comparação entre quaisquer desses algoritmos não é feita em Oliveira & Naldi [2015] e, portanto, é inédita.

O restante deste artigo é organizado da seguinte maneira: na Seção 2.2.2 é apresentada uma breve descrição da área de pesquisa deste trabalho. Os algoritmos evolutivos SF-EAC e CF-EAC são apresentados nas Seções 2.2.3 e 2.2.4, respectivamente. O algoritmo *G-means* é apresentado na Seção 2.2.5. Os resultados dos experimentos comparativos são apresentados e analisados na Seção 2.2.6. Por fim, na Seção 2.2.8 são apresentadas as conclusões.

2.2.2 Clusterização escalável e o uso do *k*-médias

A dificuldade de adaptação dos algoritmos clássicos de agrupamentos em modelos paralelos e distribuídos foi apontada em Hamstra et al. [2015]. Isso acontece pois a maioria dos algoritmos não foram originalmente projetados para trabalhar neste paradigma. O principal modelo usado é o *MapReduce* [Dean & Ghemawat, 2004], que é projetado para escalar com milhares de processadores e gerencia para o programador os detalhes de distribuição, paralelismo e recuperação de falhas. O *MapReduce* abstrai a forma com que os dados são tratados e possibilita efetuar cálculos utilizando um conjunto de pares *chave/valor*. O conjunto de pares processado gera novos pares de *chave/valor* intermediários através de funções *map*. Por fim, os valores com a mesma *chave* intermediária são mesclados pela função *reduce*. Todas as funções *map* e *reduce* que manipulam os pares *chave/valor* são executados paralelamente. A Figura 2.1 mostra de forma simplificada o fluxo de execução das operações no modelo *MapReduce*. O objetivo da aplicação é obter a quantidade de cada uma das letras distribuídas no sistema. Cada *chave* (*letra*) é analisada por uma função *map*, que realiza a contagem inicial (valor 1). Em seguida, a saída das funções *map* são processadas pelas funções *reduce*[3], selecionadas por sua *chave* (*letra*). As funções *reduce* determinam a quantidade total de cada letra no sistema, resultando em um par *chave/valor* em que a *chave* é a *letra* e o *valor* é sua quantidade. Ao final de *reduce*, o resultado é escrito em um Sistema de Arquivos Distribuídos, do inglês, *Distributed File System* (DFS).

[3]Pode-se também executar funções do tipo *combiner* antes da próxima etapa, que é uma otimização para pré-combinar os valores localmente antes de transmití-los pela rede.

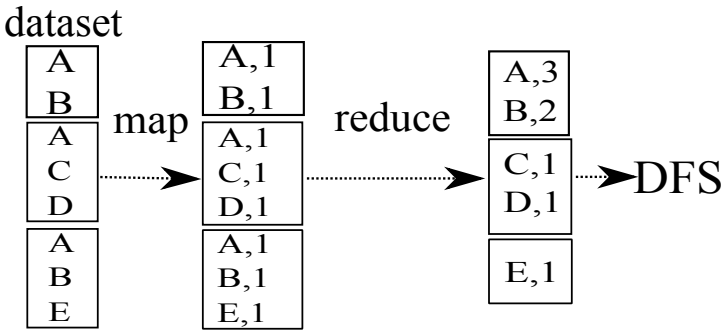

Figura 2.1. Exemplo do fluxo de execução do modelo *MapReduce* para uma contagem.

Apesar das vantagens inerentes a sua utilização, *MapReduce* é bem restritivo na prática. Os cálculos aplicados sobre cada função *map* acontecem de forma independente, não podendo existir uma troca de informação durante a execução de cada uma dessas funções entre si. Algoritmos de agrupamento de baixa complexidade computacional são geralmente escolhidos para serem trabalhados neste modelo, pois algoritmos de alta complexidade computacional tornam o processamento de grandes quantidades de dados inviável para a maioria dos cenários de aplicação. Ou seja, embora o modelo permita a implementação de algoritmos que comparem pares de objetos, para grandes quantidades de dados, esse processamento exige uma quantidade grande de recursos. O modelo *MapReduce*, apesar de permitir essas implementações, não foi projetado para otimizá-las, logo o tempo computacional para trabalhar com tais algoritmos seria demasiadamente caro.

Devido as restrições do modelo, várias técnicas desenvolvidas em *MapReduce* tem se apoiado em algum tipo de implementação do algoritmo k-médias [Zhao et al., 2009; Liu & Cheng, 2012; Xu et al., 2012; Bahmani et al., 2012; Yang et al., 2013]. A escolha desse algoritmo se dá pelo fato de ele ser um dos algoritmos mais influentes em Mineração de Dados [Wu & Kumar, 2009], além de ter seus passos facilmente adaptados para as funções do modelo *MapReduce*. Considerando que o k-médias pode ser descrito nos seguintes passos:

1. Inicializar conjunto de k protótipos;
2. Atribuir cada objeto do conjunto de dados ao protótipo mais próximo;
3. Calcular os centroides (média dos objetos de cada grupo), atribuí-los como novos protótipos;
4. Se o critério de parada não for satisfeito, voltar ao Passo 2;

o Passo 2 pode facilmente ser adaptado para uma função do tipo *map*, onde cada objeto que executar essa função resulta em um par *chave/valor*. Por sua vez, o Passo 3 pode ser adaptado para uma função do tipo *reduce*, onde a saída das funções *map* são mescladas para recalcular os novos valores dos centroides. Mais detalhes de uma implementação *MapReduce* para o k-médias são apresentados na Seção 2.2.3.1.

Uma das maneiras de otimizarmos o k-médias é contornando seu problema de sensibilidade durante a inicialização, de forma a buscar melhores protótipos iniciais, como foi apresentado em Bahmani et al. [2012]. Entretanto, o algoritmo ainda exige o número de grupos k como parâmetro de entrada. Em cenários práticos, k geralmente é um valor desconhecido, sendo então necessária a busca por este valor. Uma das formas de se fazer isso é executando várias vezes o algoritmo com diferentes valores para k e diferentes protótipos iniciais para cada valor de k. Ao final de todas as execuções, deve-se selecionar o melhor agrupamento como solução final, segundo algum critério de validação [Jain et al., 1999]. Para agilizar o processo de múltiplas execuções, Garcia & Naldi [2014] propuseram um algoritmo chamado MRMk-means. O algoritmo possui como diferencial a capacidade de executar múltiplos k-médias com diferentes valores para k em paralelo, e são predefinidos em um intervalo $[k_{min}, k_{max}]$ informados pelo usuário. Para cada execução, a solução é avaliada através do critério de validação da Silhueta Simplificada [Hruschka et al., 2004]. A partir da avaliação feita pelo critério, escolhe-se a melhor solução como final.

Outro algoritmo adaptado para *MapReduce* que objetiva encontrar o melhor valor de k aplicando o k-médias é o *G-means* [Debatty et al., 2014]. O algoritmo inicia com um número preestabelecido de k, geralmente pequeno, e incrementa o número de grupos iterativamente, caso o algoritmo julgue que tal incremento seja necessário. A cada iteração do algoritmo, os centroides que aparentemente não formam grupos com distribuição gaussiana são substituídos por novos centroides que serão testados na próxima iteração. Pode-se inicializar $k = 1$ ou outro valor maior, caso haja algum conhecimento prévio sobre o possível valor de k [Hamerly & Elkan, 2003]. É utilizado o teste estatístico de Anderson-Darling para verificar se os centroides possivelmente formam grupos pertencentes à uma distribuição gaussiana. Se o resultado for aceito, o centroide é marcado como encontrado. Senão, dois novos centroides são escolhidos para substituírem o centroide que falhou no teste [Hamerly & Elkan, 2003]. A versão *MapReduce* de *G-means* conseguiu se mostrar superior, quando comparada com múltiplas execuções de k-médias em *MapReduce*, tanto em tempo de execução, quanto em qualidade dos resultados [Debatty et al., 2014]. Maiores detalhes sobre o algoritmo são apresentados na Seção 2.2.5.

O *Fast Evolutionary Algorithm for Clustering* (F-EAC) é um algoritmo projetado para evoluir de forma eficiente agrupamentos resultantes de k-médias, com o objetivo de encontrar melhores soluções para um conjunto de dados [Naldi et al., 2011; Alves et al., 2006]. F-EAC é uma extensão do *Evolutionary Algorithm for Clustering* (EAC), proposto em Hruschka et al. [2006], e conseguiu se mostrar mais eficiente que seu predecessor [Alves et al., 2006]. Adicionalmente, F-EAC foi comparado também a múltiplas execuções de k-médias, conseguindo encontrar soluções com qualidade igual ou superior em menor tempo computacional [Naldi et al., 2011]. A razão pela qual isso ocorre é a utilização de informação sobre agrupamentos prévios feitos pelo algoritmo, que o torna capaz de guiar a busca por melhores soluções. Também é possível buscar por valores fora do intervalo preestabelecido pelo usuário. Uma variante do F-EAC foi apresentado em Oliveira & Naldi [2015], sendo esta uma versão *MapReduce* com ajustes feitos para se tornar escalável, chamada SF-EAC. No trabalho [Oliveira & Naldi, 2015], SF-EAC foi comparado com MRMk-means, conseguindo encontrar soluções de qualidade igual ou superior em menor tempo computacional, em cenários onde o número de grupos k é desconhecido.

2.2.3 *Scalable Fast Evolutionary Algorithm for Clustering* (SF-EAC)

O SF-EAC [Oliveira & Naldi, 2015] é uma variante do algoritmo F-EAC [Naldi et al., 2011; Alves et al., 2006], adaptada para o modelo *MapReduce* e, para isso, alguns ajustes foram necessários em relação ao seu predecessor. Contudo, com exceção destes ajustes de implementação (a serem apresentados nesta seção), o algoritmo segue exatamente os mesmo passos do F-EAC. Por esses motivos, o algoritmo F-EAC é apresentado no Algoritmo 6, de forma que as diferenças entre os algoritmos estão na forma como esses passos são implementados. Além disso, o F-EAC também é essencial para o algoritmo proposto na Seção 2.2.4.

Considere que um agrupamento deve particionar um conjunto de dados $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ de n objetos (com a-dimensões ou atributos) em diferentes grupos. Uma partição é dada por $\pi = \{C_1, C_2, ..., C_k\}$ de k subconjuntos C_i de dados não sobrepostos (grupos) tal que $C_1 \cup C_2 \cup ... \cup C_k = X$, $C_i \neq \emptyset$ e $C_i \cap C_l = \emptyset$ para $i \neq l$.

No F-EAC, diversas partições são criadas para a busca de uma solução para o agrupamento. Essas partições são evoluídas através de operadores de seleção e mutação. Os operadores usam regras probabilísticas para processar as partições amostradas no espaço de busca. Em outras palavras, partições com melhores soluções têm mais chances de serem escolhidas para a amostra, o que leva a busca a

Algoritmo 6: *Fast Evolutionary Algorithm for Clustering (F-EAC).*

1 Sejam k_{max} o número máximo de grupos iniciais, $\mathsf{S_{C1}}$ o critério de parada para k-médias, $\mathsf{S_{C2}}$ o critério de parada para F-EAC, v_{VC} o melhor valor de aptidão da geração atual e $\mathbf{g}_S$ o genótipo (partição) correspondente, k^* o número de grupos em $\mathbf{g}_S$, g o contador da geração, P_g a população atual, e $|P|$ o tamanho da população. Temos então o F-EAC:

Require: k_{max}, $\mathsf{S_{C1}}$, $\mathsf{S_{C2}}$ e $|P|$;

1: $g \leftarrow 1$;
2: inicializar a população P_g com $|P|$ genótipos (partições) com $k \in [2, k_{max}]$ grupos aleatórios para cada um;
3: **repeat**
4: aplique o algoritmo k-médias para cada genótipo em P_g até que $\mathsf{S_{C1}}$ seja satisfeito;
5: avalie cada genótipo em P_g de acordo com a função de avaliação;
6: calcule v_{VC}, armazene seu genótipo correspondente $\mathbf{g}_S$;
7: $k^* \leftarrow$ número de grupos da partição $\mathbf{g}_S$;
8: **if** $\mathsf{S_{C2}}$ não for satisfeito **then**
9: aplique elitismo;
10: selecione os genótipos de P_g;
11: calcule qual operador de mutação será aplicado em cada genótipo;
12: **for all** genótipos selecionados **do**
13: selecione os grupos que sofrerão mutação;
14: aplique os operadores de mutação sobre os grupos selecionados para criar os novos genótipos;
15: **end for**;
16: copie os novos genótipos para a próxima população P_{g+1};
17: $g \leftarrow g + 1$;
18: **end if**;
19: **until** $\mathsf{S_{C2}}$ seja satisfeito;
20: **return** k^* e o genótipo correspondente $\mathbf{g}_S$;

escolher soluções mais promissoras. Para avaliarmos a qualidade de uma partição, é usado um índice de validação relativo [Vendramin et al., 2010] sobre cada solução. Este índice relativo é chamado função de aptidão $f(\cdot)$.

O esquema de codificação usado pelo F-EAC é baseado em um vetor de inteiros $\mathbf{g}$ com n posições, também chamado genótipo, onde cada das posições desse vetor corresponde a um objeto do conjunto de dados. Sendo assim, cada posição do vetor recebe o rótulo de um grupo, logo, $\mathbf{g}(i)$ armazena o rótulo (do conjunto $\{1, 2, 3, ..., k\}$) do grupo relacionado ao i-ézimo objeto. Cada genótipo representa uma partição. O conjunto com todas as partições criadas pelo F-EAC representam sua população P e o tamanho dessa população é representado por $|P|$. Originalmente, os genótipos são

inicializados aleatoriamente entre o número de grupos (k) de sua respectiva partição, assim como seus protótipos iniciais. Esse processo é realizado para favorecer a diversidade da população inicial, formada por diferentes partições com diferentes números de grupos. Dessa forma, o algoritmo exige uma entrada para os valores mínimo (k_{min}) e máximo (k_{max}) de k. É importante notar que o algoritmo é capaz de modificar estes valores para fora deste intervalo durante a busca evolutiva [Naldi et al., 2011].

Para cada iteração do F-EAC (conhecido como geração), todas as partições são refinadas pelo algoritmo k-médias, como é detalhado na Seção 2.2.3.1. É necessário um critério de parada para o k-médias. Neste trabalho, um número máximo de iterações t é usado. Outro critério que pode ser utilizado é a convergência, que é obtida quando não se observa uma diferença significativa entre os centroides do mesmo grupo em duas iterações consecutivas. Por fim, F-EAC possui um critério de parada $\mathsf{s_C}$. Em aplicações práticas, este critério pode ser usado para impor um número máximo de gerações e/ou um limite mínimo para a diversidade da população. Ao atingir o critério de parada, o algoritmo retorna a partição com a melhor avaliação $\mathbf{g}_S$ e seu respectivo número de grupos k^*.

No SF-EAC, não é usado o esquema de codificação do genótipo do F-EAC original. Isso porque o algoritmo foi projetado para trabalhar com grandes quantidades de dados, que são gerenciados por plataformas que implementam o *MapReduce*. Logo, estas plataformas são as responsáveis por gerenciar e distribuir os dados, e restringindo a forma como as operações são realizadas sobre esses conjuntos. Por isso, não é uma tarefa trivial acessarmos uma posição específica do conjunto de dados ou mesmo controlar a ordem com a qual cada objeto do conjunto é mapeado por uma função do tipo *map*. Logo, o genótipo do SF-EAC não é composto por um vetor de inteiros, mas sim por um conjunto de pares *chave/valor*, onde a *chave* representa um número inteiro identificador do grupo, que é único. Já o *valor* representa um centroide referente ao grupo identificado.

Adicionalmente, as maiores diferenças entre o SF-EAC e o F-EAC estão nos Passos 4, 5 e 14 do Algoritmo 6, de forma que os detalhes de implementação desses passos descritos nas Seções 2.2.3.1, 2.2.3.2 e 2.2.3.3, respectivamente.

2.2.3.1 Busca Local

É aplicada uma busca local pelas melhores partições a cada geração do algoritmo. Essa busca é feita aplicando o algoritmo k-médias. No F-EAC, k-médias é aplicado para cada indivíduo da população de forma sequencial. Já no SF-EAC é feita

uma aplicação paralela desse algoritmo sobre todos os indivíduos, onde o k-médias é aplicado para todas as partições em uma única função *map*. Dessa forma, um único acesso é feito ao conjunto de dados por iteração de k-médias, independente do tamanho da população, que divide o processamento entre duas funções principais: *map* (Algoritmo 7) e *reduce* (Algoritmo 8).

Cada função *map* calcula as dissimilaridades entre um objeto (vetor $\mathbf{x}_i$) do conjunto de dados e os k centroides de todas as partições, armazenados em uma única matriz $\mathbf{C}$. A saída de cada função *map* é um conjunto de pares *id*/*object+*, onde *id* identifica o grupo ao qual o vetor $\mathbf{x}_i$ foi atribuído e *object+* é uma estrutura de dados que contém o objeto $\mathbf{x}_i$, um valor inteiro π para identificar partição do objeto e o total de objetos armazenados n_S na estrutura[4]. Todos os valores resultantes de *map* são estruturados de forma independente ao final do mapeamento, dessa forma, cada par *id*/*objeto+* pode ser processado individualmente na próxima etapa, independente de sua partição π.

Algoritmo 7: Função *map* k-médias SF-EAC.

Require: $\mathbf{x}_i$, P, e $\mathbf{C}$;
1: **for all** P partições **do**
2: **for all** centroides da i-ésima partição **do**
3: calcule a dissimilaridade entre o objeto $\mathbf{x}_i$ e seus centroides (armazenados em $\mathbf{C}$);
4: armazene em *id* a posição do centroide mais próximo;
5: **end for**
6: instancie uma estrutura de dados *objeto+* para o objeto $\mathbf{x}_i$;
7: crie e armazene um par *id*/*objeto+* para a partição atual;
8: **end for**
9: **return** o conjunto de pares *id*/*objeto+* do objeto $\mathbf{x}_i$;

Os pares *id*/*objeto+* que possuem a mesma *chave* (*id*) são combinadas em um resultado e enviadas a uma função *reduce*. Essa função calcula o centroide baseado no valor da *chave*, retornando um novo par *id*/*centroide+*, onde *id* é a identificação do grupo e *centroide+* é uma estrutura de dados que contém o centroide, sua partição π e o número de objetos n_S presentes no grupo. O Algoritmo 8 expressa os passos realizados pela função *reduce*.

Depois de executadas todas as funções *reduce*, os pares de valores *id*/*centroide+* são usados para atualizar os valores dos centroides de todas as partições, que estão armazenados no DFS. Caso o critério de parada não seja satisfeito,

[4]Após a função *map*, cada *objeto+* possui a variável $n_S = 1$, esse valor só é modificado caso valores de mesmo *id* sejam processados por uma função *reduce*, por exemplo, de forma que a estrutura contenha dados reduzidos de mais de um objeto do conjunto de dados.

Algoritmo 8: Função *reduce* k-médias SF-EAC.

Require: pares $id/objeto+$ combinados pelo id;
1: inicialize um contador *cont* e um vetor **soma** com todos os atributos zerados;
2: **for all** pares $id/objeto+$ **do**
3: $\quad$ **soma** $\leftarrow$ **soma** $+ \mathbf{x}_i$;
4: $\quad$ $cont \leftarrow cont + n_S$;
5: **end for**
6: calcule e armazene o centroide pela **soma**/*cont*;
7: instancie uma estrutura de dados $centroide+$;
8: **return** o par $id/centroide+$;

uma nova iteração de k-médias é iniciada, utilizando os valores atualizados, que foram armazenados na matriz $\mathbf{C}$. Caso contrário, o SF-EAC (assim como o F-EAC) segue para o Passo 5.

Para ilustrar o k-médias na versão MapReduce para múltiplas partições, um exemplo de aplicação é apresentado na Figura 2.2 Neste exemplo, um conjunto de dados (do inglês *dataset*) com nove objetos (rotulados $\mathbf{x}_1$ à $\mathbf{x}_9$) é dividido em dois nós escravos (*slaves*). Duas partições são inicializadas (identificadas com as cores branco e cinza), com dois (id_1 e id_2) e três grupos (id_3, id_4 and id_5), respectivamente. Os centroides de todos os grupos é armazenado na matriz $\mathbf{C}$ e enviada aos nós escravo, onde eles são comparados com todos os objetos pelas funções do tipo *map*. Esse procedimento resulta em pares $id/objeto+$, onde o número do id identifica a qual grupo o objeto pertence e o número seguido do *objeto* identifica quais objetos estão sumarizados na estrutura de dados $objeto+$ Os pares $id/objeto+$ com o mesmo id são localmente combinados e enviados para as funções *reduce*. Para cada id, a função *reduce* calcula o centroide do respectivo grupo. Por exemplo, o centroide do grupo $id2$ é a média dos objetos $\mathbf{x}_2, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_7$ and $\mathbf{x}_9$. Os novos centroides são enviados ao nó mestre, que pode iniciar uma nova iteração ou encerrar o algoritmo, caso o critério de parada seja satisfeito.

2.2.3.2 Função de Aptidão

Após a busca local, é necessário avaliar a aptidão de cada indivíduo. No F-EAC, o valor de aptidão é o resultado do índice de validação Silhueta Simplificada [Hruschka et al., 2004], índice que obteve o melhor resultado em um recente estudo comparativo [Vendramin et al., 2010] e que possui tempo computacional linear. Assim como na etapa anterior, este passo é realizado de maneira sequencial no F-EAC e paralelo no SF-EAC. Ou seja, cada função *map* executa sobre um objeto do conjunto de

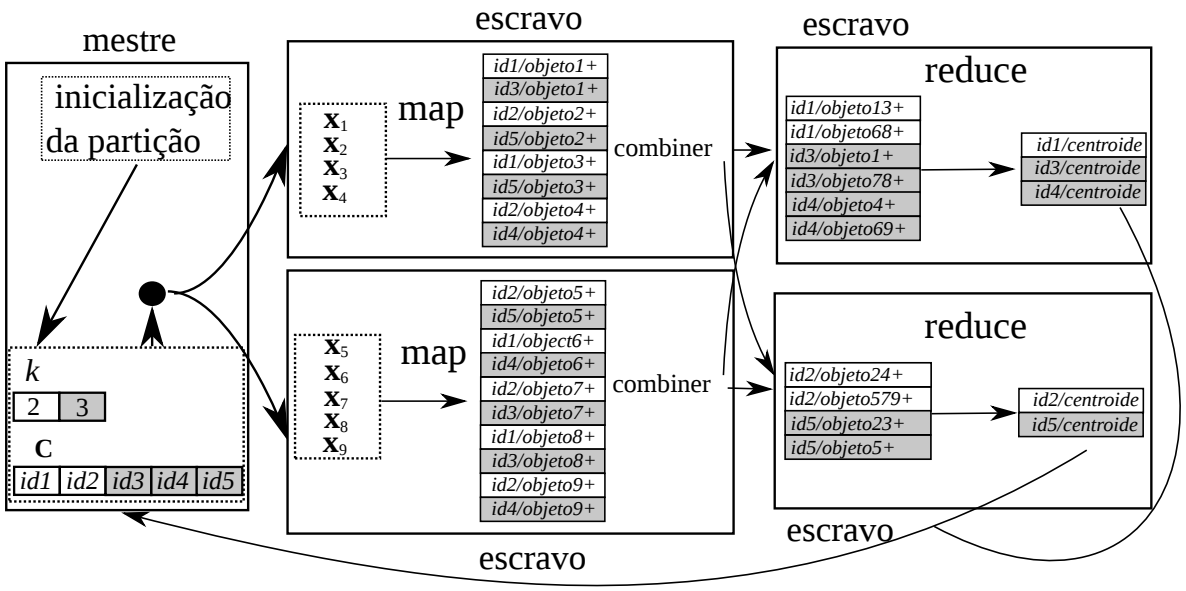

Figura 2.2. Exemplo de um fluxo de execução de uma iteração do k-médias MapReduce. As caixas indicam os nós da rede e as linhas indicam as transmissões de dados. Os centroides das duas partições (branca e cinza) são atualizados simultaneamente durante uma iteração do k-médias.

dados, calculando sua Silhueta Simplificada para cada uma das partições. O índice da Silhueta Simplificada pode ser facilmente dividida em duas etapas que atendam ao modelo *MapReduce*, sendo que:

- cada função *map* calcula os valores de Silhueta Simplificada do objeto $\mathbf{x}_i$ para todos os indivíduos.
- cada função *reduce* calcula a Silhueta Simplificada para cada grupo de todos os indivíduos.

Assim como o k-médias, o índice da Silhueta Simplificada realiza o cálculo para todas as partições com um único acesso ao conjunto de dados. Os detalhes das implementações em *MapReduce* são apresentadas nos Algoritmos 9 e 10.

Cada função *map* é realizada sobre um objeto $\mathbf{x}_i$ do conjunto de dados, calculando sua dissimilaridade para os centroides de cada partição. Após calcular a dissimilaridade, o algoritmo armazena: em *id* o identificador do grupo mais próximo; em a a distância para este grupo; em b a distância para o segundo grupo mais próximo[5]. Cada função *map* realiza essa operação para todas as partições, logo,

[5]Originalmente, o índice Silhueta Simplificada opera sobre o grupo em que o objeto se encontra e o grupo mais próximo. Definir em qual grupo o objeto se encontra resultaria, no mínimo, em uma tarefa *map* extra. Por esse motivo, nesse trabalho assumimos que o objeto pertence ao grupo cuja menor distância é calculada.

a saída é um conjunto de pares de tamanho $|P|$. Ao final da execução de todas as funções *map*, cada par é estruturado individualmente, facilitando a execução de *reduce*, que opera sobre valores baseados na sua *chave*.

Algoritmo 9: Função *map* Silhueta Simplificada SF-EAC.

Require: $\mathbf{x}_i$, $\mathbf{C}$ e $|P|$;
1: **for all** P partições **do**
2: **for all** centroides da i-ésima partição **do**
3: calcule a dissimilaridade entre o objeto $\mathbf{x}_i$ e seus centroides (armazenados em $\mathbf{C}$);
4: armazene em id o identificador do grupo mais próximo;
5: armazene em a a distância entre o objeto e o grupo mais próximo;
6: armazene em b a distância entre o objeto e o segundo grupo mais próximo;
7: **end for**
8: $silhueta \leftarrow (b-a)/max\{a,b\}$;
9: instancie uma estrutura de dados $silhueta+$ com a silhueta do objeto, sua partição π e o número de objetos mapeado ($n_S = 1$);
10: crie e armazene um par $id/silhueta+$ para a partição atual;
11: **end for**
12: **return** todos os pares $id/silhueta+$ do objeto $\mathbf{x}_i$;

Após o processamento feito por *map*, os pares com mesmo *id* são mesclados e enviados para as funções *reduce*[6]. Cada função *reduce* é responsável por calcular a Silhueta Simplificada de um grupo, e o retorno de cada uma delas será um par $id/silhuetaG+$, onde id é o identificador do grupo e $silhuetaG+$ é uma estrutura de dados contendo um valor real $silhueta$ que é a Silhueta Simplificada média do grupo, um inteiro π que representa o índice da partição do grupo e um inteiro n_S que é a quantidade de objetos pertencentes ao grupo. Com isso, temos os valores da Silhueta Simplificada média para cada grupo de todas as partições, que serão usados para selecionar grupos que sofrerão mutação (Seção 2.2.3.3). Adicionalmente, é calculada a Silhueta Simplificada de cada partição, que é a média ponderada das silhuetas de seus grupos.

2.2.3.3 Operadores de Mutação

F-EAC possui dois operadores de mutação. O primeiro operador de mutação (MO_1) elimina um ou mais grupos selecionados. Os objetos pertencentes aos grupos eliminados são redistribuídos, individualmente, para o grupo cujo centroide é menos

[6]Nessa etapa é importante ressaltar que as combinações de resultados com mesma *chave* geralmente são feitos localmente, antes de serem enviados à tarefa *reduce*, que é responsável por combinar todos os valores globalmente.

Algoritmo 10: Função *reduce* Silhueta Simplificada SF-EAC

Require: pares $id/silhueta+$ combinados pelo id;
1: inicialize um contador $cont = 0$ e um valor $soma = 0$;
2: **for all** pares $id/silhueta$ **do**
3: $soma \leftarrow soma + silhueta$;
4: $cont \leftarrow cont + n_S$;
5: **end for**
6: **if** $cont = 1$ **then**
7: $silhuetaG \leftarrow 0$;
8: **else**
9: $silhuetaG \leftarrow soma/cont$;
10: **end if**
11: instancie uma estrutura de dados $silhuetaG+$ com o valor da Silhueta Simplificada média $silhuetaG$, sua partição π e o número de objetos de seu respectivo grupo n_S (armazenados em $cont$);
12: **return** o par $id/silhuetaG+$;

dissimilar em relação ao objeto. MO_1 só é aplicado a indivíduos que possuem mais de dois grupos. O valor mínimo de grupos ao final da aplicação do operador é dois. Já o segundo operador de mutação (MO_2) é aplicado a grupos formados por pelo menos dois objetos. Este operador é responsável por dividir um ou mais grupos selecionados em dois novos grupos.

Para reduzir o número de acesso aos dados, o método de implementação dos operadores de mutação no SF-EAC possui algumas diferenças em relação aos originais [Naldi et al., 2011]. O MO_1 apenas elimina os grupos selecionados, de forma que os objetos que pertenciam a estes grupos serão realocados para outros grupos apenas na próxima etapa de busca local (Seção 2.2.3.1). Já o MO_2 elimina os grupos selecionados e duplica seus centroides, criando dois novos grupos com centroides idênticos. Na próxima etapa de busca local, se houver mais de um grupo com a menor dissimilaridade para um determinado objeto, o objeto é sorteado entre estes grupos. Apesar dos operadores implementados serem dependentes da etapa de busca local, não é necessário nenhum acesso direto aos dados durante a aplicação dos operadores de mutação.

O uso dos operadores de mutação do F-EAC implementa uma busca probabilística pelo número de grupos ideal da partição e seus representantes [Alves et al., 2006]. Neste trabalho, é aplicada uma normalização linear sobre os resultados da Silhueta Simplificada obtidos e, em seguida, é usada a estratégia da roleta [Davis, 1996] para seleção dos grupos que sofrerão mutação. Os grupos com valores de Silhueta Simplificada menores tem uma maior chance de serem selecionados neste

processo.

Se o indivíduo selecionado (Passo 13 do Algoritmo 6) pertencer a população inicial ou foi selecionado pelo elitismo, qualquer um dos operadores poderá ser escolhido com a mesma chance. Caso contrário, o operador de mutação a ser aplicado é escolhido proporcionalmente ao seu desempenho dada sua aplicação na geração anterior. Por exemplo, considerando os operadores individualmente, se o uso de um operador gerar uma partição com maior aptidão que seu predecessor, este operador será escolhido novamente para esta partição na próxima geração. Se a aptidão diminua após a mutação, o outro operador será selecionado para mutação na próxima geração (caso esta partição seja selecionada em uma próxima geração).

2.2.3.4 Análise de Complexidade Assintótica

A principal operação realizada no SF-EAC é a execução paralela de múltiplas partições de k-médias. A complexidade dessa operação é dada por $O(n \cdot \hat{k}_{max} \cdot t)$, onde n é o número de objetos do conjunto de dados, $\hat{k}_{max}$ é o valor máximo de k durante a busca evolutiva e t é o número de iterações máximo de k-médias. Considerando que cada objeto do conjunto de dados é usado na função *map* para diferentes partições de tamanho $|P|$ e que SF-EAC é executado por g gerações, estimamos que a complexidade do algoritmo será $O(n \cdot |P| \cdot g \cdot \hat{k}_{max} \cdot t)$. Como as operações são distribuídas ao longo da rede pelas tarefas *map*, podemos assumir n_{max} como o maior número de operações *map* executadas por uma única máquina[7]. Logo, a complexidade final do SF-EAC pode ser dada por $O(n_{max} \cdot |P| \cdot g \cdot \hat{k}_{max} \cdot t)$. Caso não haja falhas na execução das funções *map* e considerando que o modelo *MapReduce* preza pela distribuição uniforme do conjunto de dados pela rede, temos que $n_{max} \approx n/up$, onde up é o número de unidades de processamento distribuídas, o que resulta na complexidade $O(n/up \cdot |P| \cdot g \cdot \hat{k}_{max} \cdot t)$.

2.2.4 *Combination of Fast Evolutionary Algorithm for Clustering* (CF-EAC)

Assim como SF-EAC, o CF-EAC também é uma variante do algoritmo F-EAC. Porém este algoritmo é inspirado na ideia de Combinação de Algoritmos Distribuídos (CAD) [Coelho Naldi & Campello, 2012], que possuem duas etapas: a primeira etapa é a geração de modelos de agrupamentos para representar partes dos conjuntos de dados e o segundo passo é combinar os resultados dos modelos obtidos na

[7] Essas operações *map* podem incluir falhas de operações que ocorreram na mesma ou em outras unidades de processamento.

primeira etapa. Em outras palavras, o CF-EAC tenta encontrar o agrupamento para subconjuntos do conjunto de dados, e a partir daí, obtém o agrupamento global. As operações realizadas sobre os subconjuntos de dados são executadas de maneira local com a finalidade de diminuir a troca de informações na rede. A aplicação de um mapeamento sobre esses subconjuntos é um diferencial do algoritmo, pois, até onde é de conhecimento dos autores, todos os outros algoritmos de agrupamento feitos em *MapReduce* mapeiam objetos individualmente. Tal estratégia objetiva ganho computacional. A desvantagem dela é a restrição do tamanho máximo de cada bloco, que não pode ultrapassar a quantidade de memória principal disponível, tanto para seu armazenamento quanto para suas operações.

As duas etapas do CF-EAC são: a primeira é o agrupamento local dos dados, representado a partir do início do Algoritmo 11 até o Passo 2; a segunda é a combinação dos resultados em um agrupamento global e está descrita do Passo 3 até o final do algoritmo.

Algoritmo 11: *Combination of Fast Evolutionary Algorithm for Clustering* (CF-EAC)

1 Sejam k_{max} o número inicial máximo de grupos, b o número de blocos no qual o conjunto de dados será dividido, $|P|$ o tamanho da população, **m** o vetor com todos os melhores valores resultantes da aplicação de F-EAC sobre cada bloco, $\mathsf{S_{C1}}$ o critério de parada para F-EAC ponderado e **f** o conjunto de centroides final, resultantes da aplicação do F-EAC ponderado. A seguir, o algoritmo CF-EAC:
Require: k_{max}, b, $\mathsf{S_{C1}}$ e $|P|$;
1: dividir o conjunto de dados em b blocos;
2: aplicar F-EAC local para cada bloco de dados paralelamente até que o critério de parada do F-EAC seja satisfeito;
3: instanciar um vetor **m** contendo todos os melhores resultados obtidos de cada bloco b;
4: **f** $\leftarrow$ aplicar F-EAC ponderado utilizando **m** como conjunto de dados até que $\mathsf{S_{C1}}$ seja satisfeito ;
5: **return** o conjunto de centroides finais, armazenados em **f**;

O CF-EAC começa dividindo o conjunto de dados em b blocos (Passo 1), ou seja, um subconjunto do conjunto de dados. Este valor pode ser definido com base na quantidade de memória disponível para trabalho, número de máquinas, processadores, localização dos dados dentro de um sistema de arquivos distribuídos ou automaticamente pela própria plataforma[8].

[8]Como é o caso da implementação feita neste trabalho e gerenciada pela plataforma *Spark*

Vários blocos são executados paralelamente. Diferentemente do padrão de aplicar uma função *map* sobre cada objeto do conjunto de dados, cada função *map* é aplicada sobre um bloco de dados. Com isso, o resultado da função não é uma transformação sobre um único valor, mas sim uma operação sobre um bloco. No CF-EAC, essa operação é o uso do F-EAC local (Algoritmo 6) apresentado na Seção 2.2.3).

Caso não seja possível que todos os blocos executem em paralelo de uma única vez (devido a um limite de memória de trabalho ou unidades de processamento, por exemplo), a função *map* executará novos blocos a medida que os anteriores são completados, de forma a armazenar os resultados para o processamento da segunda etapa do algoritmo. Cada bloco retornará um conjunto de centroides, que representam a melhor solução encontrada pelo F-EAC (Passo 2). Após o término de todas as funções *map*, todos os centroides resultantes da aplicação do F-EAC são armazenados em um vetor de estruturas **m**, onde cada estrutura contém *at*, que armazena valores dos atributos do centroide, e *peso*, que armazena o número de objetos representados pelo centroide.

O próximo passo do CF-EAC é executar o F-EAC sobre os centroides resultantes do mapeamento, de forma que cada centroide seja ponderado em todos os cálculos pelo número de objetos que ele representa (Passo 4). Essa versão do algoritmo segue os mesmos passos do Algoritmo 6, porém durante os Passos 4 e 5, as funções são adaptadas para receberem os centroides e ponderar seus cálculos pelo número de objetos que eles representam. O algoritmo executa até que um critério de parada seja satisfeito. Ao final da execução de F-EAC, haverá uma única solução, ou seja, um agrupamento composto por k centroides que representam todos os dados do conjunto.

Um exemplo de aplicação do CF-EAC é ilustrado na Figura 2.3. O DFS contem o conjunto de dados (representado por pontos), que é distribuído em subconjuntos através dos escravos (unidades de processamento). Na primeira etapa, cada subconjunto é dividido em blocos de dados e é aplicado o algoritmo F-EAC localmente para cada bloco de dados. Os centroides de cada partição resultante (representado pelas estrelas) são transmitidos ao nó mestre, que executa a segunda etapa do algoritmo, aplicando o F-EAC sobre os centroides resultantes, ponderando cada um pelo número de objetos que ele representa. O resultado da segunda etapa é uma partição global dos dados com a solução final do algoritmo.

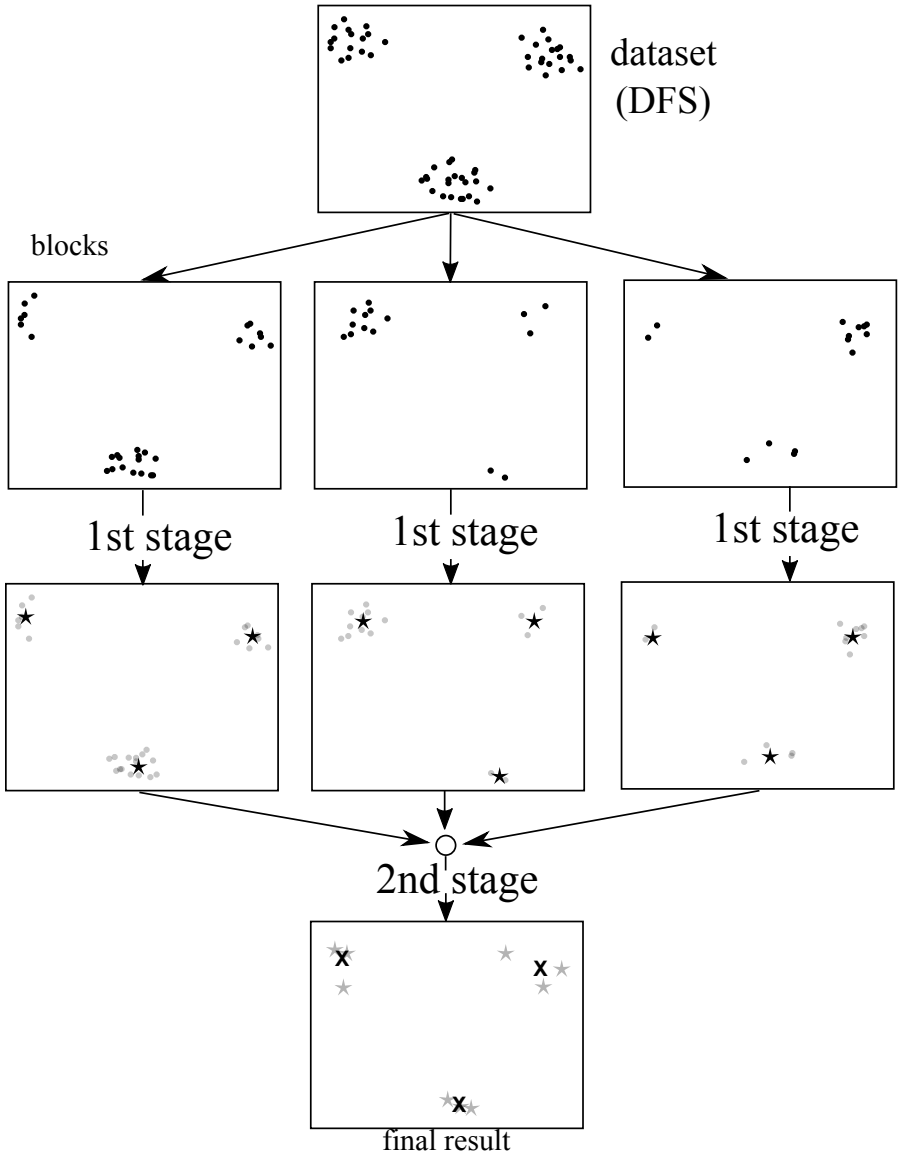

Figura 2.3. Exemplo do fluxo de execução do CF-EAC. Conjunto de dados (pontos), armazenados no DFS, é dividido em blocos. Depois de um agrupamento local, os centroides resultantes (estrelas) são combinados durante a segunda etapa do algoritmo. O agrupamento final é representado pelos novos centroides (simbolizados pelos "X") calculados através dos centroides obtidos na primeira etapa.

2.2.4.1 Análise de Complexidade Assintótica

Assim como o SF-EAC, a complexidade do CF-EAC é dada de acordo com sua principal operação, que é o k-médias. Durante a primeira etapa do algoritmo, podemos assumir que o algoritmo divide o conjunto de dados em b blocos. Assumindo que os dados sejam distribuídos uniformemente entre os blocos, temos que o número de objetos por bloco é $n_b \approx n/b$. Portanto, a complexidade de CF-EAC é dada por

$O(g \cdot |P| \cdot t \cdot \hat{k}_{max} \cdot n_b)$, onde g equivale ao número máximo de gerações realizadas pelo F-EAC local, $|P|$ é o tamanho da população, t é o limite de iterações do k-médias e $\hat{k}_{max}$ é o valor máximo de grupos encontrado durante a busca evolutiva. Na segunda etapa do algoritmo, a complexidade pode ser dada por $O(g \cdot |P| \cdot t \cdot \hat{k}_{max} \cdot n_r)$, onde n_r é a quantidade de objetos resultantes da primeira etapa do algoritmo. Esse valor é dado por $n_r = \sum_{i=1}^{b} |c_i|$, onde $|c_i|$ é o valor de k do i-ésimo bloco. Os valores para g, $|P|$, $\hat{k}_{max}$ e t podem ser diferentes entre a primeira etapa do algoritmo e a segunda.

Na maioria dos cenários, $n_r << n_b$ pode ser assumido. Assim, a complexidade do algoritmo CF-EAC seria dada por $O(g \cdot |P| \cdot t \cdot \hat{k}_{max} \cdot n_b)$.

2.2.5 *G-means*

O *G-means* [Hamerly & Elkan, 2003] é um algoritmo que utiliza k-médias para gerar grupos e encontrar seu número ideal para um determinado conjunto de dados. *G-means* busca a solução com um valor preestabelecido para k inicial (geralmente um valor pequeno). Este valor aumenta conforme os resultados obtidos pelo teste estatístico de Anderson-Darling determinam se o algoritmo valida ou não os grupos da iteração atual.

Quando o algoritmo foi adaptado para *MapReduce* em Debatty et al. [2014], algumas operações foram otimizadas para diminuir o acesso de entrada e saída. Uma vez que o propósito deste trabalho é comparar algoritmos aplicáveis no modelo, iremos assumir a versão *MapReduce* do *G-means* toda vez que o algoritmo for referenciado no resto deste artigo. O Algoritmo 12 apresenta as principais etapas do *G-means*. Os detalhes das operações *map* e *reduce* são discutidos em seguida.

Inicialmente, o algoritmo sorteia um conjunto inicial de protótipos aleatoriamente dentre os objetos do conjunto de dados. Outros métodos de inicialização de k-médias podem ser usados, como mencionado na Seção 2.2.2. Em seguida, o algoritmo k-médias (versão *MapReduce*) é executado sobre o conjunto de dados para refinar os protótipos em $\mathbf{C}$, que tornam-se centroides. O k-médias é aplicado um número pré-definido de iterações t.

Aproveitando o acesso ao conjunto de dados feito pela última iteração *MapReduce* do k-médias, o Passo 5 é responsável por selecionar dois novos protótipos ($\mathbf{c}'_1$ e $\mathbf{c}'_2$) para cada grupo não válido. O método de inicialização destes novos protótipos é feito aleatoriamente, ou seja, dois objetos do grupo são sorteados, como descrito em Debatty et al. [2014]. Com essa combinação, é possível diminuir a quantidade de vezes que o conjunto de dados é acessado.

Algoritmo 12: *MapReduce G-means*

1 Seja **C** uma matriz com todos os centroides e uma *flag* por centroide indicando se o mesmo é válido ou não, $\mathbf{c}'_1$ e $\mathbf{c}'_2$ dois novos protótipos utilizados para dividir um grupo. *G-means* é apresentado a seguir:

```
1: inicialize k protótipos em C como inválidos;
2: repeat
3:    execute k-médias em MapReduce a partir dos protótipos em C;
4:    for all centroides não válidos c_j em C do
5:       gere novos protótipos c'_1 e c'_2 a partir de objetos de grupo representado
         por c_j;
6:       aplique o teste estatístico de Anderson-Darling nos objetos do grupo
         representado por c_j projetados no vetor v = c'_1 − c'_2;
7:       if o teste indicar que o grupo segue distribuição normal then
8:          marcar c_j como válido em C;
9:       else
10:         elimina c_j de C;
11:         insere c'_1 e c'_2 em C como protótipos inválidos;
12:      end if
13:   end for
14: until que todos os protótipos em C sejam válidos;
15: return C;
```

No Passo 6, para cada grupo não válido, cada objeto do conjunto de dados é mapeado e projetado em um vetor de d-dimensões $\mathbf{v} = \mathbf{c}_1 - \mathbf{c}_2$, que passa pelos dois protótipos $\mathbf{c}'_1$ e $\mathbf{c}'_2$. O resultado do mapeamento é um par *id/projeção*, onde *id* representa o identificador do grupo ao qual o objeto está relacionado no agrupamento e *projeção* é a projeção do objeto no vetor. Os valores resultantes relativo a um mesmo grupo são reduzidos por uma função *reduce*, que transforma o conjunto de projeções de forma que sua média seja zero e variância um. Em seguida, o teste de Anderson-Darling é realizado sobre as projeções transformadas para verificar se formam uma distribuição normal, dado um nível de confiança α[9]. Este teste unidimensional detecta a normalidade baseado na Função de Distribuição Acumulada Empírica (FDAE) [Hamerly & Elkan, 2003]. Dado um conjunto $L = \{x_1, x_2, ..., x_n\}$ de n valores reais ordenados, onde x_i é o i-ésimo valor de L, seja F uma FDAE com $N(0, 1)$ e $z_i = F(x_i)$, de forma que $Z = \{z_1, z_2, ..., z_n\}$. A estatística de Anderson-Darling é calculada por A^2 para Z, dado a seguir:

$$A^2(Z) = -\frac{1}{n}\sum_{i=1}^{n}(2i-1)[log(z_i) + log(1 - z_{n+1-i})] - n \tag{2.1}$$

[9] O teste de Anderson-Darling geralmente é aplicado com 95% de confiança, segundo Debatty et al. [2014].

Em casos onde a média μ e variância σ^2 são estimados a partir dos dados (como em agrupamento), a estatística pode ser corrigida [Stephens, 1974] para:

$$A^2_*(Z) = A^2(Z)(1 + 4/n - 25/(n^2)) \tag{2.2}$$

Caso o resultado do teste para o grupo não ultrapasse o valor crítico dado o nível de confiança α pré-estabelecido, esse grupo é considerado válido e marcado com tal em $\mathbf{C}$. Caso contrário, o centroide do grupo é eliminado de $\mathbf{C}$ e os protótipos $\mathbf{c}'_1$ e $\mathbf{c}'_2$ são inseridos em $\mathbf{C}$ e marcados como inválidos.

Ao final, o algoritmo percorre todos os centroides $\mathbf{C}$ verificando se são válidos. Caso todos sejam válidos, retornará os centroides em $\mathbf{C}$, que representam o agrupamento final encontrado, e terminará. Caso contrário, o algoritmo volta ao Passo 3 e tem mais uma iteração.

2.2.5.1 Análise de Complexidade Assintótica

A principal operação do *G-means* é o k-médias, cuja complexidade é $O(n \cdot k \cdot t)$, onde n é o número de objetos do conjunto de dados, k é o número de grupos e t o número máximo de iterações de k-médias. Além da execução de k-médias, *G-means* realiza também o teste estatístico, que calcula no total $O(k \cdot n)$ distâncias e $O(n)$ projeções, no pior caso, em um acesso ao conjunto de dados. Portanto, a operação de maior complexidade aplicada a cada iteração do algoritmo é o k-médias. Assumindo que o algoritmo inicie com um único grupo na iteração 0, durante a i-ésima iteração, o número de grupos que foram testados é dado por Debatty et al. [2014]:

$$k_{total} = 1 + 2 + 4 + ... + 2^i = \sum_{j=0}^{i} 2^j = 2^{i+1} - 1 \tag{2.3}$$

Por fim, podemos concluir que a complexidade assintótica do *G-means* é dada por $O(k_{total} \cdot t \cdot n)$, no pior caso. Assumindo que os dados sejam distribuídos em up unidades de processamento, de forma que o maior número de objetos processados por uma unidade seja n_{max}, a versão *MapReduce* do *G-means* possui complexidade $O(k_{total} \cdot t \cdot n_{max})$. Se $n_{max} \approx n/up$, então podemos considerar que a complexidade é $O(k_{total} \cdot t \cdot n/up)$.

2.2.6 Experimentos

Para compararmos os algoritmos discutidos neste trabalho, foram realizados experimentos com implementações dos algoritmos evolutivos e do *G-means* sobre a

plataforma *Apache Spark* 1.3.1. A plataforma *Apache Hadoop* 2.4 foi utilizada para o armazenamento dos conjuntos de dados no Sistema de Arquivos Distribuídos do *Hadoop* (HDFS), que é acessado pela plataforma *Spark*. A rede utilizada possui 10 computadores, onde cada uma é equipada com processador AMD FX(tm)-6300 com 6 núcleos, 8 GB de memória RAM, 500 GB de HD. O sistema operacional utilizado foi uma distribuição linux com kernel 3.13.0-77-generic.

Os experimentos têm por objetivo comparar os algoritmos em dois aspectos: o tempo computacional e a qualidade das partições resultantes. O tempo computacional é medido ao final da execução de cada algoritmo. Para comparar a qualidade das partições, é usado o Índice *Rand* Ajustado (AR) [Hubert & Arabie, 1985] que compara as partições resultantes a grupos conhecidos. O índice é ajustado para retornar valores no intervalo $[-1, 1]$, de forma que quanto maior for o valor do índice, mais semelhante a partição de referência o resultado é, sendo que 1 indica que são idênticas.

2.2.7 Conjuntos de dados

Para avaliar os algoritmos, conjuntos de dados artificiais e reais foram obtidos. Os conjuntos de dados artificiais possuem grupos com distribuições gaussianas com leve sobreposição, geradas aleatoriamente utilizando o pacote *MixSim* [Melnykov et al., 2012] na linguagem R. As características dos conjuntos de dados criados são:

- Os conjuntos possuem $5 \times 10^6, 10^7$ e $1,5 \times 10^7$ objetos.
- Cada conjunto possui três diferentes valores para k, sendo eles: 5, 10 e 20.
- Os objetos de todos os conjuntos possuem 10 atributos.

O nome de cada conjunto de dados foi definido pelo seu tamanho e o número de grupos. Sendo assim, o conjunto 5×10^6_5c possui 5×10^6 objetos e o número de grupos é 5, por exemplo. Estes nomes foram usados para diferenciar os conjuntos de dados nas Tabelas 2.4, 2.2 e 2.3.

Adicionalmente, foram gerados conjuntos de dados reais sobre coleções de artigos da base de dados *Medline* (*PubMed*) [Roberts, 2001]. Os artigos foram selecionados por meio da ferramenta *PubTator*[10] com base nos conceitos: *Chemical*; *Disease*; *Gene*; *Mutation* e *Species*. Foi realizada uma transformação do corpo textual dos arquivos para *Vector Space Model* (VSM) [Owen et al., 2011] da seguinte forma: os

[10]Disponível em http://www.ncbi.nlm.nih.gov/CBBresearch/Lu/Demo/PubTator/.

conjuntos foram representados no modelo *Bag-of-Words* [Wallach, 2006], com aplicação de *stopwords* [El-Khair, 2006] para remoção das palavras comuns da língua inglesa e *Stemming* de *Porter* [Porter, 1997]. Como o número de termos resultantes ainda foi alto e conjunto de dados esparso, foi aplicado o algoritmo *Latent Dirichlet Allocation* (LDA), presente no *Mahout* [Owen et al., 2011], para delimitação de 5 tópicos. Este pré-processamento resultou em dois conjuntos de dados, sendo eles:

- $Pubmed_1$: executado sobre 10^6 artigos, 10 atributos, sendo 2 termos definidos para cada tópico.
- $Pubmed_2$: executado sobre 10^6 artigos, 50 atributos, sendo 10 termos para cada tópico.

Os conjuntos de dados testados podem não ser considerados massivos, mas são suficientes para estressar o maquinário utilizado.

2.2.7.1 Algoritmos comparados e critérios de parada

Todos os algoritmos comparados neste trabalho executam k-médias para refinar partições a cada iteração. As execuções de k-médias tem seus grupos inicializados com protótipos definidos por objetos sorteados aleatoriamente no conjunto de dados. Nos experimentos deste artigo, o k-médias executa um número máximo de iterações t, definido a *priori*. Evidência empírica sugere que $t = 5$ é suficiente para que o algoritmo encontre soluções satisfatórias [Anderberg, 1973], especialmente considerando que será inicializado múltiplas vezes a partir de protótipos iniciais distintos, portanto este valor será utilizado aqui.

Uma execução do *G-means* trabalha com uma única partição, que vai sendo modificada, acrescentando mais grupos a cada iteração. O valor inicial de k para *G-means* é $k = 2$. Além disso, caso o *G-means* não encontre o número de grupos correto, foi imposto um limite de grupos máximo de grupos válidos $k_{max} = 40$. Como o algoritmo continua dividindo grupos inválidos, o número de grupos válidos pode passar desse limite. Portanto, caso *G-means* superestime ou se perca na busca pelo número de grupos, ele encerra sua execução caso encontre um grupo de grupos válidos maior ou igual à k_{max}. Esse limite foi imposto porque a superestimação do número de grupos foi percebida em algumas execuções do algoritmo apresentadas em Debatty et al. [2014]. Adicionalmente, foi estabelecido o valor de confiança $\alpha = 95\%$ para realização do teste Anderson-Darling, por acreditarmos que o valor é suficientemente rigoroso e também foi utilizado em Debatty et al. [2014]. O *G-means* é executado dez vezes para cada conjunto de dados.

Pelos resultados presentes em Naldi et al. [2011], é possível observar que grandes populações fazem com que o F-EAC convirja para bons resultados em um número menor de gerações que, por sua vez, são computacionalmente mais custosas em relação a populações menores. Esses resultados sugerem que o algoritmo é robusto à escolha do tamanho da população $|P|$, especialmente se considerado que sua eficiência em encontrar soluções é pouco afetada por essa escolha. Os experimentos mostrados em Naldi et al. [2011] indicam que $|P| = 10$ é uma escolha sensata para o algoritmo. Como o SF-EAC é uma variante do F-EAC, iremos assumir esse tamanho de população. O número de grupos das partições inicias do algoritmo estão entre $k_{min} = 2$ e $k_{max} = 40$, o mesmo intervalo aplicado ao *G-means*. Contudo, é importante notar que SF-EAC não é limitado por esses parâmetros, uma vez que a busca evolutiva pode encontrar agrupamentos com valores superiores à k_{max} durante a execução do algoritmo. O tempo computacional do algoritmo é medido ao final de sua execução. O algoritmo também é executado dez vezes para cada conjunto de dados.

Para o CF-EAC, o número de blocos foi definido em um para cada unidade de processamento disponíveis na rede, ou seja, são 6 núcleos em 10 computadores que totalizam $b = 60$. Os valores para os parâmetros do F-EAC (aplicado em ambas as etapas do CF-EAC) são os mesmos determinados para o SF-EAC. O tempo computacional é contabilizado ao final do algoritmo e o mesmo é executado dez vezes para cada conjunto.

2.2.7.2 Resultados

Para observar a significância dos resultados experimentais, dois testes de hipóteses foram adotados. O primeiro, teste Análise de Variância (ANOVA) [Walpole et al., 2006], que assume que as amostras comparadas vem de populações com distribuição normal e variâncias similares [Demšar, 2006]. Como estes requisitos não são garantidos aqui, foi aplicado também um teste não paramétrico de Friedman [Hollander & Wolfe, 1999], ambos com 95% de confiança. Nos casos em que a hipótese nula foi rejeitada, o que indica evidência estatística para apoiar que os algoritmos possuem resultados significantemente diferentes, a correção de Bonferroni [Hochberg, 1988] foi aplicada sobre os valores críticos para compensar as múltiplas comparações e manter o nível de confiança.

A média e o desvio padrão (entre parênteses) das 10 execuções de cada algoritmo são apresentadas nas Tabelas 2.2, 2.3 e 2.4 para os valores resultantes do índice *Rand* Ajustado, tempo computacional e a diferença absoluta do número de

grupos, respectivamente. O melhor valor médio e os valores que não possuem diferença de significância estatística para o melhor valor (para ambos os testes) estão destacados em negrito.

Tabela 2.2. Média e desvio padrão do índice *Rand* ajustado dos algoritmos comparados.

Datasets	**G-means**	**SF-EAC**	**CF-EAC**
5×10^6_5c	0.40 (0.31)	**0.99 (0.00)**	0.86 (0.07)
5×10^6_10c	0.74 (0.18)	**0.96 (0.05)**	0.89 (0.06)
5×10^6_20c	**0.85 (0.06)**	**0.83 (0.07)**	0.60 (0.09)
10^7_5c	0.71 (0.32)	**0.99 (0.00)**	0.86 (0.06)
10^7_10c	0.77 (0.11)	**0.87 (0.07)**	0.77 (0.10)
10^7_20c	**0.86 (0.05)**	**0.85 (0.04)**	0.61 (0.10)
1.5×10^7_5c	0.47 (0.26)	**0.78 (0.00)**	**0.82 (0.07)**
1.5×10^7_10c	0.59 (0.22)	**0.94 (0.06)**	**0.91 (0.06)**
1.5×10^7_20c	**0.87 (0.05)**	**0.82 (0.08)**	0.57 (0.10)
$PubMed_1$	0.19 (0.03)	**0.64 (0.01)**	0.48 (0.04)
$PubMed_2$	0.10 (0.02)	**0.50 (0.07)**	0.42 (0.04)

Como pode ser visto na Tabela 2.2, o SF-EAC obteve o resultado de melhor qualidade média ou sem diferença significante para o melhor resultado em todos os conjuntos de dados, segundo o teste aplicado sobre os valores de AR. Isso mostra que o algoritmo é robusto pois foi capaz de manter resultados de qualidade, independentemente das variações nos conjuntos de dados utilizados. Embora o CF-EAC não consiga obter a mesma qualidade que o SF-EAC, ele foi o segundo algoritmo com mais resultados de melhor qualidade ou sem diferenças significantes para o melhor, segundo o teste aplicado. CF-EAC obteve seus piores resultados em os conjuntos de dados com maior número de grupos, causado pela combinação (segunda etapa do CF-EAC) de um ou mais protótipos de grupos distintos (gerados na primeira etapa do algoritmo). Por último, *G-means* obteve alguns dos melhores resultados para os conjuntos com maior número de grupos, diferentemente do CF-EAC. Contudo, o algoritmo obteve resultados ruins em outros conjuntos de dados, em especial os conjuntos de dados com maior sobreposição, como é o caso do conjunto de dados PubMed.

Para entender melhor o desempenho dos algoritmos comparados, é interessante analisar os números de grupos k encontrados em seus resultados. As médias e desvios padrões da diferença absoluta entre o número de grupos encontrados e o número de grupos real são apresentados na Tabela 2.3. Pelos valores apresentados, é possível ver que *G-means* em *MapReduce* muitas vezes superestima o número de grupos, como

também é visto em Debatty et al. [2014]. Isso fez com que seus melhores resultados sejam obtidos em conjuntos de dados com maior número de grupos. Mesmo assim, os grupos formados pelo algoritmo tendem a ser subdivisões dos grupos de referência, o que justifica os valores razoáveis de AR obtidos pelo algoritmo na Tabela 2.2. Um agravante é fato do algoritmo ter dificuldades com grupos sobrepostos, em especial os conjuntos de dados reais, o que causa a superestimação do número de grupos. Diferentemente, o número de grupos encontrado pelo SF-EAC foi muito próximo do número de grupos de referência para todos os conjuntos de dados, inclusive os conjuntos de dados reais. O CF-EAC obteve desempenho inferior ao SF-EAC, porém melhor que o *G-means*. Acreditamos que o uso do índice Silhueta Simplificada como guia para encontrar o número de grupos é o diferencial dos algoritmos evolutivos, o que confirma os resultados obtidos com o uso deste índice em Vendramin et al. [2010].

Tabela 2.3. Diferença absoluta média e desvio padrão do número de grupos encontrados k para cada um dos algoritmos comparados.

Datasets	**G-means**	**SF-EAC**	**CF-EAC**
5×10^6_5c	34.30 (21.69)	**0.00 (0.00)**	5.16 (2.06)
5×10^6_10c	15.96 (14.93)	**0.33 (0.60)**	1.83 (2.06)
5×10^6_20c	14.86 (8.93)	**1.66 (2.48)**	7.06 (1.76)
10^7_5c	17.36 (22.45)	**0.00 (0.00)**	5.13 (1.67)
10^7_10c	23.86 (15.96)	**1.16 (0.74)**	**0.63 (2.12)**
10^7_20c	14.06 (7.65)	**1.76 (1.83)**	6.70 (1.74)
1.5×10^7_5c	38.46 (22.64)	**1.00 (0.00)**	**2.83 (2.90)**
1.5×10^7_10c	26.30 (15.64)	**0.40 (0.85)**	2.50 (1.33)
1.5×10^7_20c	11.06 (8.02)	**2.73 (2.77)**	7.36 (1.84)
$PubMed_1$	105.50 (39.93)	**0.03 (0.18)**	4.03 (1.82)
$PubMed_2$	90.33 (20.20)	**2.46 (2.40)**	6.03 (1.75)

Os resultados presentes na Tabela 2.4 mostram que o CF-EAC obteve as melhores médias para todos os conjuntos de dados. Isso se dá pelo fato do algoritmo fazer uma menor quantidade de transferências de dados durante sua execução, além do volume de dados transferido ser menor. Como ele subdivide o conjunto em blocos e os processa ao mesmo tempo, só o resultado do agrupamento local precisa ser transferido, o que caracteriza a inovação do algoritmo. Diferentemente do CF-EAC, os outros algoritmos precisam constantemente fazer as transferências de informação durante o fluxo de execução do modelo *MapReduce*. Entretanto, por ser uma aproximação do SF-EAC, o CF-EAC não analisa diretamente todos os objetos do conjunto, o que resultou em uma perda de qualidade em algumas execuções quando

comparado com o SF-EAC. O *G-means* resulta em um excelente tempo computacional quando não superestima o número de grupos do conjunto de dados. Contudo, se o número de grupos é superestimado, o algoritmo tende a executar iterações a mais do que seria necessário, o que faz com que com que seu tempo de execução seja superior aos outros algoritmos comparados. Na prática, isso ocorre com frequência, especialmente em conjuntos de dados com maior sobreposição de grupos. Por fim, as médias dos tempos computacionais para o SF-EAC são menores se comparados ao *G-means* para a maioria dos conjuntos de dados.

Tabela 2.4. Média e desvio padrão dos tempos computacionais (em segundos) dos algoritmos comparados.

Datasets	**G-means**	**SF-EAC**	**CF-EAC**
5×10^6_5c	2362.2 (3366.8)	1257.1 (241.2)	**300.6 (21.4)**
5×10^6_10c	1249.9 (1025.9)	1596.5 (196.1)	**318.3 (50.1)**
5×10^6_20c	1974.5 (2080.3)	2142.1 (346.1)	**315.2 (16.6)**
10^7_5c	1772.3 (2029.7)	1415.4 (317.7)	**539.0 (30.9)**
10^7_10c	3000.1 (3045.9)	2484.6 (359.3)	**588.8(44.7)**
10^7_20c	4141.1 (8707.0)	2446.9 (544.2)	**557.9 (38.2)**
1.5×10^7_5c	2707.2 (1723.2)	1525.7 (302.7)	**846.7 (42.4)**
1.5×10^7_10c	4109.1 (5578.0)	1945.3 (317.8)	**857.5 (38.7)**
1.5×10^7_20c	1928.8 (1205.6)	2875.5 (495.8)	**877.8 (76.4)**
$PubMed_1$	918.6 (217.9)	363.4 (31.0)	**62.4 (3.1)**
$PubMed_2$	849.5 (125.2)	510.9 (72.7)	**135.5 (11.0)**

2.2.8 Conclusão

O algoritmo *G-means* é conhecido por ter bons resultados para conjuntos de dados com grupos que seguem distribuições gaussianas bem distribuídas (sem sobreposição) [Hamerly & Elkan, 2003]. Contudo, sua versão *MapReduce* superestima com frequência o número de grupos dos conjuntos de dados, como pode ser visto nos experimentos feitos neste trabalho e em Debatty et al. [2014]. Essa superestimação ocorre com maior frequência em conjuntos de dados com grupos sobrepostos. Portanto, a versão *MapReduce* do algoritmo é recomendada para cenários com grupos bem comportados (gaussianos e distribuídos).

Diferentemente de Oliveira & Naldi [2015], neste trabalho, o SF-EAC foi avaliado comparativamente em relação a grupos de referência (conhecidos) de novos conjuntos de dados. Os resultados obtidos mostram que o algoritmo foi o que obteve os resultados de melhor qualidade média ou sem diferença significante para o melhor em todos os conjuntos de dados, segundo teste aplicado. Isso mostra que

o algoritmo é robusto (qualidade varia pouco) a mudanças nas características dos conjuntos de dados, de forma que também foi o algoritmo que mais se aproximou dos valores de referência para o número de grupos k.

Apesar de obter resultados próximos do SF-EAC, porém inferiores, o CF-EAC foi o algoritmo mais rápido dentre os comparados para todos os conjuntos de dados. Isso se deve ao seu método de mapeamento diferenciado, que é aplicado a um subconjunto de objetos, aumenta o uso das unidades de processamento e reduz consideravelmente a quantidade de dados transmitidos, além do número de transmissões. Com isso, o algoritmo atinge seu propósito.

Agradecimentos

Os autores agradecem as agências de pesquisa brasileiras CAPES, CNPq, FAPESP e FAPEMIG pelo suporte financeiro. Também agradecemos aos autores de Debatty et al. [2014] pelo código cedido e o Gustavo Avelar pelos conjuntos de dados PubMed.

Capítulo 3

CONCLUSÕES GERAIS E TRABALHOS FUTUROS

Neste trabalho foi apresentado um estudo sobre algoritmos de agrupamentos que possam ser adaptados para o modelo escalável *MapReduce* e que sejam capazes de agrupar os dados sem a especificação prévia do número de grupos k.

Todas as técnicas estudadas e desenvolvidas são baseadas no uso de um dos algoritmos mais influentes na área, o k-médias. O algoritmo geralmente é escolhido para o cenário escalável por possuir complexidade linear e respeitar limitações impostas por modelos como *MapReduce*.

Foram propostos dois novos algoritmos evolutivos em *MapReduce* para a solução do agrupamento escalável. Ambos foram inspirados por algoritmos que apresentaram bons resultados no passado [Alves et al., 2006; Naldi et al., 2011; Naldi & Campello, 2014], porém com maior paralelismo, pois a aplicação de todas as etapas que acessam os dados é feita totalmente em paralelo para todos os indivíduos da população, o que não foi feito em algoritmos anteriores. Além disso, por seguirem o modelo *MapReduce*, sua distribuição e balanceamento de carga são feitos de maneira completamente automática, independentemente do tamanho do conjunto de dados ou topologia do sistema computacional utilizado. Também foi estudado um algoritmo da literatura que é capaz de encontrar automaticamente o numero de grupos k para cenários escaláveis.

A partir dos resultados obtidos nesse trabalho, podemos concluir que o primeiro algoritmo evolutivo proposto, o *Scalable Fast Evolutionary Algorithm for Clustering* (SF-EAC) encontra soluções de qualidade igual ou superior ao MRMk-means em menor tempo computacional.

3.1 Trabalhos Futuros

Como trabalhos futuros para o agrupamento de dados em cenário escalável, com determinação automática do número de grupos k, pode-se incluir diferentes tipos de inicialização de k-médias, como por exemplo, o uso do k-*means*++ [Bahmani et al., 2012]. Esse tipo de otimização pode, por exemplo, diminuir o número de iterações necessárias para os algoritmos encontrarem boas soluções.

Outro trabalho futuro seria a adaptação dos algoritmos estudados para lidar com conjuntos de dados relacionais [Rabbany et al., 2012], ou seja, conjuntos compostos apenas pelas relações entre os objetos e não pelas características dos mesmos. Como só existem essas relações, não é possível calcular centroides. Portanto, é preciso utilizar algoritmos relacionais como por exemplo, o k-medoides [Kaufmann & Rousseeuw, 1987].

Também pode ser feito um estudo de escalabilidade dos algoritmos. O modelo *MapReduce* possibilita a utilização de milhares de processadores, porém durante a pesquisa apenas uma pequena rede foi utilizada. Além disso não houve estudo sobre a quantidade de tarefas *map* e *reduce* que falharam durante suas execuções ou sobre a carga de transferência de dados que foi utilizada.

Referências Bibliográficas

Alves, V.; Campello, R. & Hruschka, E. (2006). Towards a fast evolutionary algorithm for clustering. In *IEEE Congress on Evolutionary Computation, 2006, Vancouver, Canada*, pp. 1776–1783.

Anderberg, M. R. (1973). *Cluster Analysis for Applications*. Academic Press Inc.

Bahmani, B.; Moseley, B.; Vattani, A.; Kumar, R. & Vassilvitskii, S. (2012). Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622--633.

Chen, M.-S.; Han, J. & Yu, P. S. (1996). Data mining: An overview from a database perspective. *IEEE Trans. on Knowl. and Data Eng.*, 8(6):866--883.

Coelho Naldi, M. & Campello, R. J. G. B. (2012). Combining information from distributed evolutionary k-means. In *Proceedings of the 2012 Brazilian Symposium on Neural Networks*, SBRN '12, pp. 43--48, Washington, DC, USA. IEEE Computer Society.

Davis, L. (1991). *Handbook of genetic algorithms*. VNR computer library. Van Nostrand Reinhold.

Davis, L. (1996). *Handbook of Genetic Algorithms*. International Thomson Computer Press.

de Vega, F. & Cantú-Paz, E. (2010). *Parallel and Distributed Computational Intelligence*, volume 269 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Dean, J. & Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pp. 10--10, Berkeley, CA, USA. USENIX Association.

Debatty, T.; Michiardi, P.; Thonnard, O. & Mees, W. (2014). Determining the k in k-means with MapReduce. In *ICDT 2014, 17th International Conference on Database Theory, in conjunction with EDBT/ICDT 2014, 24-28 March 2014, Athens, Greece*, Athens, GRÈCE.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1--30.

Ducournau, A.; Bretto, A.; Rital, S. & Laget, B. (2012). A reductive approach to hypergraph clustering: An application to image segmentation. *Pattern Recogn.*, 45(7):2788--2803.

El-Khair, I. A. (2006). Effects of stop words elimination for arabic information retrieval: a comparative study. *International Journal of Computing & Information Sciences*, 4(3):119--133.

Garcia, K. D. & Naldi, M. C. (2014). Multiple parallel mapreduce k-means clustering with validation and selection. In *Intelligent Systems (BRACIS), 2014 Brazilian Conference on*, pp. 432–437, São Carlos, SP, Brazil.

Hamerly, G. & Elkan, C. (2003). Learning the k in k-means. In *In Neural Information Processing Systems*, p. 2003. MIT Press.

Hamstra, M.; Karau, H.; Zaharia, M.; Konwinski, A. & Wendell, P. (2015). *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, Incorporated.

Hochberg, Y. (1988). A sharper bonferroni procedure for multiple tests of significance. *Biometrika*, 75(4):800–802.

Hochberg, Y. & Tamhane., A. C. (1987). *Multiple Comparison Procedures*. John Wiley & Sons.

Hollander, M. & Wolfe, D. A. (1999). *Nonparametric Statistical Methods*. Wiley-Interscience.

Hruschka, E. R.; Campello, R. J. G. B. & de Castro, L. N. (2004). Evolutionary algorithms for clustering gene-expression data. In *Proc. IEEE Int. Conf. on Data Mining*, pp. 403–406, Brighton/England.

Hruschka, E. R.; Campello, R. J. G. B. & de Castro, L. N. (2006). Evolving clusters in gene-expression data. *Inf. Sci.*, 176(13):1898--1927.

Hubert, L. & Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2(1):193--218.

Jain, A. K. & Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Jain, A. K.; Murty, M. N. & Flynn, P. J. (1999). Data clustering: a review. *ACM Comput. Surv.*, 31(3):264--323.

Kaufmann, L. & Rousseeuw, P. J. (1987). Clustering by means of medoids.

Liu, S. & Cheng, Y. (2012). Research on k-means algorithm based on cloud computing. In *Computer Science Service System (CSSS), 2012 International Conference on*, pp. 1762–1765.

Manyika, J.; Chui, M.; Brown, B.; Bughin, J.; Dobbs, R.; Roxburgh, C. & Byers, A. H. (2011). Big data: The next frontier for innovation, competition, and productivity. Technical report, McKinsey Global Institute.

Melnykov, V.; Chen, W.-C. & Maitra, R. (2012). Mixsim: An r package for simulating data to study performance of clustering algorithms. *Journal of Statistical Software*, 51(12):1–25.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. A Bradford book. Bradford Books.

Müller, F.-J. J.; Laurent, L. C.; Kostka, D.; Ulitsky, I.; Williams, R.; Lu, C.; Park, I.-H. H.; Rao, M. S.; Shamir, R.; Schwartz, P. H.; Schmidt, N. O. & Loring, J. F. (2008). Regulatory networks define phenotypic classes of human stem cell lines. *Nature*, 455(7211):401--405.

Naldi, M. C. & Campello, R. J. G. B. (2014). Evolutionary k-means for distributed data sets. *Neurocomputing*, 127:30–42.

Naldi, M. C.; Campello, R. J. G. B.; Hruschka, E. R. & Carvalho, A. C. P. L. F. (2011). Efficiency issues of evolutionary k-means. *Appl. Soft Comput.*, 11(2):1938--1952.

Narasimhamurthy, A.; Greene, D.; Hurley, N. & Cunningham, P. (2010). Partitioning large networks without breaking communities. *Knowledge and Information Systems*, 25(2):345–369.

O'Driscoll, A.; Daugelaite, J. & Sleator, R. D. (2013). 'Big data', Hadoop and cloud computing in genomics. *Journal of Biomedical Informatics*.

Oliveira, G. V. d. & Naldi, M. C. (2015). Scalable fast evolutionary k-means. In *Intelligent Systems (BRACIS), 2015 Brazilian Conference on*, pp. 74–79, Natal, RN, Brazil.

Owen, S.; Anil, R.; Dunning, T. & Friedman, E. (2011). *Mahout in Action*. Manning Publications Co., Greenwich, CT, USA.

Porter, M. F. (1997). Readings in information retrieval. chapter An Algorithm for Suffix Stripping, pp. 313--316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Rabbany, R.; Takaffoli, M.; Fagnan, J.; Zaane, O. R. & Campello, R. J. G. B. (2012). Relative validity criteria for community mining algorithms. *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, 0:258–265.

Roberts, R. J. (2001). Pubmed central: The genbank of the published literature. *Proceedings of the National Academy of Sciences of the United States of America*, 98(2):381–382.

Schroeck, M.; Shockley, R.; Smart, J.; Romero-Morales, D. & Tufano, P. (2012). Analytics: The real-world use of big data. Technical report, IBM Institute for Business Value.

Schutt, R. & O'Neil, C. (2013). *Doing Data Science: Straight Talk from the Frontline*. O'Reilly Media, Inc.

Stephens, M. A. (1974). Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730--737.

Strauss, C.; Rosa, M. B. & Stephany, S. (2013). Spatio-temporal clustering and density estimation of lightning data for the tracking of convective events. *Atmospheric Research*, 134(0):87 – 99.

Suarez, A. P.; Trinidad, J. F. M.; Ochoa, J. A. C. & Pagola, J. E. M. (2013). An algorithm based on density and compactness for dynamic overlapping clustering. *Pattern Recognition*, 46(11):3040 – 3055.

Tan, P.-N.; Steinbach, M. & Kumar, V. (2009). *Introdução ao Data Mining Mineração de Dados*. Editora Ciência Moderna Ltda.

Vendramin, L.; Campello, R. J. G. B. & Hruschka, E. R. (2010). Relative clustering validity criteria: A comparative overview. *Statistical Analysis and Data Mining*, 3(4):209--235.

Wallach, H. M. (2006). Topic modeling: beyond bag-of-words. In *Proceedings of the 23rd international conference on Machine learning*, pp. 977--984. ACM.

Walpole, R. E.; Myers, R. & Myers, S. L. (2006). *Probability & Statistics for Engineers & Scientists*. Macmillan.

White, T. (2012). *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 3rd edição.

Wu, X. & Kumar, V. (2009). *The Top Ten Algorithms in Data Mining*. Chapman & Hall/CRC, 1st edição.

Xu, R. & Wunsch, D., I. (2005). Survey of clustering algorithms. *IEEE Trans. on Neural Networks*, 16(3):645–678.

Xu, Y.; Zhang, Y. & Ma, R. (2012). K-means algorithm based on cloud computing. In *Computational Intelligence and Design (ISCID), 2012 Fifth International Symposium on*, volume 2, pp. 363–365.

Yang, Y.; Long, X. & Jiang, B. (2013). K-means method for grouping in hybrid mapreduce cluster. *Journal of Computers*, 8(10).

Zhao, W.; Ma, H. & He, Q. (2009). Parallel k-means clustering based on mapreduce. In Jaatun, M.; Zhao, G. & Rong, C., editores, *Cloud Computing*, volume 5931 of *Lecture Notes in Computer Science*, pp. 674–679. Springer Berlin Heidelberg.