UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO
CIÊNCIA DA COMPUTAÇÃO

VINÍCIUS PIRES DE MOURA FREIRE

NACLUSTER: RESOLVENDO ENTIDADES EM LARGA ESCALA A PARTIR DE MÚLTIPLOS CATÁLOGOS DE ASTRONOMIA

FORTALEZA, CEARÁ

2016

VINÍCIUS PIRES DE MOURA FREIRE

NACLUSTER: RESOLVENDO ENTIDADES EM LARGA ESCALA A PARTIR DE MÚLTIPLOS CATÁLOGOS DE ASTRONOMIA

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como parte dos requisitos necessários à obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Banco de Dados.

Orientador: Prof. Dr. José Antônio F. de Macêdo

Coorientador: Prof. Dr. Fábio André Machado Porto

FORTALEZA, CEARÁ

2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F933n Freire, Vinícius Pires de Moura.
Nacluster: resolvendo entidades em larga escala a partir de múltiplos catálogos de astronomia / Vinícius Pires de Moura Freire. – 2016.
171 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação , Fortaleza, 2016.
Orientação: Prof. Dr. José Antônio F. de Macêdo.
Coorientação: Prof. Dr. Fábio André Machado Porto.

1. Resolução de entidades. 2. Casamento de catálogos. 3. Big data. I. Título.

CDD 005

VINÍCIUS PIRES DE MOURA FREIRE

NACLUSTER: RESOLVENDO ENTIDADES EM LARGA ESCALA A PARTIR DE MÚLTIPLOS CATÁLOGOS DE ASTRONOMIA

Tese submetida ao corpo docente do Centro de Ciências da Universidade Federal do Ceará como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciência da Computação. Área de concentração: Banco de Dados

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. José Antônio Fernandes de Macêdo (Orientador)
Universidade Federal do Ceará

Prof. Dr. Fábio André Machado Porto (Coorientador)
Laboratório Nacional de Computação Científica

Prof^{a}. Dr^{a}. Vânia Maria Ponte Vidal
Universidade Federal do Ceará

Prof. Dr. José Maria da Silva Monteiro Filho
Universidade Federal do Ceará

Prof. Dr. Marco Antônio Casanova
Pontifícia Universidade Católica do Rio de Janeiro

Prof. Dr. Ricardo Lourenço Correia Ogando
Observatório Nacional/MCT e LIneA

Dedico esta tese à minha mãe (em memória).

AGRADECIMENTOS

Agradeço primeiramente a Deus pelas inúmeras proteções recebidas durante todo o doutorado.

Esses longos cinco anos e meio de dedicação me fizeram amadurecer bastante. Deixo meus sinceros agradecimentos aos meus orientadores, José Macêdo e Fábio Porto, pelo grande cuidado e atenção dados para o desenvolvimento deste trabalho. Graças às suas orientações, eu consegui chegar até aqui e realizar este sonho. Espero que esta tese retribua minimamente o fato de terem investido recursos preciosos como tempo, estrutura laboratorial e confiança. Foi um prazer imensurável participar das equipes ARiDa e DEXL, como aluno, pesquisador e como uma pessoa que compartilha o mesmo alicerce de vocês: o objetivo principal de, através da disseminação da educação de excelência e de trabalho árduo e correto, fazer nossa parte para tornar nosso querido país um lugar cada vez melhor, colaborando de fato para a nossa sociedade.

Agradeço à minha família, pelo amor e apoio incondicional. Em especial à minha esposa, por estar sempre presente e me dando força em todos os momentos.

Agradeço imensamente a oportunidade de passar dois anos do doutorado no LNCC. Esta experiência foi muito importante para minha carreira e para o desenvolvimento desta tese. Além disso, construí amizades verdadeiras, como a do Fred, Rodrigo, Adolfo, Enver, João, Douglas, dentre outros. Em especial, quero agradecer ao Daniel Gaspar e Rodrigo Botelho, por ajudarem no desenvolvimento deste trabalho ao implementarem o FRANCE e o AODP, respectivamente; e ao Douglas me ajudar a entender detalhes técnicos do Spark.

Agradeço aos amigos que fiz no ARiDa por serem sempre prestativos e estarem prontos para ajudar em qualquer dificuldade. Dentre eles, destaco os amigos Jeovane Reges, Macedo Maia, Diego Victor, Diego Sá, David Araújo, Régis Pires, Igo Brilhante, Ticiana Linhares, Lívia Almada, José Wellington, Mônica Regina, Flávio Sousa e Leonardo Moreira.

Meus agradecimentos ao Laboratório Interinstitucional de e-Astronomia (LIneA) por nos apresentar o problema e nos fornecer os *datasets* e sua infraestrutura para executar experimentos.

Agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, CAPES, pelo financiamento deste estudo através de quatro anos de bolsa de estudos, e a todos aqueles que não foram citados até aqui, mas que também contribuíram direta ou indiretamente para a realização deste trabalho.

Por fim, agradeço ao Prof. Raimundo Moura, chefe do departamento de Computação da UFPI, por me ceder uma sala para eu realizar as pesquisas do doutorado, bem como minhas atividades profissionais como professor substituto.

RESUMO

Levantamentos astronômicos usam instrumentos poderosos para navegar o céu e identificar objetos de interesse dentro da região pesquisada. Objetos do céu são caracterizados individualmente com coordenadas espaciais, identificando a sua posição no céu, além de outros atributos descritivos. Gerar uma visão integrada do céu com base em catálogos produzidos por diferentes levantamentos enfrenta um difícil problema de casamento de catálogos. Devido às variações na calibração dos instrumentos de captura, a posição de um único objeto celeste pode variar de um catálogo para outro. Além disso, em determinadas regiões densas do céu, este problema é agravado por um grande número de potenciais candidatos ao casamento para cada objeto. As abordagens tradicionais para lidar com esse problema usam uma distância limite de ε para reduzir o número de candidatos correspondentes. Além disso, elas adotam uma abordagem para casar *n* catálogos aos pares, inferindo transitividade no casamento, o que nem sempre possuem. Nesta tese, apresentamos o NACluster, um algoritmo de clusterização não-supervisionado para realizar o casamento de vários catálogos. A estratégia de casamento do NACluster estende o tradicional algoritmo de clusterização *k-means* relaxando o número *k* de clusters (ou seja, objetos celestes casados). Também propomos o *ParallelNACluster*, uma versão paralela do NACluster que se aproveita do particionamento dos dados de entrada, e aceita grandes volumes de dados mesmo utilizando um conjunto de hardware de pequeno porte. Além disso, propomos o *SCIBoundary*, uma nova estratégia para tratamento do casamento de objetos espacialmente separados em partições de dados vizinhas. O *SCIBoundary* permite que obtenhamos resultados equivalentes entre o *NACluster* e o *ParallelNACluster*. Nesta tese, também apresentamos o *AODP*, um *workflow* para realizar o particionamento dos dados em disco local e executar o casamento dos mesmos em ambiente distribuído através do *ParallelNACluster*. Nossos experimentos mostram a eficiência das estratégias centralizada e paralela através de excelentes índices de acertos obtidos nas clusterizações, bem como a eficiência do *SCIBoundary* no tratamento das fronteiras.

Palavras-chave: Resolução de entidades. Casamento de catálogos. Big data.

ABSTRACT

Astronomy surveys use powerful instruments to browse the sky and identify objects of interest within the surveyed region. Sky objects are individually characterized with spatial coordinates, identifying their position in the sky, in addition to other descriptive attributes. Composing an integrated view of the sky based on catalogues produced by different surveys faces a hard problem of matching objects that have been captured by different telescopes. Due to variations on capturing instruments calibration, the sky position of a single sky object may vary from a catalog to the other. Moreover, in particular dense regions of the sky this problem is exacerbated by a huge number of candidate matches for each given object. Traditional approaches for dealing with this problem use a threshold distance of ε to reduce the number of matching candidates. Additionally, they adopt a pairwise approach for matching n catalogues inferring transitivity among matches, which not always hold. In this thesis, we present NACluster a non-supervised clustering algorithm for dealing with sky object matching in multiple catalogues. NACluster matching strategy extends the traditional k-means clustering algorithm by relaxing the number k of cluster (i.e. matched sky objects). We propose the ParallelNACluster, a parallel version of NACluster that takes advantage of partitioning the input data, and accept large volumes of data using a set of conventional hardware. In addition, we propose the SCIBoundary, a new strategy for matching neighboring stars placed in different data partitions. The strategy leads to equivalent solutions in both NACluster and ParallelNACluster. In this thesis, we present also the AODP, a workflow to perform partitioning of data on local disk and run their matching in distributed environment via ParallelNACluster. Our experiments show the efficiency of centralized and parallel strategies, as well as the efficiency of *SCIBoundary* in the treatment of borders.

Keywords: Entity Resolution. Catalogues cross-matching. Big data.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

LISTA DE SIGLAS

PAM	Point Access Method
SAM	Spatial Access Method
PH-tree	PATRICIA-hypercube-tree
RDD	Resilient Distributed Dataset
HDFS	Hadoop Distributed File System
PNC	Polo Norte Celeste
PSC	Polo Sul Celeste
PNG	Polo Norte Geográfico
PSG	Polo Sul Geográfico
RA	Right Ascencion
Dec	Declination
2MASS	Two Micron All Sky Survey
NASA	National Aeronautics and Space Administration
WISE	Wide-field Infrared Survey Explorer
HTM	Hierarchical Triangular Mesh
HEALPix	Hierarchical Equal Area isoLatitude Pixelisation
Q3C	Quad Tree Cube
SGBD	Sistema de Gerenciamento de Banco de Dados
BD	Banco de Dados
Aspects	Association positionnelle/probabiliste de catalogues de sources
SVM	Support Vector Machine
DBSCAN	(Density-Based Spatial Clustering of Applications with Noise)
CLARANS	Clustering Large Applications based upon Randomized Search
PRBP	Partition with Reduced Boundary Points
FoF	Friends-of-Friends
dFoF	Distributed Friends-of-Friends
KD-Tree	K-dimensional tree
RAM	Random Access Memory
FRANCE	(FRAgmeNtador de Catálogos Espaciais
SCIBoundary	**S**elect **C**lusters **I**nfluenced by **B**oundary
AODP	Application Oriented Data Partitioning
QEF	Query Evaluation Framework
QEP	Query Execution Plan
arcseg	segundo de arco
DEXL LAB	Extreme Data Lab
LNCC	Laboratório Nacional de Computação Científica
RAM	Random Access Memory

AMD	Advanced Micro Devices
GB	Gigabytes
CPU	Central Process Unit
LSST	Large Synoptic Survey Telescope

1 INTRODUÇÃO

A evolução da ciência fez o estudo da astronomia mudar muito nos últimos anos. A ciência, cuja investigação de fenômenos naturais se utilizava apenas de uma abordagem experimental, evoluiu com o surgimento de modelos teóricos e, nas últimas décadas, com o início das simulações. Essas simulações têm gerado um grande volume de dados. As observações do céu, que eram feitas através da observação visual em telescópios, passaram a ser feitas com instrumentos complexos que transmitem dados para centros de dados em larga escala, e só então as informações são vistas em computadores. A astronomia deixou de ser visual e agora é digital.

Levantamentos astronômicos usam instrumentos poderosos para navegar o céu e identificar objetos de interesse dentro da região pesquisada. Essas observações dão origem ao mapeamento do céu, conhecido como catálogo. Mais precisamente, um catálogo é um conjunto de dados que contém uma coleção de objetos e suas características, tais como posição, magnitude e cor. Nos catálogos também pode existir uma seção contendo informações mais detalhadas dos objetos: o levantamento espectroscópico. Nele, é possível encontrar informações sobre os *redshifts*, ou o quanto o espectro de um objeto está deslocado, comparado aos objetos que não estão se movendo em relação à Terra, tornando possível calcular a distância entre o corpo celeste e a Terra (SDSS, 2013).

Existem diversos catálogos, frutos de levantamentos astronômicos. Eles caracterizam-se por apresentarem diferentes formatos, esquemas, estruturas de dados, uma vez que são tratados por projetos distintos e independentes. Os catálogos cobrem uma determinada faixa do céu e muitos deles possuem interseções em suas coberturas. Dessa forma, é interessante obter uma visão integrada deles, identificando os objetos em comum. Para isso, é necessário realizar o casamento de dados entre os catálogos, em outras palavras, o *cross-matching* de catálogos. Nesta tese, o termo casamento corresponde aos termos *matching* e *cross-matching* em inglês. Eles têm o mesmo significado. A diferença está apenas no uso, pois o termo *cross-matching* é mais utilizado na astronomia.

Na área de banco de dados, esse problema é identificado como resolução de entidades, no qual seu objetivo é identificar instâncias de objetos a partir de diferentes bases de dados que correspondem à mesma entidade no mundo real (AYAT; AKBARINIA; AL., 2012). A principal característica que identifica um mesmo elemento em catálogos distintos é sua posição no espaço, representada pelo “sistema de coordenadas equatoriais” através das coordenadas declinação ou *dec* (coordenada análoga à latitude das coordenadas geográficas) e da ascensão reta ou *ra* (coordenada análoga

à coordenada geográfica longitude). No entanto, essa identificação de objetos em diferentes bases de dados não é simples, pois devido às variações na calibração dos instrumentos de captura, a posição de um único objeto espacial pode variar de um catálogo para outro. Além disso, em determinadas regiões densas do espaço, este problema é agravado por um grande número de potenciais candidatos ao casamento para cada objeto.

O casamento geralmente é aplicado de forma ponto a ponto, entre dois catálogos diferentes, e gera um único catálogo saída que identifica objetos comuns entre os levantamentos. A maioria dos algoritmos realizam uma junção por proximidade ao selecionar casamentos considerando a menor distância entre objetos através de um raio de busca ε definido pelo usuário. No entanto, quando se deseja calcular um casamento entre três ou mais catálogos, um processo mais cuidadoso deve ser aplicado, como averiguar se a ordem com que os catálogos são escolhidos para realizar as junções pode produzir resultados diferentes ou se existe transitividade no casamento. Problemas de transitividade ocorrem, por exemplo, quando dados três objetos O_1, O_2 e O_3 de diferentes catálogos, O_2 casa com O_1 e O_3, mas O_1 não casa com O_3. Portanto, $O_1 = O_2$, $O_2 = O_3$, mas $O_1 \neq O_3$. Neste cenário, algoritmos que considerem transitividade do relacionamento proporiam a relação: $O_1 = O_2 = O_3$.

1.1 Motivação, Problema, Objetivo e Metodologia

Poucos trabalhos na literatura abordaram casamentos no domínio de pesquisa da astronomia. Particularmente, em (FREIRE et al., 2013) alguns algoritmos de *cross-matching* de catálogos astronômicos são apresentados. Dentre eles, o algoritmo *Q3C Join* (KOPOSOV; BARTUNOV, 2006), que realiza *cross-matching* de dois catálogos através da função *q3c_join(catalogo1.ra, catalogo1.dec, catalogo2.ra, catalogo2.dec, raioEpsilon)*, cujos parâmetros referem-se, respectivamente, às coordenadas *ra* e *dec* do primeiro catálogo, às coordenadas *ra* e *dec* do segundo catálogo, e ao raio de busca utilizado a partir de cada objeto para realizar o casamento.

As abordagens de casamentos tradicionais, como o *Q3C*, realizam casamentos somente entre um par de catálogos por vez. Este fato gera dúvidas sobre a qualidade do resultado do casamento entre *n* catálogos ao usar essa metodologia.

Para adicionar um terceiro catálogo ao casamento, previamente temos que realizar o casamento do primeiro par de catálogos. O resultado do primeiro casamento possui os objetos distantes até ε entre si (os demais objetos com distância maior que ε de qualquer outro objeto não são listados). Ele deve ser usado como entrada para o próximo casamento juntamente com o terceiro catálogo.

No *Q3C*, a função *q3c_join* aceita somente parâmetros *ra* e *dec* de dois catálogos por vez. Como o resultado do primeiro casamento possui as coordenadas do objeto do primeiro catálogo e do objeto do segundo catálogo, devemos escolher os atributos de um dos dois para serem usados como entrada na função *Q3C Join* e encontrar os objetos próximos pertencentes a um terceiro catálogo, por exemplo *q3c_join(catalogo1.ra, catalogo1.dec, catalogo3.ra, catalogo3.dec, raioEpsilon)* ou *q3c_join(catalogo2.ra, catalogo2.dec, catalogo3.ra, catalogo3.dec, raioEpsilon)*. Assim, o resultado do casamento entre 3 catálogos sempre depende:

- da ordem dos casamentos entre pares de catálogos;
- e da escolha de qual dos dois catálogos iniciais é tomado por base para realizar o casamento com o terceiro.

Um exemplo dessa dependência está ilustrado na Figura 1.1 ao considerar uma região do céu contendo objetos de três catálogos serem casados pelo Q3C. Cada catálogo é representado por uma cor diferente.

Na Figura 1.1, o primeiro retângulo corresponde à uma região do céu contendo objetos de três catálogos distintos. Cada cor representa um catálogo e os números representam os identificadores (*ids*) dos objetos. Os demais retângulos (de **a** à **m**) apresentam diferentes resultados do casamento do Q3C aplicado aos mesmos objetos do primeiro retângulo. A variação dos resultados se dá pela ordem em que o casamento foi feito. O par de catálogos entre parênteses na legenda de cada item indica o primeiro casamento binário, cuja saída é usada no casamento com um terceiro catálogo. No entanto, somente os atributos do catálogo sublinhado são considerados nesse novo casamento, pois a função *q3c_join* só aceita parâmetros de um dos dois catálogos. As elipses representam os resultados finais dos casamentos, ou seja, objetos reais.

Ainda sobre a Figura 1.1, constatamos que a ambiguidade está presente no casamento, pois existem cinco resultados distintos. No primeiro deles, ocorreu o casamento dos objetos 1, 4 e 7, ilustrados nas Figuras 1.1a, 1.1e, 1.1j e 1.1m. No segundo resultado, apresentado pelas Figuras 1.1b e 1.1i, existiu apenas o casamento entre 1 e 4. Já no terceiro, o objeto 3 casou com o 5 e os objetos 2, 4, 7 e 6 casaram entre si, resultados ilustrados pelas Figuras 1.1c e 1.1h. O quarto casamento é apresentado nas Figuras 1.1d e 1.1g, onde ocorre o casamento entre os objetos 7, 2 e 6 e entre 3 e 5. E, finalmente, o quinto resultado diferente no casamento entre os três catálogos é apresentado pelas Figuras 1.1f e 1.1l. Notamos ainda que nas Figuras 1.1c, 1.1d, 1.1g e 1.1h os objetos 7 e 6 casam entre si apesar de pertencerem ao mesmo catálogo.

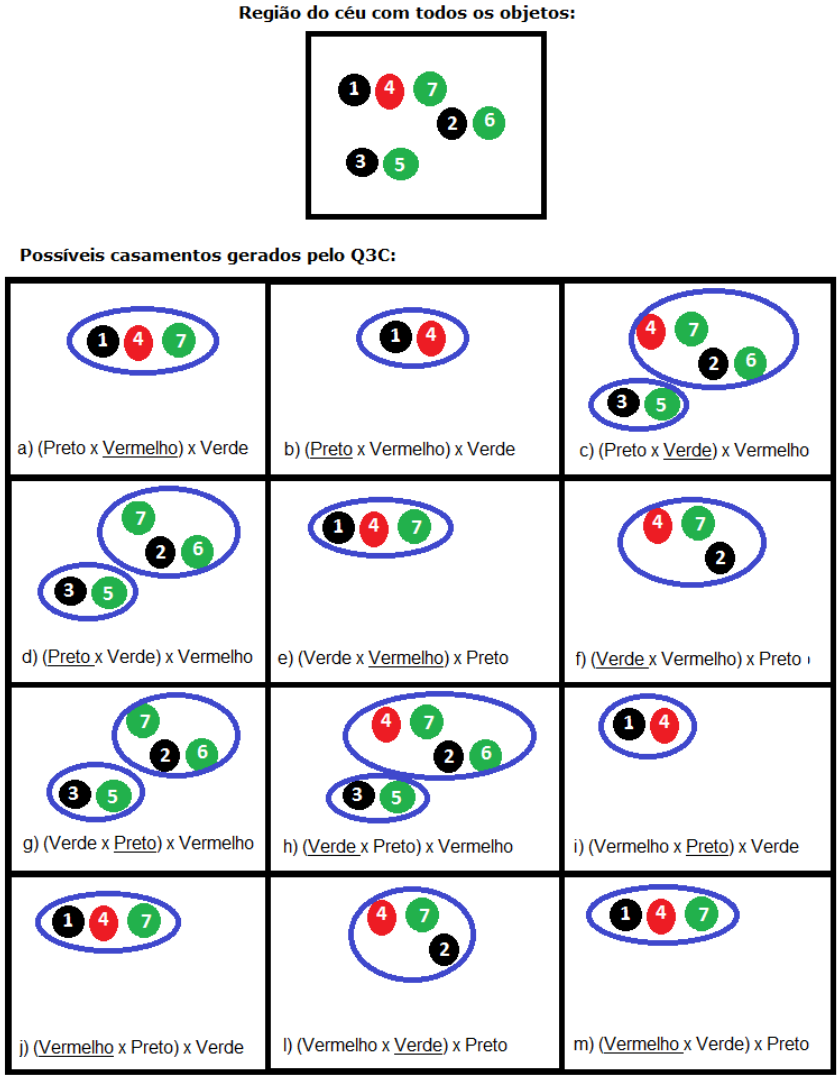


Figura 1.1: Exemplo de casamento de 3 catálogos no Q3C

A dificuldade de realizar o casamento entre *n* catálogos, bem como a diversidade dos resultados encontrados nos métodos de casamento tradicionais despertaram a motivação deste trabalho. A motivação principal é **a necessidade de uma solução de resolução de ambiguidade neste contexto, em especial a resolução de casamentos entre *n* catálogos**. A questão da ambiguidade dos casamentos é um dos problemas em aberto na área e precisa ser explorada.

Outra característica comum do estado da arte dos algoritmos de casamento presentes na literatura é a falta de escalabilidade, pois existe uma limitação física na execução desses algoritmos ao lidar com grandes volumes de dados. Usar dados nessa proporção pode tornar o casamento inviável em tempo de execução nos computadores comuns, demorando vários dias, semanas ou meses (dependendo do volume) ou, até mesmo a inviabilidade pode existir por falta de recursos físicos, como memória. Para aumentar o desempenho, os usuários devem melhorar o poder de pro-

cessamento do seu computador através da aquisição de mais memória ou disco, e de processadores mais velozes. No entanto, como esses algoritmos executam de forma centralizada em uma única máquina, eles são limitados pela tecnologia do computador em uso, e a atualização do hardware pode sair cara, pois muitas vezes necessita-se substituir toda a máquina por um computador caro de grande porte, situação inviável para a comunidade em geral.

A escalabilidade nesta área é desejável devido ao rápido desenvolvimento da tecnologia de captura dos objetos espaciais. Podemos ver a escalabilidade de diferentes formas, como por exemplo, na variação do número de catálogos, da quantidade de objetos em cada catálogo e do número de atributos. É desejável garantir desempenho ao realizar essas variações, logo é importante ter um cluster virtualizado capaz de distribuir as demandas entre as diversas máquinas e garantir a máxima performance da solução em todos seus aspectos. O cluster deve ser capaz de escalar verticalmente, ou seja, adicionando recursos em um único nó do sistema (mais memória ou um disco rígido mais rápido, por exemplo) e escalar horizontalmente, ou seja, adicionar mais nós ao cluster de computadores.

Apesar de sermos motivados pelo problema da astronomia, generalizamos este tema para casamento de catálogos espaciais. Portanto, o problema sob investigação pode ser sintetizado em uma pergunta: **Como realizar o casamento entre "n" catálogos espaciais de forma escalável minimizando a ambiguidade no resultado?**

O objetivo final deste estudo, que corresponde à busca de respostas ao problema enunciado, pode ser expresso como: **Propor uma solução para o casamento entre *n* catálogos de forma escalável**.

Como metodologia para o desenvolvimentos desta tese, optamos por estudar técnicas de clusterização para solucionar nosso problema de resolução de entidades, e utilizar um ambiente distribuído para solucionar o problema da escalabilidade. Embora nossos experimentos sejam feitos com foco no casamento de catálogos astronômicos, as definições apresentadas no Capítulo 5 ajudarão a entender que qualquer catálogo espacial pode ser usado na nossa solução.

É possível dividir o trabalho em 9 estágios de desenvolvimento:

- primeiramente fizemos um levantamento do estado da arte em casamentos de catálogos astronômicos. Publicamos esse levantamento no *VII Brazilian e-Science workshop* (BreSci), em 2013 (FREIRE et al., 2013).
- em seguida, modelamos o problema como casamento de entidades, no qual os objetos espaciais de diferentes catálogos representam a mesma entidade ao

casarem entre si, ou seja, o mesmo objeto do mundo real.

- na terceira parte, propomos uma solução centralizada para o problema, adotando técnicas de clusterização para agrupar os objetos dos catálogos em agrupamentos contendo potenciais casamentos. Os clusters formados são utilizados para determinar a identidade de uma entidade.

- na quarta parte, geramos catálogos sintéticos para validarmos o algoritmo proposto, dado que não existe, até onde conhecemos, um catálogo de referência (em inglês "Golden Standard") de casamento entre catálogos reais e realizamos experimentos para medir a "qualidade" dos agrupamentos produzidos pelo algoritmo.

- na quinta parte, realizamos uma comparação entre o algoritmo proposto e o *Q3C Join* (KOPOSOV; BARTUNOV., 2012). As contribuições trazidas da segunda à quinta parte do nosso trabalho foram publicadas no 10th IEEE International Conference on e-Science, em 2014 (FREIRE et al., 2014).

- em seguida, na sexta parte, propomos uma métrica para avaliarmos a ambiguidade do ponto de vista dos objetos, ou seja, o quanto cada objeto é ambíguo em relação aos clusters, com o objetivo de encontrarmos alguma correlação entre a qualidade da clusterização do NACluster e a ambiguidade existente no agrupamento.

- na sétima parte, propomos a solução distribuída para o nosso problema. Ela realiza a clusterização com as mesmas características da centralizada, mas os dados de entrada precisam ser divididos em partes menores e, em seguida, processados paralelamente.

- na oitava parte, propomos um algoritmo para tratarmos o problema do casamento introduzido ao particionarmos os dados do catálogo em partições distintas. Neste contexto, objetos vizinhos no espaço podem ser alocados em partições distintas.

- finalmente, na nona parte, propomos um *workflow* para particionamento dos dados em disco local e execução do casamento dos mesmos em ambiente distribuído. As contribuições obtidas até então estão sendo organizadas para serem submetidas a um periódico internacional.

O desenvolvimento deste trabalho recebeu o apoio da equipe do (LIneA[1]). A mesma nos apresentou o problema e nos forneceu os *datasets* e sua infraestrutura

[1] http://www.linea.gov.br/

para executar experimentos. O LIneA é Laboratório Interinstitucional de e-Astronomia que envolve o Observatório Nacional, o Laboratório Nacional de Computação Científica, e a Rede Nacional de Ensino e Pesquisa, criado com a finalidade de dar suporte à participação brasileira em experimentos científicos utilizando os dados provenientes de grandes levantamentos astronômicos - os projetos Dark Energy Survey[2] e Sloan Digital Sky Survey III [3]. Para alcançar os objetivos científicos destes projetos, o LIneA gerencia toda uma infraestrutura de armazenamento, processamento, análise e distribuição de dados astronômicos através do projeto Astrosoft. Participam do LIneA pesquisadores e técnicos dos institutos do MCTI mencionados acima, além de professores de universidades.

1.2 Contribuições

Este trabalho tem cinco principais contribuições. A primeira delas é a proposta do algoritmo *NACluster*, uma solução para a resolução de entidades de vários catálogos espaciais através de técnicas de clusterização. O algoritmo proposto realiza o casamento de n catálogos através da clusterização de objetos vizinhos. Na busca por objetos vizinhos, utiliza-se de uma estrutura de indexação muti dimensional, de forma a reduzir a complexidade do processo. Cada cluster produzido representa um objeto real, ou seja, uma entidade, e não deve possuir mais de um objeto do mesmo catálogo. Assim, não se sabe de antemão quantos clusters serão produzidos ao final da execução. Esta restrições nos abstém de adotar um algoritmo de clusterização tradicional, como *k-means*. A qualidade do NACluster é avaliada através da comparação do seu resultado com um catálogo de referência simulado. A clusterização mostrou-se eficiente e com um excelente índice de acerto. Essa contribuição gerou um publicação no 10th IEEE International Conference on e-Science, em 2014 (FREIRE et al., 2014).

A segunda contribuição é a métrica grau médio de ambiguidade. Esta métrica foi criada para avaliarmos a ambiguidade do ponto de vista dos objetos, ou seja, o quanto cada objeto é ambíguo em relação aos clusters, com o objetivo de encontrarmos alguma correlação entre a qualidade da clusterização do NACluster e a ambiguidade existente no agrupamento. Os experimentos com a métrica nos proporcionaram encontrar essa correlação.

A terceira contribuição é a proposta do algoritmo *ParallelNACluster*, uma solução distribuída para a resolução de entidades de vários catálogos espaciais através de técnicas de clusterização. O algoritmo tem o objetivo de reproduzir a clusterização

[2]http://www.darkenergysurvey.org/
[3]https://www.sdss3.org/

do NACluster em um ambiente de larga escala com eficiência. De modo que, inicialmente, produza um particionamento dos dados de entrada tal que as fronteiras entre as partições sejam conhecidas e que haja um balanceamento das partições, ou seja, cada uma delas tenha um número semelhante de objetos e/ou variação de densidades semelhantes, para que NACluster seja executado em cada partição com tempo relativamente próximo. O algoritmo paralelo mostrou-se eficiente e produziu clusters com qualidade semelhante ao centralizado, a diferença mínima apresentada foi em torno de 10^{-4} na precisão do resultado.

A quarta contribuição é a criação do *SCIBoundary*, uma estratégia para o tratamento das fronteiras geradas pelo particionamento e que objetiva encontrar os objetos influenciados por elas. Como o NACluster precisa da vizinhança para realizar o casamento, segundo seu algoritmo, e os dados particionados fazem com que a vizinhança da fronteira de duas partições vizinhas seja separada, é preciso conhecer as fronteiras e os objetos próximos em partições vizinhas para a que execução da clusterização seja feita corretamente sobre eles. Nos experimentos realizados constatamos a viabilidade do *SCIBoundary* e que a qualidade dos clusters produzidos por ele é semelhante aos do NACluster e ParallelNACluster, com uma diferença em torno de 10^{-2} para baixo no valor da precisão, justificada pelo tamanho da amostra ser bem menor que nos demais.

A quinta contribuição é o desenvolvimento do *AODP*, um workflow para particionamento dos dados em disco local e execução do casamento dos mesmos em ambiente distribuído através do *ParallelNACluster*. O objetivo do dataflow é automatizar e acelerar a execução do pré-processamento dos dados.

As contribuições obtidas até então estão sendo organizadas para serem submetidas a um periódico internacional.

1.3 Organização dos capítulos

A seguir apresentamos brevemente a maneira como esta tese está organizada.

No Capítulo 2 apresentamos o conjunto principal de conceitos que será a base para o entendimento deste trabalho. Dentre estes conceitos destacam-se a apresentação mais detalhada de resolução de entidades; uma introdução das técnicas de clusterização, em especial o *K-means*; uma breve apresentação das medidas de qualidade usadas para avaliar os resultados produzidos pelas resolução de entidades; e uma visão geral sobre indexação multidimensional, em particular, em particular dá-

se ênfase à estrutura de indexação multidimensional *PH-Tree*.

O Capítulo 3 apresenta os conceitos, problemas e soluções existente na unificação de catálogos astronômicos que serviram de motivação para definirmos o problema alvo desta tese. Dentre os conceitos importantes, discorremos sobre os catálogos, como eles são produzidos, suas principais características e atributos relevantes. Neste mesmo capítulo, fazemos uma introdução ao problema de unificação de catálogos e abordamos a forma tradicional de unificação, o *cross-matching* aos pares de catálogos. Além disso, apontamos os problemas nas soluções existentes que motivaram as investigações desta tese e os resultados esperados da unificação.

Um apanhado geral das pesquisas relacionadas a algoritmos de clusterização usados para solucionar o problema de resolução de entidades e os algoritmos de casamento presentes na literatura, os quais nos baseamos para desenvolvermos o NACluster, são apresentados no Capítulo 4. Neste mesmo capítulo, tratamos de trabalhos relacionados ao nosso novo método de tratamento das fronteiras, cujo o principal objetivo é identificar os clusters influenciados pela fronteira.

No Capítulo 5 apresentamos a nosso proposta de uma nova estratégia de resolução de entidades de catálogos espaciais através do uso de técnicas de clusterização. Esse estratégia é o algoritmo NACluster, cujo objetivo é agrupar objetos de diferentes catálogos em agrupamentos contendo potenciais casamentos, de forma que cada cluster determine a identidade de uma entidade.

Em seguida, no Capítulo 6, propomos o ParallelNACluster, um algoritmo com o mesmo objetivo do NACluster, mas que funciona de forma paralela, graças ao particionamento dos dados de entrada, e executa grandes volumes de dados mesmo utilizando um conjunto de hardware de pequeno porte. Além disso, propomos o *SCI-Boundary*, uma nova estratégia de tratamento das fronteiras geradas pelo particionamento, cujo objetivo é resolver o problema de objetos vizinhos em partições diferentes não considerarem a existência um do outro, e assim, fazer com que o resultado final do ParallelNACluster seja semelhante ao do NACluster. Neste capítulo também propomos o *AODP*, um *workflow* para realizar o particionamento dos dados em disco local e executar o casamento dos mesmos em ambiente distribuído através do ParallelNACluster.

No Capítulo 7 apresentamos os experimentos realizados nesta tese. Eles são divididos em duas partes. Primeiramente apresentamos os experimentos com o *NACluster* e em seguida com o *ParallelNACluster* e o *AODP*. Na primeira parte analisamos os resultados dos experimentos do NACluster aplicados a diversos *datasets* e medimos a qualidade dos casamentos. Além disso, propomos uma métrica para avaliar a ambiguidade do ponto de vista dos objetos, ou seja, o quanto cada objeto é

ambíguo em relação aos clusters, e realizamos experimentos com o objetivo de encontrarmos alguma correlação entre a qualidade da clusterização do *NACluster* e a ambiguidade existente no agrupamento. Na segunda parte deste capítulo, realizamos experimentos com o *ParallelNACluster*. Avaliamos a qualidade do seu casamento, bem como do casamento produzido pela fronteira. Além disso, analisamos o impacto da mudança dos parâmetros de execução do ambiente distribuído usado para submeter a aplicação e testamos a escalabilidade ao aumentarmos o volume de dados. Neste segunda parte também foram feitos experimentos com o *AODP*.

No Capítulo 8, apresentamos as considerações finais sobre esta tese e as propostas de alguns temas para trabalhos futuros.

Por fim, no Anexo A, apresentamos todas as definições formais na sequência em que elas foram mostradas na tese, com o intuito de facilitar o entendimento, pois os conceitos seguem uma ordem sequencial.

2 FUNDAMENTOS TEÓRICOS

Este capítulo apresenta, de forma sucinta, os principais fundamentos teóricos que foram importantes para o desenvolvimento desta tese. A estrutura do capítulo é indicada a seguir.

Na Seção 2.1 apresentamos a definição geral de resolução de entidades. Ela é importante pois as contribuições trazidas por esta tese tratam-se deste tema.

A Seção 2.2 é destinada a apresentar uma visão geral sobre algoritmos de clusterização, em especial o *K-means*, um dos seus principais algoritmos de aglomeração presentes na literatura. Estes conceitos são úteis, pois no Capítulo 5 propomos um algoritmo de clusterização baseado no *K-means*.

Na Seção 2.3 apresentamos medidas de qualidade usadas na literatura para avaliar os resultados produzidos pelas resolução de entidades. Três das medidas apresentadas são utilizadas no Capítulo 7 para medir a qualidade dos resultados obtidos nos experimentos.

E, finalmente, na Seção 2.4 apresentamos uma visão geral sobre indexação multidimensional, em particular, sobre a *PH-Tree*, pois esta foi utilizada nas implementações propostas nos Capítulos 5 e 6.

2.1 Resolução de Entidades

Integração de banco de dados é o problema de ter duas (ou mais) bases de dados desenvolvidas independentemente e resolver as diferenças entre elas, para fazê-las parecer como uma só. A necessidade de integração pode surgir devido às novas aplicações que abrangem vários bancos de dados, ou devido à integração das operações de diferentes organizações (LIM; SRIVASTAVA; AL., 1993).

Bancos de dados preexistentes na maioria das organizações são definidos e povoados por pessoas diferentes, em momentos diferentes, em resposta a diferentes necessidades organizacionais ou do usuário final. Tal desenvolvimento independente de bancos de dados muitas vezes resulta em diferentes bases de dados que possuem partes do mesmo domínio do mundo real. Normalmente, quando há uma necessidade de fornecer acesso integrado a esses bancos de dados, identificar as representações da mesma entidade do mundo real a partir de duas ou mais bases de dados é muitas vezes difícil, se não impossível, sem especificar a informação semântica adicional que resolve essa ambiguidade (LIM; SRIVASTAVA; AL., 1993).

Esse problema tem sido conhecido há mais de cinco décadas, como *record linkage* ou o problema de *record matching* (NEWCOMBE; KENNEDY; AL., 1959), (NEWCOMBE; KENNEDY, 1962), (NEWCOMBE, 1967), (FELLEGI; SUNTER, 1969), (TEPPING, 1968), (NEWCOMBE, 1988) na comunidade de estatística. O mesmo problema tem vários nomes nas diferentes comunidades acadêmicas (ELMAGARMID; MEMBER, 2007). Na comunidade de banco de dados, por exemplo, o problema é denominado como *merge-purge* (HERNáNDEZ; STOLFO, 1998), eliminação de duplicação de dados (SARAWAGI; BHAMIDIPATY, 2002), identificação de instância (WANG; MADNICK, 1989) e identificação de entidade (LIM; SRIVASTAVA; AL., 1993); na comunidade de inteligência artificial, o mesmo problema é descrito como *database hardening* (COHEN; KAUTZ; MCALLESTER, 2000) e *name matching* (BILENKO; MOONEY; AL., 2003). Resolução de correferência, incerteza de identidade, detecção de duplicados (ELMAGARMID; MEMBER, 2007) e resolução de entidade (AYAT; AKBARINIA; AL., 2012) são também comumente usados para se referir à mesma tarefa (ELMAGARMID; MEMBER, 2007). Este trabalho adota o termo resolução de entidade (ER).

Portanto, ER é o processo de identificação de registros que representam a mesma entidade do mundo real. Por exemplo, considere os dois registros (“Thomas Michaelis”, “Rua Principal, 45”) e (“T. Michaelis”, “R. Principal, 45”) no esquema S (nome, endereço), que representam a mesma pessoa com diferentes convenções. Uma solução de ER pode detectar os registros que representam a mesma entidade. O problema de resolução de entidades é um desafio, já que a mesma entidade pode ser codificada de maneiras diferentes devido a uma variedade de razões, tais como diferentes convenções de formatação, abreviações e erros tipográficos (AYAT; AKBARINIA; AL., 2012).

A dificuldade da ER aumenta consideravelmente quando os bancos de dados são incertos, nos quais existe incerteza na correspondência dos objetos. Incerteza é um estado de conhecimento limitado, onde não se sabe qual das duas ou mais alternativas é verdadeira. No contexto da integração de dados, um exemplo típico de incerteza diz respeito à correspondência de objetos, tais como o fato de que duas colunas de endereço nas diferentes bases coincidem ou não. Manter estas alternativas, leva à produção de vários bancos de dados integrados, um para cada escolha (MAGNANI; MONTESI, 2010), todas essas opções são conhecidas como mundos possíveis.

Fontes não confiáveis, instrumentos de medição imprecisa e métodos incertos, são algumas das razões que causam a incerteza em dados. A principal diferença entre uma base de dados tradicional e uma base de dados incerta é que uma base de dados incerta representa um conjunto de possíveis casos em banco de dados, em

vez de um único caso (AYAT; AKBARINIA; AL., 2012).

2.1.1 Modelo Básico

A definição formal de resolução de entidades (ER) apresentada em (MOLNÁR; BENCZÚR; SIDLÓ, 2012) é mostrada na Definição 1.

Definição 1. *Dado um conjunto de registros* $R = \{r_1, r_2, ...r_m\}$*, onde cada* r_j *consiste de um conjunto de atributos. O objetivo da resolução de entidades é particionar os registros de acordo com as entidades as quais pertencem: Seja* $E = \{e_1, e_2, ...e_n\}$ *um conjunto de entidades, onde cada* e_i *consiste de um subconjunto de registros* $e_i \subseteq R$ *tal que a união das entidades cobre todos os registros,* $\cup_{i=1}^{n} e_i = R$*, e nenhum registro pertence a mais de uma entidade:* $r \in e_i \wedge r \in e_j \Rightarrow i = j$*.*

2.2 Algoritmos de Clusterização

Segundo (BERKHIN, 2006), clusterização, ou agrupamento, é uma divisão de dados em grupos de objetos semelhantes. Cada grupo, chamado de cluster, é composto por objetos que são semelhantes entre si e muito diferente de objetos de outros grupos. A clusterização modela dados através de clusters. A modelagem de dados coloca a clusterização em uma perspectiva histórica enraizada na matemática, estatística e análise numérica. Do ponto de vista prático, o agrupamento desempenha um papel de destaque em aplicações de mineração de dados, tais como a exploração científica de dados, recuperação de informação e de mineração de texto, aplicações de banco de dados espaciais, análise de Web, CRM, marketing, diagnósticos médicos, biologia computacional, e muitos outros.

Um exemplo simples de divisão de dados em grupos está disposto na Figura 2.1 e foi apresentado em (MATTEUCCI, 2013). Neste caso, identifica-se facilmente os 4 grupos em que os dados podem ser divididos; o critério de similaridade é a distância: dois ou mais objetos pertencem ao mesmo conjunto se eles são “próximos” de acordo com uma dada distância (neste caso, a distância geométrica). Esse tipo de agrupamento é chamado de clusterização baseada em distância. Outro tipo de agrupamento é o agrupamento conceitual: dois ou mais objetos pertencem ao mesmo cluster se existe uma característica comum a todos que os objetos. Em outras palavras, os objetos são agrupados de acordo com sua adequação aos conceitos descritivos, e não de acordo com as medidas de similaridade simples.

Segundo (MATTEUCCI, 2013), não existe um “melhor” critério absoluto para decidir o que constitui um bom agrupamento que seja independente do obje-

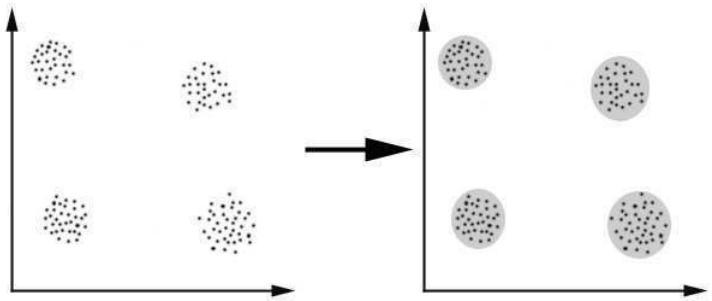


Figura 2.1: Exemplo simples de clusterização (MATTEUCCI, 2013)

tivo final do agrupamento. Portanto, é o usuário que deve fornecer este critério, de tal maneira que o resultado do agrupamento irá atender às suas necessidades. Por exemplo, talvez o usuário possa estar interessado em encontrar representantes de grupos homogêneos (redução de dados), ou em buscar “agrupamentos naturais” e descrever suas propriedades desconhecidas (tipos de dados “naturais”), ou por buscar agrupamentos úteis e apropriados (classes de dados “úteis”) ou ainda em buscar objetos de dados incomuns (detecção de outliers).

Quanto à classificação dos algoritmos de clusterização, (BERKHIN, 2006) não a considera simples, pois as classificações se sobrepõem. São elas:

- métodos hierárquicos;
- métodos de particionamento;
- métodos baseados em grade;
- métodos baseados em coocorrência de dados categóricos;
- clusterização baseada em restrição;
- algoritmos de agrupamento usados em aprendizado de máquina;
- algoritmos de agrupamento escaláveis;
- algoritmos para dados de alta dimensionalidade.

No entanto, conforme (BERKHIN, 2006), as técnicas de clusterização são tradicionalmente, amplamente divididas em hierárquicas e de particionamento. Cada uma deles se subdivide em várias diferentes técnicas. Enquanto algoritmos hierárquicos constroem clusters gradualmente, algoritmos de particionamento aprendem clusters diretamente. Ao fazer isso, eles tentam descobrir clusters iterativamente realocando pontos entre os subgrupos, ou tentam identificar os clusters como áreas

altamente povoadas com dados. De acordo (OCHI; DIAS; SOARES, 2004), os algoritmos do segundo tipo funcionam da seguinte forma: o conjunto de elementos é dividido em *k* subconjuntos, podendo *k* ser conhecido ou não, e cada configuração obtida é avaliada através de uma função objetivo Caso a avaliação da clusterização indique que a configuração não atende ao problema em questão, uma nova configuração é obtida realocando pontos entre os clusters, e o processo continua de forma iterativa até que algum critério de parada seja alcançado. Neste esquema de realocação dos elementos entre os clusters, também conhecido como otimização iterativa (OCHI; DIAS; SOARES, 2004), os clusters podem ser melhorados gradativamente, o que não ocorre nos métodos hierárquicos. O k-médias, ou *k-means* (MACQUEEN, 1967) é um exemplo deste tipo de algoritmo.

(BERKHIN, 2006) ainda comenta que alguns algoritmos que usam técnicas de clusterização de particionamento tentam descobrir componentes conexas densas de dados e estas são flexíveis quanto à sua forma. Um exemplo desse tipo de algoritmo bastante difundido é o DBSCAN (ESTER et al., 1996). Esses algoritmos são menos sensíveis aos *outliers* e podem descobrir conjuntos de formas irregulares. Eles geralmente trabalham com dados de baixa dimensionalidade de atributos numéricos, conhecidos como dados espaciais.

Já na clusterização hierárquica, conforme (OCHI; DIAS; SOARES, 2004), os clusters vão sendo formados gradativamente através de aglomerações ou divisões de elementos clusters, gerando uma hierarquia de clusters, normalmente representada através de uma estrutura em árvore, conforme exemplificado na Figura 2.2. Nesta classe de algoritmos, cada cluster com tamanho maior que 1 pode ser considerado como sendo composto por clusters menores.

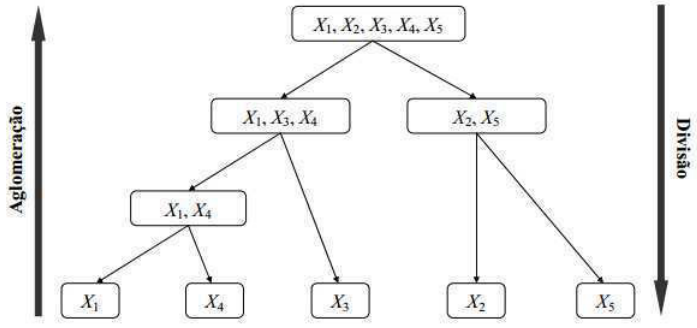


Figura 2.2: Exemplo de árvore de clusters na clusterização hierárquica (OCHI; DIAS; SOARES, 2004)

Nestes algoritmos existem dois tipos de abordagem: *bottom-up* e *top-down*. De acordo com (OCHI; DIAS; SOARES, 2004), nos algoritmos de aglomeração, que

utilizam uma abordagem *bottom-up*, cada elemento do conjunto é, inicialmente, associado a um cluster distinto, e novos clusters vão sendo formados pela união dos clusters existentes. Esta união ocorre de acordo com alguma medida que forneça a informação sobre quais deles estão mais próximos uns dos outros. Nos algoritmos de divisão, com uma abordagem top-down, inicialmente tem-se um único cluster contendo todos os elementos do conjunto e, a cada passo, são efetuadas divisões, formando novos clusters de tamanhos menores, conforme critérios preestabelecidos

Para esta tese é interessante detalhar os algoritmos de clusterização classificados como métodos de particionamento. Essa informação é justificada pelo fato do NACluster, apresentado no Capítulo 5, ser um algoritmo deste tipo, pois a configuração dos seus clusters permite ser avaliada por uma função objetivo, e caso a avaliação da clusterização indique que a configuração não atende ao problema em questão, uma nova configuração é obtida realocando pontos entre os clusters, o que não acontece numa clusterização hierárquica.

2.2.1 K-Means

K-means, ou k-médias, (MACQUEEN, 1967) é um algoritmo bastante difundido na literatura e um dos mais simples algoritmos de aprendizagem não supervisionada que resolvem o problema de clusterização. O objetivo é classificar um conjunto de elementos, dado como entrada, em um número *k* de clusters. A ideia principal é definir *k* centroides, um para cada cluster, geralmente aleatórios, mas a melhor escolha é colocá-los longe um do outro. O próximo passo é levar cada ponto pertencente a um determinado conjunto de dados e associá-lo ao centroide mais próximo. Quando nenhum ponto está pendente, a primeira etapa é concluída e uma clusterização parcial é feita. Neste ponto é preciso voltar a calcular os *k* novos centroides dos clusters resultantes da etapa anterior. Logo após, uma realocação tem de ser feita entre os mesmos pontos e o novo centroide mais próximo. Esses passos são repetidos em loop e os k centroides mudam a sua localização passo a passo até que não haja mais mudanças. Em outras palavras, os centroides não se movem mais (MATTEUCCI, 2013).

O algoritmo visa minimizar uma função objetivo, neste caso uma função do erro quadrado. (MATTEUCCI, 2013) descreve a função objetivo como sendo $J = \sum_{j=1}^{k} \sum_{i=1}^{n} ||x_i^j - c_j||^2$, onde $||x_i^j - c_j||^2$ é a medida de distância escolhida entre um ponto x_i^j e o centroide do cluster c_j. Essa função é um indicador da distância dos n pontos e seus respectivos centroides.

Os passos do algoritmo são (FONTANA; NALDI, 2009):

1. Atribuem-se valores iniciais de centroides seguindo algum critério, por exemplo, sorteio aleatório desses valores dentro dos limites de domínio de cada atributo.
2. Atribui-se cada objeto ao cluster cujo centroide esteja mais próximo ao objeto.
3. Recalcula-se o valor do centroide de cada cluster, como sendo a média dos objetos atuais do cluster.
4. Repete-se os passos 2 e 3 até que os clusters se estabilizem.

Em (FONTANA; NALDI, 2009), apresenta-se a Figura 2.3 ilustrando a execução do *K-means*. Nas Figuras 2.3 (a), 2.3 (b), 2.3 (c), mostra-se a execução dos passos 1, 2 e 3 respectivamente. Na Figura 2.3 (d) apresenta-se a repetição dos passos 2 e 3; e nas figuras 2.3 (e) e 2.3 (f) ilustra-se a repetição dos passos 2 e 3, respectivamente.

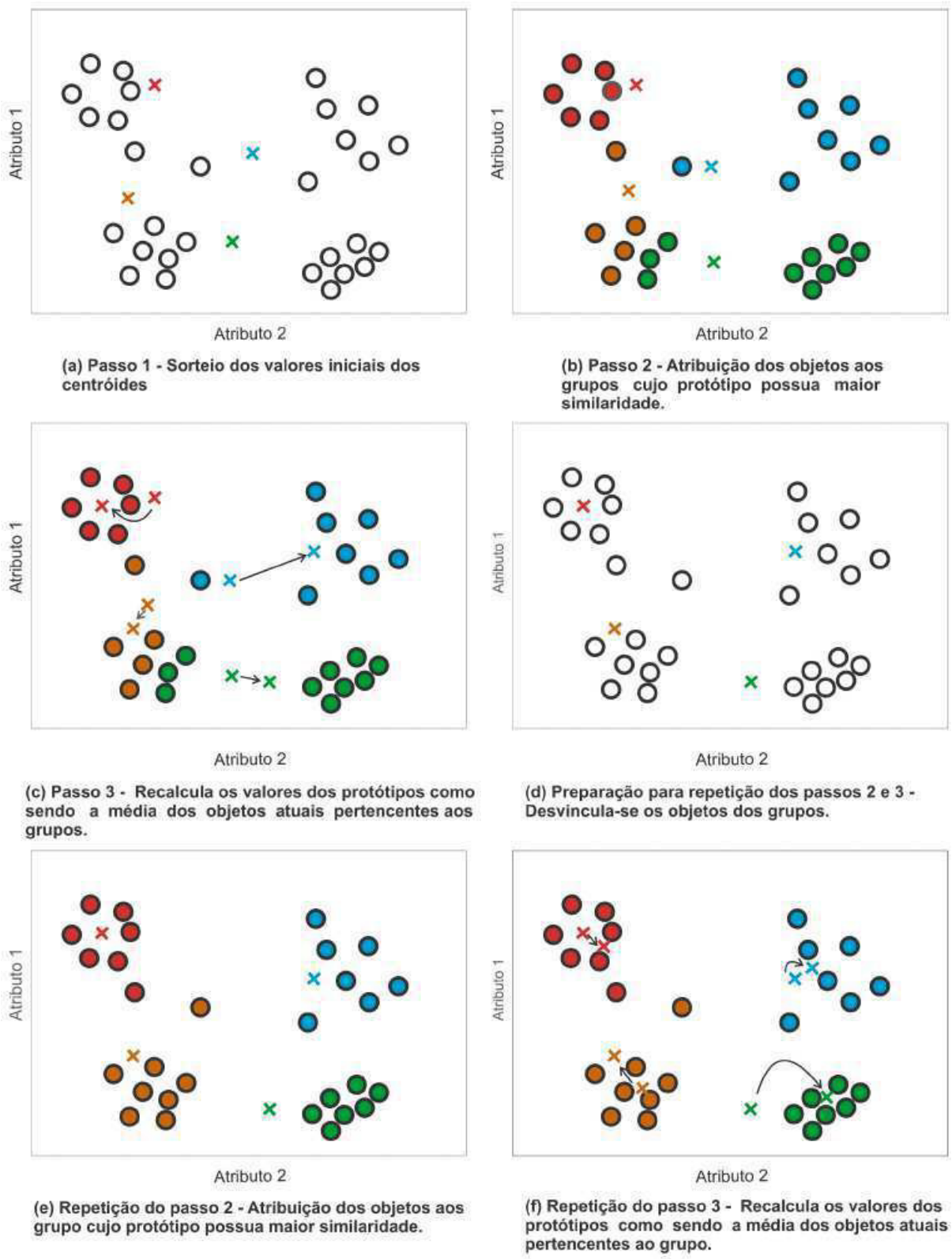


Figura 2.3: Exemplo de execução do k-means. Fonte: (FONTANA; NALDI, 2009)

O *K-means* possui complexidade computacional equivalente a $O(n \times k \times i)$, onde n é o número de objetos, k é o número de clusters e i o número de iterações. A

distância entre os n objetos até cada um dos k centroides é calculada a cada iteração i. O número de dimensões também influência na complexidade do algoritmo, pois se o objeto tem d dimensões, a comparação entre dois desses objetos está em $O(d)$. Logo, para objetos que tenham 2 ou mais dimensões, a complexidade computacional do *K-means* passa a ser $O(n \times k \times i \times d)$.

2.3 Medidas de Qualidade em Resolução de Entidades

É importante avaliar a qualidade dos resultados produzidos pelos algoritmos de resolução de entidades. Geralmente, quando avalia-se a corretude da saída produzida por um algoritmo de ER, compara-se o resultado com um "golden standard". O "golden standard" é o resultado considerado correto da resolução de entidades (WANG et al., 2011).

É fundamental neste tipo de avaliação adotar um método de atribuição de um número para expressar o quão perto a saída do algoritmo de ER é do "golden standard". Foram propostas muitas medidas de qualidade para comparar os resultados de algoritmos de ER (MENESTRINA; WHANG; GARCIA-MOLINA, 2010), mas não há atualmente nenhuma medida padrão acordada para avaliar os resultados de ER.

Por exemplo, considere um problema de resolução de entidade com uma entrada sendo um conjunto registros $I = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}\}$. Três possíveis resultados ER são mostrados na Tabela 2.1, juntamente com o *golden standard*.

Set	**Resultado ER**
Golden Standard	$\{\{r_1, r_2\}, \{r_3, r_4\}, \{r_5, r_6, r_7, r_8, r_9, r_{10}\}\}$
R_1	$\{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}, \{r_5, r_6, r_7, r_8, r_9, r_{10}\}\}$
R_2	$\{\{r_1, r_2\}, \{r_3, r_4\}, \{r_5, r_6, r_7\}, \{r_8, r_9, r_{10}\}\}$
R_3	$\{\{r_1, r_2, r_3, r_4\}, \{r_5, r_6, r_7, r_8, r_9, r_{10}\}\}$

Tabela 2.1: Resultados ER e *Golden Standard*

Na avaliação dos resultados R_1 e R_2 contra o *golden standard*, usando uma medida de qualidade que avalia o resultado com base no número de pares de registros que casam, R_1 pode ser uma solução melhor, pois encontra 15 pares corretos (ou seja, a combinação de 2 a 2 dos casamentos em $(\{r_5, r_6, r_7, r_8, r_9, r_{10}\})$ enquanto R_2 encontra apenas 8 pares corretos. Por outro lado, caso seja usada uma medida que avalia os resultados com base em entidades corretamente resolvidas no *golden standard*, R_2 pode ser considerado melhor que R_1, pois R_2 contém duas entidades corretamente resolvidas, $\{r_1, r_2\}$ e $\{r_3, r_4\}$, enquanto que R_1 tem apenas uma entidade

correta, ($\{r_5, r_6, r_7, r_8, r_9, r_{10}\}$). Como outro exemplo, na comparação de R_2 e R_3 com o *golden standard*, uma medida poderia ser mais focada na alta precisão e preferir R_2 ao R_3, pois R_2 só possui registros que casam corretamente entre si, enquanto R_3 tem alguns registros que casam e que, de fato, não deveriam casar (por exemplo, r_1 e r_3).

A seguir são apresentadas algumas métricas do estado da arte em qualidade (BELKIN; CROFT, 1992) da resolução de entidades. Nesse tipo de medida, a avaliação dos resultados é feita com base em entidades corretamente resolvidas no *golden standard*, ou seja, existe o interesse em saber se o casamento foi realizado corretamente entre todos os objetos da entidade, de forma com que a entidade formada seja igual à que se encontra no *golden standard*, e não apenas parte dela.

2.3.1 Precisão, Abrangência e Medida-F

As métricas precisão, abrangência e medida-F são medidas frequentemente utilizadas na área de recuperação da informação para avaliar os resultados de buscas, ranqueamentos e resolução de entidades. Para esta avaliação ocorrer é necessário haver uma comparação dos resultados com os dados corretos (*golden standard*), e assim, detectar quão bons são os resultados coletados pela resolução de entidades.

Duas métricas são utilizadas para fazer esta avaliação: precisão e abrangência.

Precisão é a proporção de um conjunto de entidades retornadas que é realmente relevante (BELKIN; CROFT, 1992). Abrangência é a proporção de entidades relevantes que foram retornadas (ZHU, 2004).

A figura 2.4 será utilizada para explicar os conceitos de precisão e abrangência.

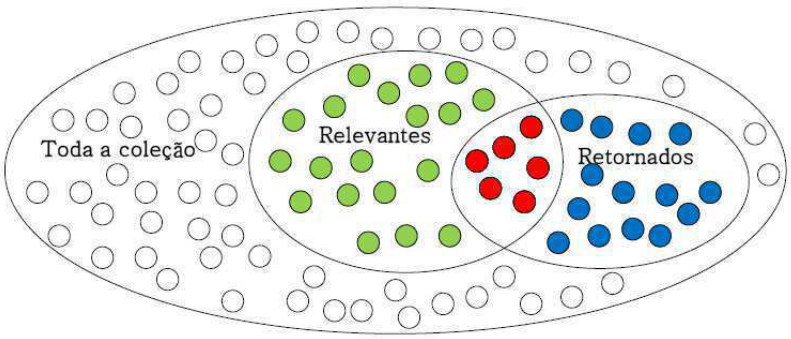


Figura 2.4: Toda a coleção de entidades com destaque para os recuperados e relevantes.

As medidas de precisão e abrangência são definidas por:

$$\text{Precisão} = \frac{\text{Número de entidades relevantes retornadas}}{\text{Número total de entidades retornadas}} \tag{2.1}$$

$$\text{Abrangência} = \frac{\text{Número de entidades relevantes retornadas}}{\text{Número total de entidades relevantes}} \tag{2.2}$$

Ou utilizando as cores dos objetos da figura 2.4 para a definição, pode-se definir precisão e abrangência por:

$$\text{Precisão} = \frac{\text{Vermelhos}}{\text{Azuis + Vermelhos}} \tag{2.3}$$

$$\text{Abrangência} = \frac{\text{Vermelhos}}{\text{Verdes + Vermelhos}} \tag{2.4}$$

Os dois valores devem ser sempre calculados para um determinado conjunto de entidades retornadas e estão compreendidos entre zero e um. Um cenário ideal seria ter sempre uma precisão e abrangência igual a um, o que significa que todos e apenas as entidades relevantes são retornadas. Entretanto, em um sistema real, ao melhorar uma das medidas em geral deteriora a outra. Para melhorar a precisão, deve-se diminuir o número de entidades retornadas, porém, isso diminui a abrangência. Para melhorar a abrangência, deve-se aumentar a quantidade de entidades retornadas. Porém a precisão irá diminuir.

Este compromisso entre as duas métricas torna difícil a qualificação da qualidade de um conjunto de resultados. A medida-F facilita essa análise, pois utiliza apenas um valor numérico entre 0 e 1. A medida-F identifica situações em que os resultados contém informações desnecessárias (baixa precisão), e quando os resultados não contêm informação suficiente (baixa abrangência) (KANDEFER; SHAPIRO, 2009).

A medida-F é uma média harmônica que considera ao mesmo tempo a precisão e abrangência. Esta métrica é dada pela equação 2.5.

$$F = \frac{2 \times \text{ Precisão } \times \text{ Abrangência}}{\text{Precisão } + \text{Abrangência}} \tag{2.5}$$

2.4 Indexação multidimensional

Algoritmos de clusterização que necessitam encontrar objetos ou centroides próximos podem fazer computações desnecessárias ao realizar essa busca. Ingenuamente, faz-se comparações da distância entre todos os objetos para verificar qual é o mais próximo. Dessa forma, a complexidade computacional da busca é muito alta, na ordem de $O(n^2)$, onde n é o número de objetos. Caso um objeto tenha mais de uma dimensão, ou seja, d dimensões, essa complexidade muda para $O((n \times d)^2)$. Para reduzir essa complexidade faz-se uso de estruturas de dados que possam indexar os objetos, ou centroides, se for o caso, com d dimensões. No caso de centroides, essa estrutura de dados precisa ser dinâmica, ou seja, ela tem que permitir atualizações no seu valor com baixa complexidade de computação.

De acordo com (GAEDE; GUNTHER, 1998), no domínio de dados multidimensionais, os métodos de acesso às dimensões podem ser divididos em métodos de acesso a ponto (PAM) e métodos de acesso a espaço (SAM). Métodos de acesso a ponto foram projetados principalmente para realizar pesquisas espaciais em bancos de dados de pontos (ou seja, os bancos que armazenam apenas pontos). Os pontos podem ser incorporados em duas ou mais dimensões, mas eles não têm uma extensão espacial. Métodos de acesso espacial, no entanto, podem gerenciar objetos extensos, como linhas, polígonos, ou mesmo poliedros de dimensão superior.

Segundo (ZÄSCHKE; ZIMMERLI; NORRIE, 2014), os principais representantes da categoria PAM são *KD-trees* e *quadtrees*. *KD-trees* são árvores binárias com k dimensões, onde os nós internos da árvore descrevem hiperplanos alinhados ao eixo que dividem o espaço em duas metades que correspondem ao seus nós filhos. O eixo ao longo do qual a separação é realizada é o mesmo para todos os nós da árvore de um certo nível. Geralmente, as dimensões são alternadas entre os níveis de forma *round-robin*. Uma vez que cada nó representa um ponto inserido, a estrutura da árvore é dependente da ordem de operações de inserção e deleção, além de ser propensa a desbalanceamento. Rebalancear a árvore é complicado, uma vez que as operações de rotação habituais de outras árvores binárias não são aplicáveis para a relação entre dimensão e nível de árvore.

Diferentemente da *KD-tree*, a *quadtree* divide o espaço em todas as dimensões em cada nó. Isto significa que, no caso de 2 dimensões, cada nó interno tem quatro filhos, um para cada quadrante. *Quadtrees* são frequentemente utilizadas em problemas 2D ou 3D, pois elas tendem a exigir uma grande quantidade de memória devido à sua propensão em exigir muitos e grandes nós (ZÄSCHKE; ZIMMERLI; NORRIE, 2014).

Todos esses exemplos de métodos de acesso multidimensional citados foram projetados para lidar com conjuntos de pontos de dados e apoiar pesquisas espaciais sobre eles. Nenhum desses métodos é diretamente aplicável às bases de dados contendo objetos com uma extensão espacial. Os exemplos típicos incluem as bases de dados geográficos, contendo principalmente, polígonos ou dados CAD mecânico, consistindo de poliedros tridimensionais. A *R-tree* é um exemplo de métodos de acesso do tipo SAM (GAEDE; GUNTHER, 1998).

Para os casos onde os pontos a serem indexados forem dinâmicos, ou seja, existem atualizações na árvore frequentemente, é imprescindível utilizar uma estrutura de dados que seja capaz de atualizar a árvore com um mínimo de computação possível. A KD-tree, por exemplo, tem um alto custo de atualização, na ordem de $O(n)$ no pior caso, logo ela não é ideal para indexar esse tipo de dado. No entanto, a PH-tree (ZÄSCHKE; ZIMMERLI; NORRIE, 2014) permite que essas atualizações sejam feitas com baixa complexidade computacional e rebalanceamentos não são necessários. Portanto, é ideal para indexar centroides com d dimensões. Ela é usada nas implementações propostas por esta tese nos Capítulo 5 e 6.

2.4.1 PH-Tree

Conforme (ZÄSCHKE; ZIMMERLI; NORRIE, 2014), a *PH-tree* (*PATRICIA-hypercube-tree*) é uma estrutura de indexação de dados multidimensionais que armazena todos os valores na forma de sequência de bits. A *PH-tree* é baseada na *PATRICIA-tries* combinada com hipercubos para acessar dados eficientemente. Possui eficiência de espaço, obtida pela combinação de prefixo compartilhado com uma implementação de espaço otimizada. O espaço ocupado pela *PH-Tree* é comparável ou menor que o espaço de armazenamento de estruturas de dados sem índices, como, por exemplo, vetores de objetos. A estrutura de armazenamento também serve como um índice multidimensional em todas as dimensões dos dados armazenados. Isto permite um acesso eficiente aos dados armazenados por meio de consultas por pontos ou por área (*range queries*). As *point queries* verificam a existência ou não de um determinado objeto na árvore; e as *range queries* buscam por todos os objetos dispostos dentro de uma área retangular definida pelo ponto do canto inferior esquerdo e pelo ponto do canto superior direito.

Ao contrário da *KD-Tree*, que divide o espaço de cada nó em dois subespaços ao longo de apenas uma das dimensões, a *PH-Tree* é construída em cima de uma quadtree e divide o espaço de cada nó em todas as k dimensões, o que torna as consultas praticamente independentes da ordem em que as dimensões são armazenadas. Isto também tende a reduzir o número de nós da árvore, pois cada nó pode

conter até 2^k filhos em vez de 2. Ao mesmo tempo, a profundidade máxima da árvore é independente de k e é igual ao número de bits no valor mais longo armazenado, ou seja, 8 ao armazenar valores em bytes (ZÄSCHKE; ZIMMERLI; NORRIE, 2014). Além isso, a *PH-Tree* usa navegação hipercubo dentro de um nó para localizar sub-nós e entradas. Uma das vantagens de hipercubos é que, uma vez que os valores de um ponto k-dimensional forem intercalados em um fluxo de bits, eles requerem apenas uma operação de tempo constante para navegar ao sub-nó ou entrada armazenada. Isto é útil para consultas, inserções, deleções e para localizar os pontos da consulta por área. Em *datasets* densos, por exemplo, a *PH-tree* pode se beneficiar do aumento do prefixo compartilhado, economizando espaço, e aumentando a prevalência de hipercubos nos nós, aumentando, assim, o desempenho na navegação e atualização.

Ainda segundo (ZÄSCHKE; ZIMMERLI; NORRIE, 2014), muitas estruturas visam balancear a árvore, a fim de evitar árvores desbalanceadas que são ineficientes em termos de desempenho e requisitos de espaço. A PH-Tree, porém, é desbalanceada, tendo a vantagem de que não há necessidade de rebalanceamento e a árvore é estável no que diz respeito ao inserir ou excluir operações. Isto é útil para a concorrência e no armazenamento em disco, uma vez que ela limita o número de páginas que devem ser reescritas.

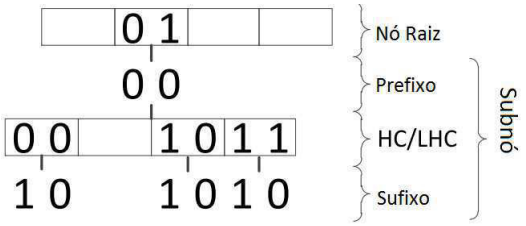


Figura 2.5: Um exemplo de PH-tree 2D com 3 entradas de 4 bits: (0001, 1000), (0011, 1000), (0011, 1010). Fonte: (ZÄSCHKE; ZIMMERLI; NORRIE, 2014)

A instanciação da árvore é feita através do construtor *PHtree(d,w)*, onde d é o número de dimensões e w é o comprimento dos valores em bits. Os comprimentos dos valores típicos representados na árvore são 8, 16, 32 ou 64, pois representam os tipos de dados mais comuns. As entradas armazenadas na árvore são conjuntos de valores e estes valores são convertidos em fluxo de bits para serem armazenados. Um ponto 2D, por exemplo, armazena como entrada dois valores, um valor por dimensão. A Figura 2.5 representa uma *PH-Tree* de duas dimensões que armazena valores de 4 bits, de 0 (0000) a 15 (1111). Nela estão armazenados os valores (0001, 1000), (0011, 1000), (0011, 1010), os quais são armazenados em paralelo, reaproveitando, assim, os prefixos no armazenamento. Neste exemplo, todos os valores da primeira dimensão iniciam com o bit 0, e são representados pela coluna da esquerda no nó raiz, e todos os valores da segunda dimensão iniciam com o bit 1, e são representados pela coluna

da direita. Abaixo do nó raiz está o prefixo do sub-nó compartilhado por todas as entradas. O HC (hipercubo) do sub-nó referencia três sufixos, os quais representam as três entradas na árvore. Para maiores detalhes é necessário consultar o artigo original (ZÄSCHKE; ZIMMERLI; NORRIE, 2014).

2.5 Apache Spark

A busca por desempenho leva os desenvolvedores a paralelizar seus algoritmos. Dentre os diversos *frameworks* de processamento distribuído destacamos o Apache Spark[1]. Spark (ZAHARIA et al., 2012) é uma plataforma de computação em cluster projetada para ser rápida e de uso geral. Do lado da velocidade, o Spark estende o modelo popular MapReduce para apoiar de forma eficiente mais tipos de cálculos, incluindo consultas interativas e processamento de fluxo. A velocidade é importante no processamento de grandes conjuntos de dados, e uma das principais características que o spark oferece para a velocidade é a capacidade de executar cálculos na memória, mas o sistema também é mais eficiente do que o MapReduce para aplicações complexas executadas em disco (KARAU et al., 2015).

Spark utiliza o conceito de RDD. Segundo (KARAU et al., 2015), o RDD (*Resilient Distributed Datasets*) é a principal abstração de programação do Spark, a qual representa uma coleção de itens imutáveis e particionados que são distribuídos em vários nós de computação para serem manipulados em paralelo.

Nesta tese, é importante conhecer os fatores que afetam o desempenho da execução de uma aplicação em Apache Spark. Dentre eles:

- O tamanho do *dataset* de entrada;
- O número de partições ao qual o *dataset* é dividido;
- A quantidade de instâncias;
- A quantidade de núcleos de CPU;
- A quantidade de memória por instância.

O Apache Spark detém resultados intermediários na memória, o que é muito útil quando se precisa processar o mesmo conjunto de dados muitas vezes. Seu projeto teve por objetivo torná-lo um mecanismo de execução que funcionasse tanto na memória como em disco e, por isso, o Spark executa operações em disco quando

[1] http://spark.apache.org/

os dados não cabem mais na memória. Assim, é possível usá-lo para o processamento de conjuntos de dados maiores do que a memória agregada em um cluster (PENCHIKALA, 2015). No entanto, quando isso acontece, as operações de leitura e gravação em disco afetam o desempenho da aplicação.

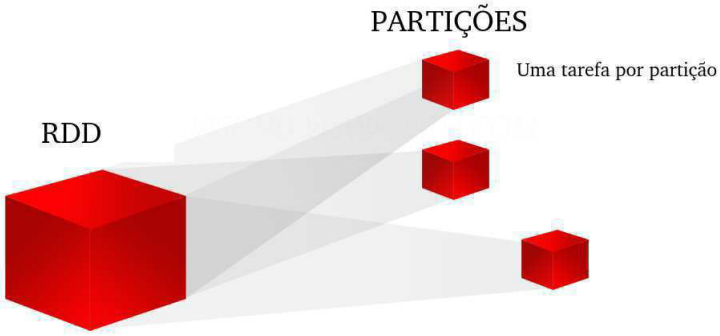


Figura 2.6: Divisão de um RDD em pedaços - Fonte: (VISWANATH, 2015)

*RDD*s são divididos em pequenos pedaços ou partições (ver Figura 2.6), e quando executamos alguma ação, uma tarefa é lançada por partição (e os dados são carregados na memória). Logo, quanto maior o número de partições, maior é o paralelismo.

Quando temos poucas partições há mais dados em cada uma delas, assim, aumentamos o consumo de memória no programa, o que pode fazer uma tarefa levar mais tempo para ser concluída. Por outro lado, quando temos um grande número de partições, a troca de dados pela rede e o uso de disco aumentam (KESTELYN, 2015). Portanto, o número de partições não deve ser muito alto nem baixo. Ele deve ser equilibrado. De acordo com (VELLORE, 2015), não existe um número de partições ideal, ele deve ser encontrado a medida que for executando e obter um bom equilíbrio de simultaneidade e tempo de execução das tarefas. No entanto, para usar todos os nós do *cluster*, devemos ter a quantidade de partições maior ou igual ao número de instâncias (*executors*). No entanto, (KUMAR, 2016) sugere que um cluster com *n* instâncias e *c cores* (núcleos) tenha $n \times c$ partições ou, se não couber em memória, que elas se mantenham entre 50MB e 200MB. Ao termos muitas partições, poderá haver uma sobrecarga excessiva na gestão de pequenas tarefas e prejudicar o seu desempenho, pois o uso da rede e do disco é aumentado.

Um aplicativo spark consiste em um único processo *driver* e um conjunto de processos *executors*, ou instâncias, espalhados entre os nós no cluster. O *driver* é o processo que está no comando do fluxo de controle de alto nível do trabalho que precisa ser feito. As instâncias são responsáveis pela execução deste trabalho, sob a forma de tarefas, bem como por armazenar todos os dados que o utilizador escolheu para armazenar em cache. Uma única instancia tem um número de *slots* para tarefas

em execução, estas são executadas simultaneamente. O esquema de funcionamento do *driver* e das instâncias é ilustrado pela Figura 2.7.

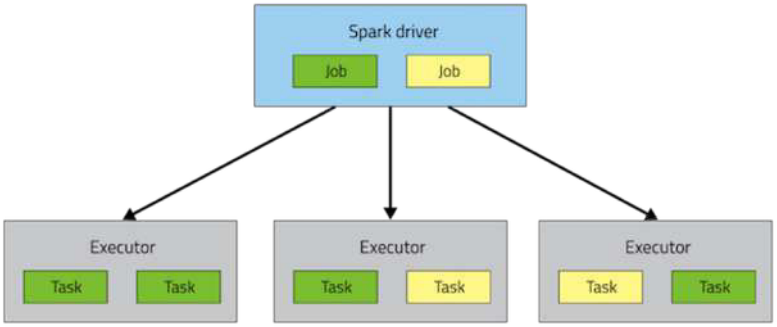


Figura 2.7: *Driver* e *executors* - Fonte: (RYZA, 2015a)

Segundo (RYZA, 2015b), cada instância em uma aplicação *spark* tem o mesmo número de núcleos e tamanho da *heap*. Elas podem ser configuradas através de parâmetros ao executar uma aplicação *spark*. O número de núcleos para cada *executor* pode ser especificado pelo parâmetro *–executor-cores*. O tamanho da *heap* de uma instância pode ser configurado pelo parâmetro *–executor-memory*. O parâmetro *–executor-cores* determina o número de tarefas que uma instância pode executar simultaneamente e *–executor-memory* controla o tamanho da memória que pode ser usada para armazenar em *cache* e o tamanho dos dados no *shuffle* em rede. A quantidade de instâncias para uma aplicação spark pode ser controlada pelo parâmetro *–num-executors*.

Submeter o parâmetro *–executor-cores 3*, por exemplo, ao executar uma aplicação *spark* significa que cada instância pode executar um máximo de três tarefas ao mesmo tempo. Já o parâmetro *–executor-memory 10G*, destinará a cada instância, por padrão, 60% de memória alocada (ou seja, 60% dos 10 gigabytes) a ser usada para armazenamento em cache do RDD, 20% a ser usado para o *shuffle* dos dados e os outros 20% a serem usados para armazenar objetos criados por execução da tarefas. (KUMAR, 2016) recomenda não configurar o *–executor-memory* com mais de 40G de memória, pois pode levar a grandes pausas com a coleta de lixo (*Garbage Collector*). (KUMAR, 2016) também recomenda que o parâmetro *–executor-cores* não seja configurado com mais de 4 núcleos, pois pode impactar negativamente com alta taxa de acesso ao HDFS.

No contexto do nosso trabalho, dado o grande volume de dados a ser processado pelos catálogos, é importante que o spark seja configurado de tal forma a processar o casamento de catálogos. Em particular, na Seção 7.5 do Capítulo 7 vamos utilizar os parâmetros apresentados nesta seção e analisar desempenho da aplicação spark ao variar cada um desses parâmetros.

2.6 Considerações Finais

Neste capítulo apresentamos os principais fundamentos teóricos que foram importantes para o desenvolvimento desta tese. Primeiramente foi abordado o tema de resolução de entidades. Em seguida, apresentamos uma visão geral sobre algoritmos de clusterização, em especial o *K-means*, pois no Capítulo 5 propomos um algoritmo de clusterização baseado no *K-means*.

Também apresentamos medidas de qualidade usadas na literatura para avaliar os resultados produzidos pelas resolução de entidades. Três das medidas apresentadas são utilizadas no Capítulo 7 para medir a qualidade dos resultados obtidos nos experimentos.

E, finalmente, apresentamos uma visão geral sobre indexação multidimensional, em particular, sobre a *PH-Tree*, pois esta foi utilizada nas implementações propostas nos Capítulos 5 e 6.

3 UNIFICAÇÃO DE CATÁLOGOS ASTRONÔMICOS

Este capítulo apresenta os conceitos, problemas e soluções existente na unificação de catálogos astronômicos que serviram de motivação para definirmos o problema alvo desta tese. No entanto, é importante destacar que as propostas da tese não são somente focadas para o problema da astronomia, pois podem ser utilizadas também em outros domínios.

Na Seção 3.3, discorremos sobre como são produzidos os catálogos, suas principais características e atributos relevantes. Dentre os atributos, destacamos as coordenadas equatoriais, ascensão reta e declinação, e o *redshift*.

Na Seção 3.2 fazemos uma introdução ao problema de unificação de catálogos e abordamos a forma tradicional de unificação, o *cross-matching* aos pares de catálogos. Ao final desta seção apontamos os problemas nas soluções existentes que motivaram as investigações desta tese.

E, finalmente, as demais seções tratam das características dos catálogos utilizados nesta tese, bem como dos resultados esperados da unificação e as considerações finais.

3.1 Catálogos Astronômicos

O avanço da tecnologia e da astronomia digital está permitindo a construção de mapas de todo o Universo. O mapeamento, conhecido como catálogo, é um mapa do céu e diz onde olhar para encontrar qualquer um dos astros. Mais precisamente, um catálogo é um *dataset* que contém uma coleção de objetos celestes e suas características, tais como posição, magnitude e cor. Nos catálogos também pode existir uma seção contendo informações mais detalhadas dos objetos: o levantamento espectroscópico. Nela, é possível encontrar informações sobre os *redshifts*, ou o quanto o espectro neste objeto está deslocado, comparado aos objetos que não estão se movendo em relação à Terra, tornando possível calcular a distância entre o corpo celeste e a Terra (SDSS, 2013). Na seção 3.1.1 serão apresentados com mais detalhes alguns dos atributos relevantes dos catálogos.

Segundo (SDSS, 2013), na construção de catálogos são usados telescópios para fotografar o céu sobre toda a área de mapeamento. A partir deste conjunto de observações, os objetos podem ser catalogados em tipos diferentes de objetos celestes, como estrelas, galáxias, quasares, cada um com sua posição no espaço e/ou outras informações importantes. Para obter as posições dos objetos celestes,

supõe-se que eles estão à uma distância comum, todos eles fixados na parte interna da esfera celeste. Portanto, o céu noturno visto por um observador sobre a superfície da Terra é a projeção sobre a esfera celeste de todos os objetos presentes no céu, conforme ilustrado na Figura 3.1.

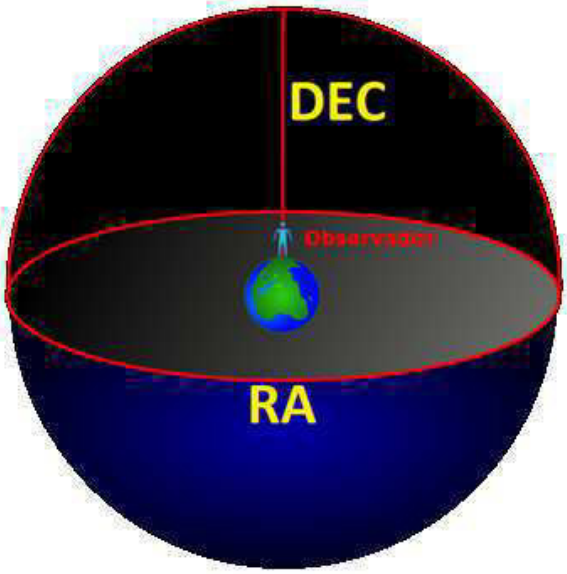


Figura 3.1: A esfera celeste

Ao longo da história da astronomia tem havido um grande número de catálogos de estrelas. Os diferentes catálogos refletem diferentes interesses no céu ao longo da história, bem como mudanças na tecnologia. (THURMOND, 2003) destaca que construir um catálogo de estrelas é uma árdua tarefa, e provavelmente precisa de justificação forte, bem como a mais recente instrumentação.

Diversas fontes bibliográficas, dentre elas (SDSS, 2013) e (THURMOND, 2003), citam que Hiparco de Nicéia provavelmente compilou o primeiro catálogo de estrelas em 127 A.C. No entanto, não é sabido o formato deste, pois não existe cópia escrita dele nos dias de hoje. De qualquer forma, estima-se que poderia conter de 1025 a 1080 estrelas (THURMOND, 2003). (SDSS, 2013) destaca que através do seu catálogo e de observações feitas pelos antigos babilônios, Hiparco descobriu que se olhar para as posições das estrelas a cada ano, elas terão se deslocado um pouco em relação ao ano anterior. Logo, a época em que os catálogos foram gerados é uma informação importante, pois o eixo polar da Terra tem movimento circular, ou seja, ele muda de direção no espaço. Este movimento é chamado de Precessão.

Devido à precessão, as posições dos objetos celestes capturadas a partir da Terra variam com o tempo. Conforme consta em (THURMOND, 2003), a precessão dos polos é cerca de 50,27 segundos de arco por ano ou 1,4 graus por século. A Figura 3.2, apresentada no livro (SANTIAGO; SALVIANO, 2013), ilustra a variação

da posição dos polos celestes devido à precessão. A figura da esquerda, segundo o autor, mostra a situação no presente, em que o polo norte celeste (PNC) coincide aproximadamente com a estrela Polaris. Daqui a milhares de anos, o polo celeste norte coincidirá aproximadamente com a estrela Vega (figura da direita). As siglas PNC, PSC, PNG e PSG correspondem respectivamente à Polo Norte Celeste, Polo Sul Celeste, Polo Norte Geográfico e Polo Sul Geográfico

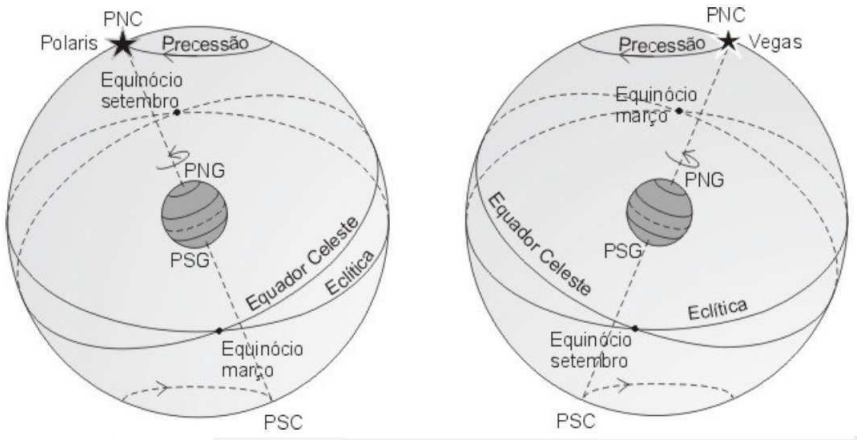


Figura 3.2: Precessão do eixo de rotação terrestre. Fonte: (SANTIAGO; SALVIANO, 2013)

Portanto, quando a posição de uma estrela é dada, também deve ser dado o período de observação. Acontece que, conforme argumentado em (THURMOND, 2003), um catálogo de estrelas pode levar anos ou décadas para ser construído, mas é desejado que a data de observação dos objetos celestes seja a mesma. Para isso, a precessão deve ser calculada e as posições dos astros devem ser ajustadas de forma que todas as medições pareçam ser feitas ao mesmo tempo. O processamento de dados a ser feito é chamado de redução, e o ano para o qual o catálogo é ajustado chama-se época (THURMOND, 2003). A época escolhida para um catálogo de estrelas é arbitrária, mas costuma-se colocá-los em intervalos de cinquenta anos. As épocas mais usadas são primeiro de janeiro de 1950 e de 2000 (ou seja 1950.0 e 2000.0). Detalhes sobre como calcular a precessão podem ser encontrado em (LANG, 1974).

3.1.1 Atributos Relevantes

Dentre os atributos importantes (*RA*, *Dec*, *redshift*, magnitude, cor, etc.) dos objetos presentes nos catálogos, destacamos nesta capítulo suas coordenadas (*RA* e *Dec*) e o *redshift*, embora nas soluções propostas nesta tese utilizamos apenas as coordenadas devido à sua simplicidade, já que elas estão presentes em todos os catálogos. Enquanto que nem todos os catálogos possuem informações sobre o *redshift*.

3.1.1.1 Sistema de Coordenadas Equatoriais

Do mesmo modo que as longitudes e latitudes estabelecem localizações sobre a superfície da Terra, também foi criada uma forma similar para designar localizações no espaço sobre a esfera celeste. De modo semelhante ao sistema geográfico, o sistema de coordenadas equatoriais é o sistema de coordenadas para os objetos celestes cujo plano fundamental, ou seja, o plano em relação ao qual as coordenadas de um astro são definidas, é o equador celeste. Suas coordenadas são chamadas de declinação (δ), ou Dec[1], (coordenada análoga à latitude das coordenadas geográficas) e de ascensão reta (α), ou RA[2], (coordenada análoga à coordenada geográfica longitude) (DAED, 2013). Para definir o sistema, usa-se a esfera celeste, conforme apresentado na Figura 3.3 (DAED, 2013).

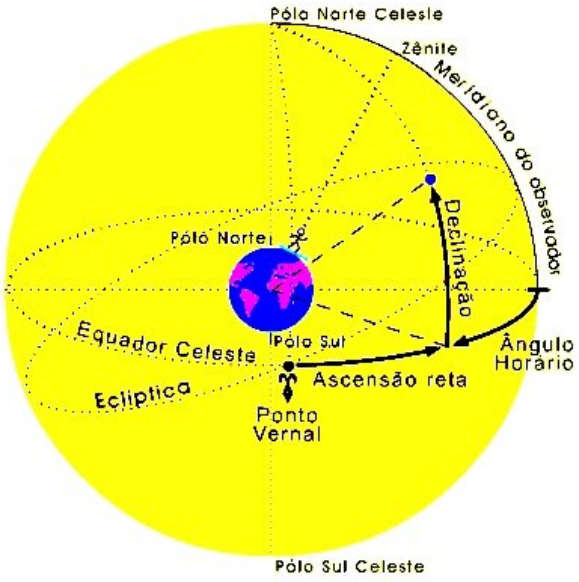


Figura 3.3: O sistema de coordenada equatoriais (DAED, 2013)

Segundo (DAED, 2013), a declinação (δ) de uma estrela é a sua distância angular medida para o norte ou para o sul do equador celeste, análogo, como citado acima, à latitude sobre a Terra medida a partir do equador terrestre. A declinação é medida em graus, minutos e segundos. Logo, δ é a posição angular de um objeto celeste, para o norte (+) ou para o sul (-), medida a partir do equador celeste ao longo do seu círculo horário. Ela pode variar de $+90^\circ$ (no norte) a -90° (no sul), tendo como metade do seu intervalo, o equador celeste com declinação de 0°.

A ascensão reta (α) é obtida pela projeção das linhas de longitude. O ponto sobre a esfera celeste que representa a longitude zero é o equinócio vernal, logo, ele

[1]Sigla originada da palavra *Declination*, em inglês
[2]Sigla originada da palavra *Right Ascension*, em inglês

é o ponto zero da ascensão reta (DAED, 2013).

A ascensão reta (α) de um objeto celeste é o ângulo até o círculo horário do corpo, seu valor cresce sempre no sentido do leste, a partir do ponto vernal, e é expresso em graus (de $0°$ a $360°$). A Figura 3.4[3] ilustra um exemplo de astros e suas coordenadas sobre a esfera celeste.

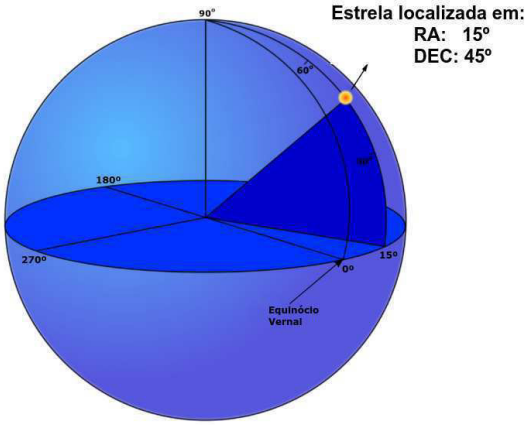


Figura 3.4: Exemplo de estrela e suas coordenadas sobre a esfera celeste

3.2 Unificação de Catálogos

Os diversos levantamentos astronômicos, ou catálogos, se caracterizam por diferentes formatos, esquemas, estruturas de dados, uma vez que são tratados por projetos distintos e independentes. Alguns catálogos cobrem uma determinada faixa do céu, enquanto outros cobrem todo o Universo. Logo, possuem interseções em suas coberturas e é interessante obter uma visão integrada deles, identificando os objetos em comum.

3.2.1 Problemas

Em astronomia, assim como em outras áreas científicas, a correlação e integração de dados observacionais é a chave para a obtenção de novos conhecimentos científicos. Observatórios astronômicos, distribuídos por todo o mundo, produzem dados sobre estudos do céu na forma de catálogos. Os astrônomos que desejam obter informações sobre um determinado objeto astronômico *x* estão muito interessados na seguinte consulta: entre os objetos astronômicos observados em todos os catálogos, encontrar aqueles que, provavelmente, são o mesmo objeto *x*.

[3]Figura baseada no exemplo mostrado em http://lcogt.net/spacebook/equatorial-coordinate-system/

Para resolver essa consulta, é preciso levar em conta as característica do objeto. A principal característica que identifica um mesmo astro em diferentes catálogos é sua posição no espaço. No entanto, encontrar essas correspondências entre fontes de catálogos distintos não é simples, pois durante a captura das posições dos objetos são produzido minúsculos graus de incerteza em suas posições, devido aos erros de astrometria. Logo, a mesma fonte em catálogos diferentes, possui pequenas diferenças em suas coordenadas, gerando dúvida ou ambiguidade na busca do casamento verdadeiro. O processo de resolução de entidades envolvendo objetos astronômicos é conhecido na literatura como *cross-matching*.

3.2.2 Soluções Existentes

A maioria dos algoritmos de *cross-matching* realiza o casamento de dois objetos utilizando somente suas posições, através das coordenadas *RA* e *Dec*. Por exemplo, supõe-se que exista um catálogo *A* e deseja-se realizar seu casamento com um catálogo *B*. Para cada objeto do catálogo *A*, procuram-se por objetos próximos em um raio ε. Se existir algum objeto pertencente ao catálogo *B* neste raio, este objeto é candidato ao casamento.

A ilustração dessa generalização do casamento binário está presente na Figura 3.5.

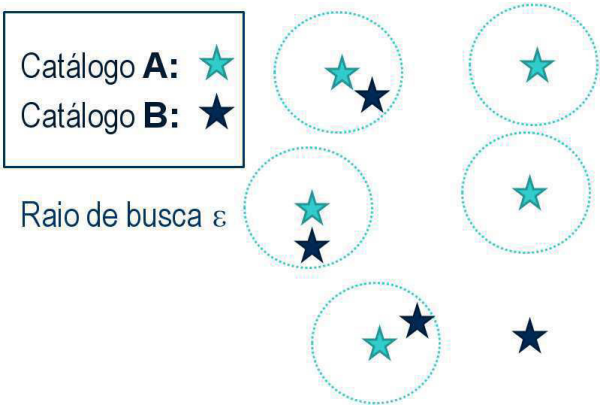


Figura 3.5: Ideia geral de como funciona um cross-matching entre dois catálogos (*A* e *B*)

Poucos trabalhos na literatura abordaram casamentos no domínio de pesquisa da astronomia. Particularmente, em (FREIRE et al., 2013) alguns algoritmos de *cross-matching* de catálogos astronômicos são apresentados. Dentre eles, o algoritmo *Q3C Join* (KOPOSOV; BARTUNOV, 2006), que realiza *cross-matching* de dois catálogos através da função *q3c_join(catalogo1.ra, catalogo1.dec, catalogo2.ra, catalogo2.dec, raioEpsilon)*, cujos parâmetros são autoexplicativos.

No Capítulo 4 discutimos em detalhes sobre os diversos algoritmos de cross-matching presentes na literatura.

É importante destacar que esta tese, nem os trabalhos relacionados, tratam do problema de lentes gravitacionais, onde a força gravitacional exercida por um corpo de grande massa, como galáxias e buracos negros, distorcem o espaço-tempo fazendo com que a luz e outras partículas realizem um movimento curvilíneo na sua proximidade, gerando assim, duas posições para o mesmo objeto, sendo que nenhuma delas representa a posição correta (BURCHAT, 2008). A Figura 3.6 ilustra o caso, onde as linhas brancas representam o caminho da luz de uma objeto distante até um observador na Terra, e as linhas laranjas representam as posições aparentes do objeto por um observador.

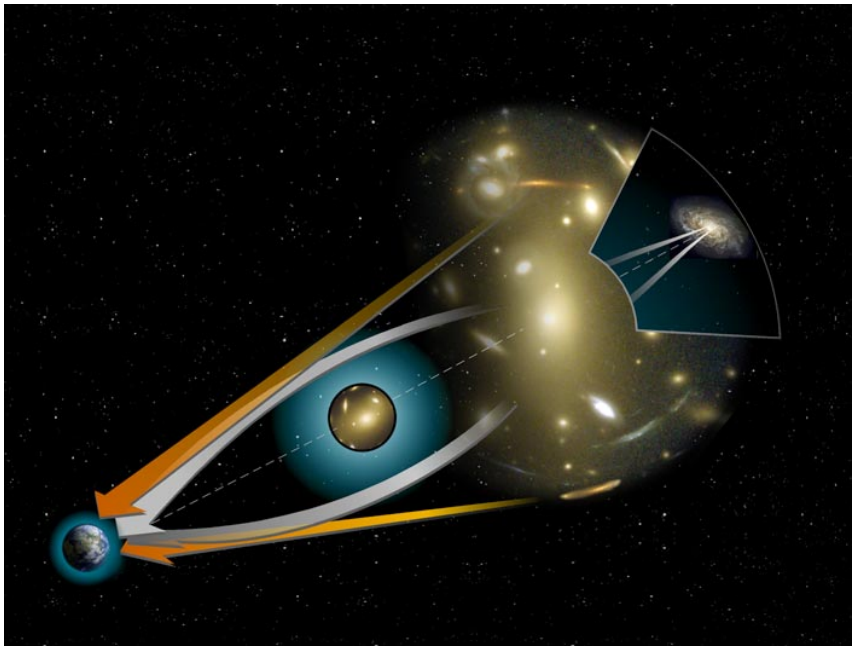


Figura 3.6: Problema de Lentes Gravitacionais. Fonte: (FRUCHTER et al., 2000)

3.2.3 Problemas nas Soluções Existentes

As abordagens de casamentos tradicionais, como o *Q3C*, realizam casamentos somente entre um par de catálogos por vez. Este fato gera dúvidas sobre a qualidade do resultado do casamento entre n catálogos ao usar essa metodologia.

Para adicionar um terceiro catálogo ao casamento do *Q3C*, previamente temos que realizar o casamento do primeiro par de catálogos. O resultado do primeiro casamento possui os objetos distantes até ε entre si (os demais objetos com distância maior que ε não são listados). Ele deve ser usado como entrada para o próximo casamento juntamente com o terceiro catálogo.

A função *q3c_join* aceita somente parâmetros *ra* e *dec* de dois catálogos por vez. Como o resultado do primeiro casamento possui as coordenadas do objeto do primeiro catálogo e as coordenadas do objeto do segundo catálogo, devemos escolher os atributos de um destes dois catálogos para serem usados como entrada na função *Q3C Join* e encontrar os objetos próximos de um terceiro catálogo, por exemplo *q3c_join(catalogo1.ra, catalogo1.dec, catalogo3.ra, catalogo3.dec, raioEpsilon)* ou *q3c_join(catalogo2.ra, catalogo2.dec, catalogo3.ra, catalogo3.dec, raioEpsilon)*. Assim, o resultado do casamento entre 3 catálogos sempre depende:

- da ordem dos casamentos entre pares de catálogos;
- e da escolha de qual dos dois catálogos iniciais é tomado por base para realizar o casamento com o terceiro.

Um exemplo dessa dependência está ilustrado na Figura 3.7 ao considerar uma região do céu contendo objetos de três catálogos a serem casados pelo Q3C.

Na Figura 3.7, o primeiro retângulo corresponde à uma região do céu contendo objetos de três catálogos distintos. Cada cor representa um catálogo e os números representam os *ids* dos objetos. Os demais retângulos (de **a** à **m**) apresentam diferentes resultados do casamento do Q3C aplicado aos mesmos objetos do primeiro retângulo. A variação dos resultados se dá pela ordem em que o casamento foi feito. O par de catálogos entre parênteses na legenda de cada item indica o primeiro casamento binário, cuja saída é usada no casamento com um terceiro catálogo. No entanto, somente os atributos do catálogo sublinhado são considerados nesse novo casamento, pois a função *q3c_join* só aceita parâmetros de um dos dois catálogos. As elipses representam os resultados finais dos casamentos, ou seja, objetos reais. Para o mesmo conjunto de objetos, a ambiguidade está presente no casamento, pois existem seis resultados distintos na figura, além de alguns deles contendo objetos do mesmo catálogo.

A dificuldade de realizar o casamento entre *n* catálogos, bem como a diversidade dos resultados encontrados nos métodos de casamento tradicionais despertaram a motivação deste trabalho. A motivação é **a necessidade de uma solução de resolução de ambiguidade neste contexto, em especial a resolução de casamentos entre *n* catálogos**. A questão da ambiguidade dos casamentos é um dos problemas em aberto na área e precisa ser explorada.

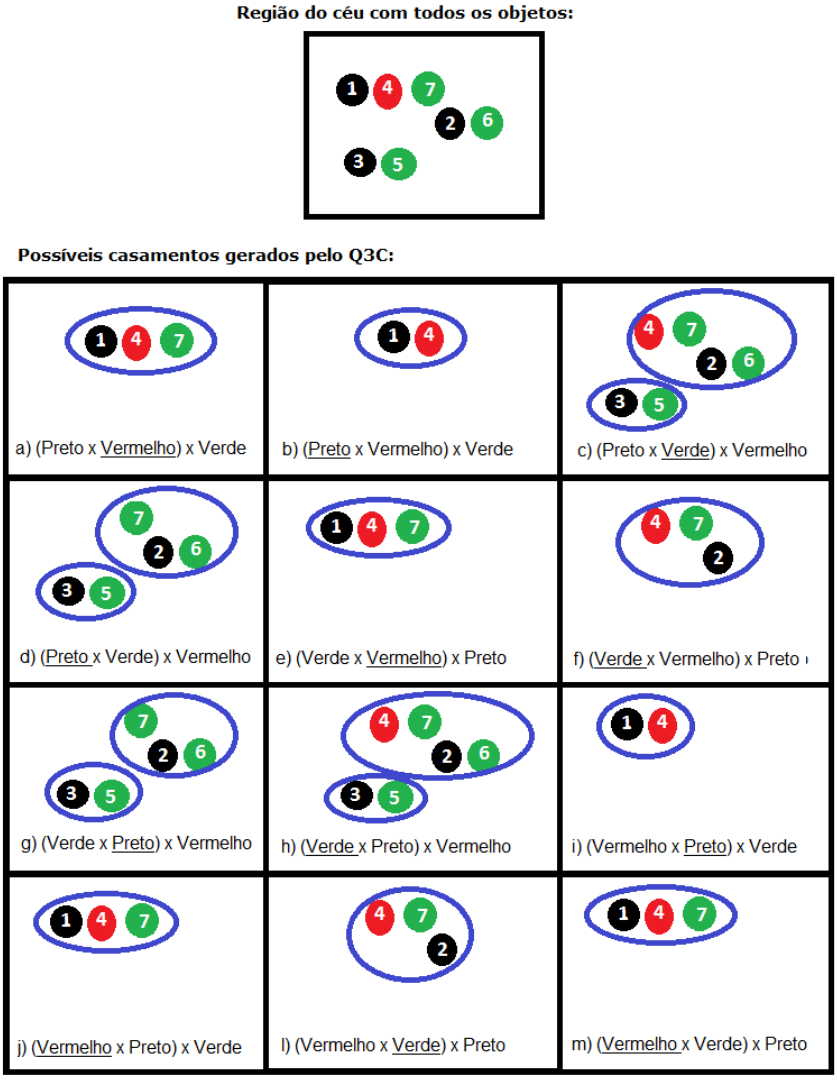


Figura 3.7: Exemplo de casamento de 3 catálogos no Q3C

3.3 Os Catálogos a Serem Usados Nesta Tese

Dentre os catálogos a serem utilizados nesta tese, podemos destacar o 2MASS (SKRUTSKIE; CUTRI; AL., 2006), WISE (WRIGHT et al., 2010) e UCAC (FINCH; Zacharias; Wycoff, 2010).

3.3.1 2MASS

2MASS[4] significa *Two Micron All Sky Survey*. Este catálogo foi produzido pela Universidade de Massachusetts e pelo Centro de Análise e Processamento de Infravermelhos da NASA. Ele contém 470.992.970 objetos e cobre 99,998% do céu observado, representado pela região colorida da Figura 3.8. Foram usados dois teles-

[4]Disponível em http://www.ipac.caltech.edu/2mass/releases/allsky/

cópios 1.3m para sua produção, um em *Mount Hopkins*, no Arizona, e um em Cerro Tololo, no Chile. Os telescópios eram altamente automatizados e varreram o céu em três bandas de infravermelho (SKRUTSKIE; CUTRI; AL., 2006).

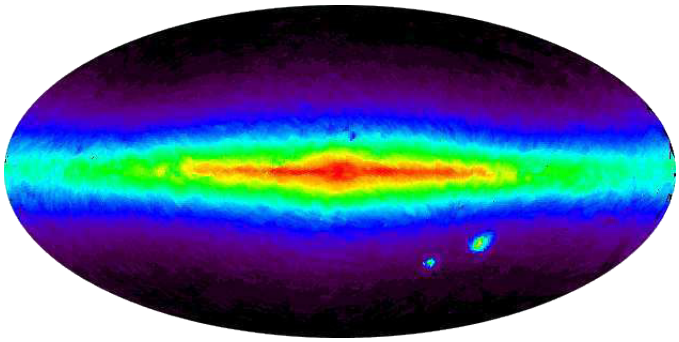


Figura 3.8: Área coberta pelo catálogo 2MASS. Fonte: (CDS, 2016)

3.3.2 WISE

WISE[5] cataloga todo o céu em luz infravermelha e possui mais de 300 milhões de objetos. O catálogo foi lançado em 2010 e apresentado em (WRIGHT et al., 2010). A Figura 3.9 mostra a área coberta pelo catálogo.

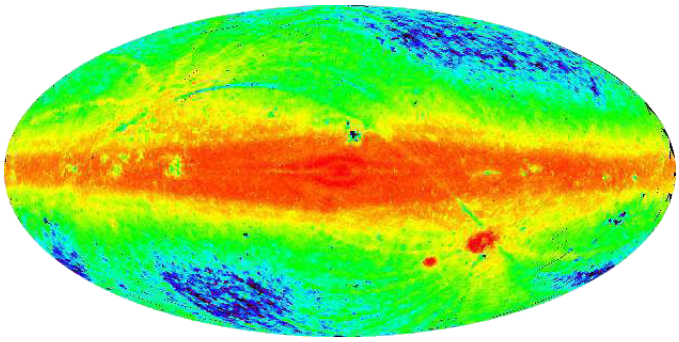


Figura 3.9: Área coberta pelo catálogo WISE. Fonte: (CDS, 2016)

3.3.3 UCAC3

UCAC3[6] é um catálogo que abrange todo o céu de estrelas, cobrindo principalmente a faixa de magnitude 8 a 16. A área colorida da Figura 3.10 representa a região coberta. O catálogo é complementado por dados fotométricos dos catálogos *2MASS* e *SuperCosmos*, além dos movimentos próprios das estrelas brilhantes baseadas em cerca de 140 catálogos, incluindo os catálogos *Hipparcos* e *Tycho* (FINCH;

[5]Disponível em http://wise2.ipac.caltech.edu/docs/release/allsky/
[6]http://www.usno.navy.mil/USNO/astrometry/optical-IR-prod/ucac

Zacharias; Wycoff, 2010). O catálogo UCAC3 foi lançado em 10 de agosto de 2009 na Assembleia Geral da União Astronômica Internacional no Rio de Janeiro e possui informações sobre 113.780.093 objetos celestes.

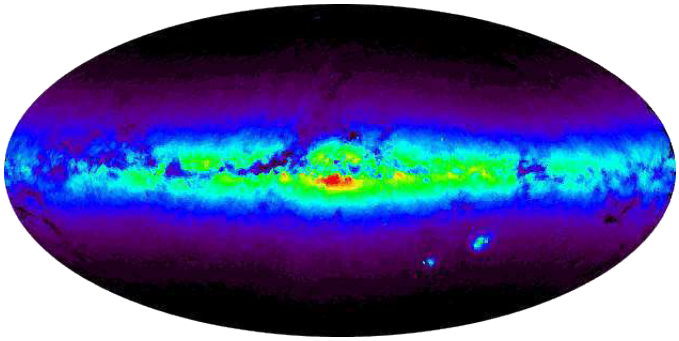


Figura 3.10: Região do universo coberta pelo UCAC3. Fonte: (CDS, 2016)

3.4 Resultados Esperados da Unificação

Na seção 3.2.3 vimos o exemplo da Figura 3.7 em que um casamento de três catálogos, aplicado ao pares, produz diferentes resultados. Esses resultados dependem da ordem com que são realizados os casamentos. A diversidade de resultados no casamento de *n* catálogos induz dúvidas quanto à confiabilidade do casamento. Portanto, esperamos que a unificação dos catálogos:

- realize os casamentos independente da quantidade de catálogos dados na entrada;
- produza um resultado correto e que independa da ordem dos casamentos;
- tenha a característica de transitividade no casamento. Problemas de transitividade ocorrem, por exemplo, quando dados três objetos O_1, O_2 e O_3 de diferentes catálogos, O_2 casa com O_1 e O_3, mas O_1 não casa com O_3. Portanto, $O_1 = O_2$, $O_2 = O_3$, mas $O_1 \neq O_3$. Neste cenário, algoritmos que considerem transitividade do relacionamento proporiam a relação: $O_1 = O_2 = O_3$.

3.5 Considerações Finais

A evolução da ciência causou avanços na astronomia. Antigamente as observações do céu eram feitas sem a ajuda de nenhum instrumento. O surgimento dos telescópios fez com que essas observações passassem a ser mais precisas. E

hoje, os dados capturados através deles são digitais e a quantidade de informação é imensa.

Catálogos com um grande volume e variedade de informações são produzidos a todo o momento. Eles cobrem parte do céu ou toda a sua área. Portanto, diversos catálogos possuem áreas e objetos celestes em comum. Os cientistas estão interessados em identificar objetos semelhantes presentes em diferentes catálogos. Esse processo é conhecido como casamento.

Neste capítulo, apresentamos os problemas das abordagens tradicionais de casamento, dentre eles o fato do casamento entre n catálogos acontecer aos pares e produzir diferentes saídas, já que estas dependem da ordem em que os catálogos são casados. Estes problemas motivaram a investigação desta tese abordada nos próximos capítulos.

4 TRABALHOS RELACIONADOS

Este capítulo apresenta os principais trabalhos relacionados com esta tese. Os primeiros três conjuntos de trabalhos apresentados nas Seções 4.1 e 4.2 estão relacionados com as contribuições apresentadas no Capítulo 5.

Nas Seções 4.1.1 a 4.1.4 apresentamos as propostas de algoritmos de clusterização que são usados no problema de resolução de entidade.

Em seguida, na Seção 4.2.1 comentamos sobre os trabalhos que contribuem com soluções que apoiam o casamento por localização em astronomia através da representação do céu fazendo uso de estruturas de dados, a fim de facilitar a localização dos objetos e os seus vizinhos no espaço. Discutimos as características e apresentamos as vantagens e desvantagens de cada uma delas.

A Seção 4.2 apresenta diversos trabalhos que contêm algoritmos de *cross-matching* estudados durante o desenvolvimento desta tese. Classificamos os trabalhos em duas vertentes: aqueles que usam apenas a distância como critério para realizar o casamento; e os que usam outros atributos no casamento.

O quarto grupo de trabalhos, apresentados na Seção 4.3, são importantes para que se entenda o escopo e as contribuições apresentadas no Capítulo 6. A Seção 4.3 trata de trabalhos em que a ideia central consiste em criar algoritmos de clusterização que funcionam de forma distribuída. Portanto, eles abordam o tema de particionamento dos dados de entrada, bem como do tratamento das fronteiras, com o objetivo de melhorar a eficiência da execução e do balanceamento de cada nó. Esses algoritmos estão interessados em unir os objetos próximos para identificar os grupos com formas arbitrárias, no entanto, os algoritmos propostos nesta tese estão interessados em formar agrupamentos de objetos próximos, mas que pertençam a diferentes catálogos.

4.1 Algoritmos de clusterização para o problema de resolução de entidade

A solução para o problema de resolução de entidades atacado nesta tese é feita através do uso de técnicas de clusterização. Logo, é importante conhecer o estado da arte em algoritmos de clusterização para este problema. Um *survey* que faz esse estudo pode ser encontrado em (HASSANZADEH et al., 2009).

Uma das etapas do processo de resolução de entidade em dados incertos é o cálculo de similaridade. (HASSANZADEH et al., 2009) apresenta o resultado do cálculo de similaridade como um grafo e aplica algoritmos de clusterização sobre ele

para resolver as entidades.

Segundo (HASSANZADEH et al., 2009), para representar registros similares como um grafo, considera-se uma relação como um grafo $G(V,E)$, onde cada nó $u \in V$ representa um registro na relação e cada aresta $(u,v) \in E$ conecta dois nós u e v somente se eles são similares, ou seja, se seu valor de similaridade baseado em alguma função de similaridade $sim()$ é acima de um limite especificado.

O grafo de similaridade frequentemente é ponderado, ou seja, cada aresta (u,v) tem um peso $w(u,v)$, que é igual ao valor de similaridade entre os registros correspondentes ao nós u e v. Mas um ponto chave é que as técnicas de similaridade são extremamente eficientes em encontrar um conjunto pequeno e preciso de itens semelhantes. Esta característica permite a utilização eficaz das técnicas de clusterização na saída do cálculo de similaridade (HASSANZADEH et al., 2009).

Dado o resultado de uma similaridade, um algoritmo de clusterização pode ser usado para agrupar nós no grafo de similaridade de registros em clusters contendo potenciais casamentos. Assim, os algoritmos de clusterização devem ser algoritmos sem restrições, ou seja, algoritmos que não necessitam ter como entrada o número de clusters ou outros parâmetros específicos do domínio (HASSANZADEH et al., 2009).

Algoritmos de clusterização sem restrições visam criar clusters contendo registros semelhantes $C = \{c_1, ..., c_k\}$ em que o valor de k é desconhecido. A clusterização pode ser exclusiva (disjunta), na qual a relação é particionada e não há sobreposição de nós entre os clusters. Alternativamente, a clusterização não exclusiva permite que os nós pertençam a mais de um cluster, embora seja desejável que esta sobreposição seja pequena (HASSANZADEH et al., 2009). Todos os algoritmos podem ser eficientemente implementados por uma única varredura na lista de pares semelhantes devolvidos pela etapa de cálculo de similaridade, embora alguns exijam que a lista seja ordenada por valores de similaridade. O grafo G é usado para ilustrar as técnicas.

4.1.1 Algoritmo 1: Particionamento

Segundo (HASSANZADEH et al., 2009), no algoritmo *Particionamento* clusteriza-se o grafo de registros para encontrar componentes conexas no grafo e colocar os registros de cada componente em um cluster diferente. Primeiramente, deve-se associar cada nó a um cluster diferente e depois varrer a lista de pares similares. Se dois nós conectados não estão no mesmo cluster, seus clusters são mesclados. A Figura 4.1(a) apresenta o resultado deste algoritmo em um grafo exemplo. Como mostra a Figura 4.1(a), este algoritmo pode inserir muitos registros que não são semelhan-

tes no mesmo cluster (HASSANZADEH; MILLER, 2009). O particionamento é um algoritmo muito usado nos primeiros trabalhos de resolução entidade (ELMAGARMID; MEMBER, 2007) (HERNáNDEZ; STOLFO, 1998).

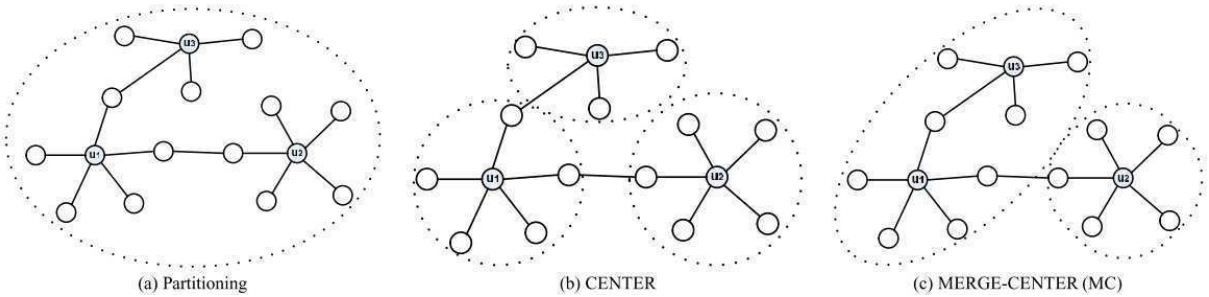


Figura 4.1: Ilustração de algoritmos de clusterização disjuntos. (a) Partitioning, (b) CENTER, (c) MERGE-CENTER (MC) (HASSANZADEH et al., 2009)

4.1.2 Algoritmo 2: CENTER

Conforme (HASSANZADEH et al., 2009), esse algoritmo, chamado de CENTER (HAVELIWALA; GIONIS; INDYK, 2000), executa a clusterização através do particionamento do grafo de registros de forma que cada cluster tenha um centro e todos os registros do cluster sejam semelhantes ao centro. Isso pode ser realizado por um único percorrimento da lista ordenada de pares semelhantes. A primeira vez que um nó u aparece na varredura, ele é atribuído como o centro do cluster. Todos os nós subsequentes v que aparecem em um par (u,v) são atribuídos ao cluster de u e não são considerados novamente. A Figura 4.1(b) mostra como este algoritmo clusteriza um exemplo de grafo de registros, no qual o nó u_1 é o primeiro nó na lista ordenada de registros semelhantes, o nó u_2 aparece logo depois dos nós similares a u_1, e o nó u_3 aparece depois de todos os nós similares a u_2. Este algoritmo pode resultar em mais clusters que o algoritmo Particionamento, uma vez que insere em um cluster somente os registros que são similares ao registro que é o centro do cluster (HASSANZADEH et al., 2009).

4.1.3 Algoritmo 3: MERGE-CENTER

Segundo (HASSANZADEH et al., 2009), MERGE-CENTER, ou MC, é semelhante ao CENTER, no entanto, mescla dois clusters c_i e c_j sempre que um registro similar ao nó central de c_j já está no cluster c_i. Deve-se observar que quando dois clusters se fundem, não é escolhido um único nó central neste algoritmo, para cada cluster pode haver vários nós centrais. Tal como acontece com CENTER, uma única varredura da lista de registros semelhantes é feita, mas mantendo o controle dos registros

que já estão em um cluster. A primeira vez que um nó u aparece na varredura, ele é designado como o centro do cluster. Todos os nós v subsequentes que aparecem em um par (u,v) e não estão presentes em nenhum cluster, são associados ao cluster de u, e não são selecionados como o centro de nenhum outro cluster. Sempre que um par (u,v') é encontrado, tal que v' já está em outro cluster, todos os nós do cluster de u (registros semelhantes a u) são mesclados com o cluster de v'. A Figura 4.1(c) mostra os clusters criados por este algoritmo assumindo novamente que os nós u_1, u_2 e u_3 são os três primeiros nós na lista ordenada de registros similares que são selecionados como o centro do cluster. Como mostrado na Figura 4.1(c), este algoritmo cria menos clusters para o grafo exemplo do que o algoritmo CENTER, mas cria mais clusters do que o algoritmo de particionamento (HASSANZADEH; MILLER, 2009).

4.1.4 Algoritmo 4: ND (clusterização não disjunta)

(HASSANZADEH et al., 2009) ainda apresenta o algoritmo ND. Neste algoritmo, não se deseja que $c_i \cap c_j = \emptyset$ para todo $i, j \in 1...k$. A ideia é ter um núcleo para cada cluster que contém as registros que são altamente similares e os registros marginais para cada cluster que são relativamente menos semelhantes. Ou seja, os registros que casam com o centro do cluster são inseridas em seu núcleo e os registros que potencialmente casam com o centro ficam nos registros marginais do cluster. Cada registro aparece no núcleo de apenas um cluster, mas pode aparecer nos registros marginais de mais de um cluster (HASSANZADEH; MILLER, 2009).

No algoritmo ND, cria-se um conjunto de núcleos de clusters (similar ao MERGE-CENTER), e então um conjunto de registros menos semelhantes ao centro do cluster é adicionado a cada cluster. O algoritmo executa como segue. Supõe-se que exista a lista de registros com valores de similaridade acima de um limite θ_2 juntamente com seu valor de similaridade. O algoritmo inicia por uma varredura na lista. A primeira vez que um nó u aparece na varredura, ele é associado como o centro do núcleo do cluster. Todos os nós v subsequentes que aparecem em um par (u,v), tem $sim(u,v) \geqslant \theta_1$ e não estão presentes no núcleo de nenhum outro cluster, são associados ao núcleo do cluster de u e não são selecionados como o centro de nenhum outro cluster. Outros pares (u,v) que tem $sim(u,v) \leqslant \theta_1$ (mas tem $sim(u,v) \geqslant \theta_2$) são adicionados como membros marginais do cluster. Sempre que um par (u,v') com $sim(u,v') \geqslant \theta_1$ é encontrado, tal que v' já está no núcleo de outro cluster, todos os nós do cluster de u são mescladas com o cluster de v'.

4.2 Cross-matching - Casamento de objetos celestes

Esta tese tem como principal contribuição um algoritmo para realizar casamentos de objetos espaciais. Como visto no Capítulo 3, os problemas existentes nos casamentos de objetos celestes nos motivaram a desenvolver nosso trabalho. Portanto, é importante descrevermos o estado da arte dos algoritmos de *cross-matching* existentes na literatura.

A Seção 3.2.2 do Capítulo 3 introduz o casamento de catálogos astronômicos e mostra que a maioria dos algoritmos de *cross-matching* realiza o casamento de dois objetos utilizando somente suas posições, através das coordenadas *RA* e *DEC*. Por exemplo, supõe-se que exista um catálogo *A* e deseja-se realizar seu casamento com um catálogo *B*. Para cada objeto do catálogo *A*, procura-se por objetos próximos em um raio ε. Se existir algum objeto pertencente ao catálogo *B* neste raio, este objeto é candidato ao casamento. A ilustração desta generalização do casamento binário está presente na Figura 4.2.

Figura 4.2: Ideia geral de como funciona um cross-matching entre dois catálogos (*A* e *B*)

No decorrer desta seção apresentaremos e discutiremos sobre diversos algoritmos de cross-matching presentes na literatura.

4.2.1 Indexação espacial como suporte ao casamento

Dividir o céu é uma prática antiga e útil tanto para localizar quanto para identificar objetos no espaço. Em (ORTIZ, 2003), os autores afirmam que o problema hoje em dia imposto pelo grande volume de dados é o de ter a capacidade de localizar e comparar objetos encontrados na mesma região do céu a partir de dois (ou mais) catálogos rapidamente.

Existem algumas soluções atuais que apoiam o casamento por localização. A principal estratégia é a de representar o céu fazendo uso de estruturas de dados, a fim de facilitar a localização das estrelas e os seus vizinhos no espaço. Em bancos de dados, essas estruturas de dados são conhecidas como estruturas de indexação. Indexar o céu é particionar a esfera celeste, a fim de alcançar um melhor desempenho em consultas que envolvem encontrar vizinhos próximos. Essa estratégia também é conhecida como tesselação.

A literatura aponta quatro estruturas importantes de indexação:

- HTM (*Hierarchical Triangular Mesh*)(KUNSZT; Szalay; Thakar, 2001)
- HEALPix (*Hierarchical Equal Area isoLatitude Pixelisation*) (GORSKI,)
- Q3C (*Quad Tree Cube*) (KOPOSOV; BARTUNOV, 2006)
- ZONES (GRAY; SZALAY; AL., 2004)

O HTM foi proposto em (KUNSZT; Szalay; Thakar, 2001). Nessa estrutura de indexação, a esfera celeste é inicialmente representada por um octaedro com oito triângulos, quatro para representar o hemisfério norte e quatro para o sul, conforme Figura 4.3. Cada triângulo do octaedro é dividido em quatro e o processo é repetido de forma recursiva a uma profundidade pré-determinada. O octaedro inicial é considerado o nível 0 e a cada subdivisão um novo nível de profundidade. Os triângulos são numerados com o objetivo de preservar a proximidade espacial.

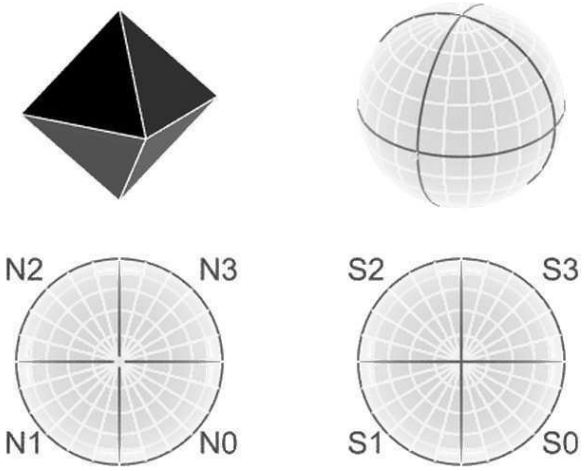


Figura 4.3: O octaedro e os 3 pontos de vista de sua projeção na esfera (FEKETE, 2007).

Em (SZALAY et al., 2005), os autores argumentam que a implementação do HTM tem um desempenho aceitável mesmo em bancos de dados astronômicos,

contendo várias centenas de milhões de objetos. No entanto, em (O'MULLANE et al., 2001), os autores criticam o HTM, pois ele faz muitos cálculos complexos devido à área e a forma de cada pixel ser diferente, bem como às diferentes direções dos triângulos. O HTM também recebe críticas em (KOPOSOV; BARTUNOV, 2006), pois o mesmo não possui versão de implementação para sistema de gerenciamento de banco de dados de código aberto. A implementação fornecida é para uso no sistema proprietário Microsoft SQL Server.

No HEALPix (**H**ierarchical **E**qual **A**rea iso**L**atitude **Pix**elization) [1], apresentado em (GÓRSKI; HIVON; WANDELT, 1998), a superfície esférica é subdividida em quadriláteros curvilíneos, de tal modo que cada pixel possua a mesma área. A partição de resolução mais baixa é formada por 12 pixeis. A cada subdivisão, todos os pixeis são subdivididos em quatro novos pixeis de áreas iguais. A Figura 4.4 ilustra (do canto superior esquerdo para o inferior esquerdo), o aumento de resolução por três etapas a partir do nível base (ou seja, a esfera é dividida, respectivamente, em 12, 48, 192 e 768) (GORSKI; ARBALLO,). Logo, o céu é dividido em 12×4^n pixeis no nível *n*. E cada objeto celeste, com sua informação de posição (RA e DEC) pertencerá a um desses pixeis. Em outras palavras, o espaço bidimensional é mapeado para um espaço de linha (1 dimensão). Finalmente, uma função de índice de árvore B, suportada por todos os SGBDs, pode ser usada para indexar estes dados (ZHAO et al., 2009).

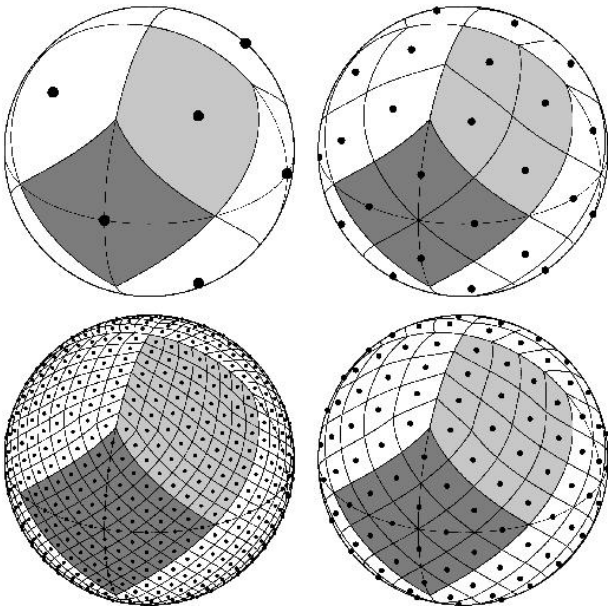


Figura 4.4: Divisão da esfera celeste no HEALPix (GORSKI; ARBALLO,).

Segundo (DU et al., 2014), o princípio do HEALPix é o mesmo do HTM, mas a diferença entre as duas funções de indexação espacial é que o HEALPix particiona o céu em quadriláteros de áreas iguais, e tem essa característica como vantagem

[1] Disponível em http://healpix.jpl.nasa.gov/

sobre o HTM. De acordo com (O'MULLANE et al., 2001), outra vantagem do HEALPix é que a numeração aninhada dos pixeis de nível abaixo facilita a busca pelos vizinhos. No entanto, o mesmo critica a variação na forma dos pixeis, mas ressalta que não é um problema quando se usa uma grande resolução. Além da forma, a orientação dos pixeis vizinhos pode ser diferente. Dentre outras anomalias apontadas por ele, destaca-se também que a maioria dos pixeis tem oito vizinhos, mas o pixel norte e sul de cada face equatorial tem somente sete vizinhos, gerando um impacto na busca pela vizinhança.

A ideia do Q3C é semelhante a outros esquemas de indexação do céu. Neste caso, a esfera celeste é representada por um cubo inscrito na esfera. Em cada face do cubo é construída uma *quadtree*. Na qual cada região que for um quadrante "ocupado" é subdividida recursivamente em quatro sub-quadrantes menores, até que os subquadrantes "ocupados" tenham no máximo um objeto celeste. Essa subdivisão pode ser exemplificada através da Figura 4.5. Ela mostra os passos necessários para a criação de uma *quadtree* através da subdivisão recursiva do espaço, no qual os locais em amarelo correspondem às áreas "ocupadas", ou seja, que contém objetos e que precisam ser subdivididas. A numeração dos quadrantes é feita em sentido horário partindo do quadrante superior esquerdo. A divisão final da esfera celeste em diferentes profundidades é mostrada na Figura 4.6.

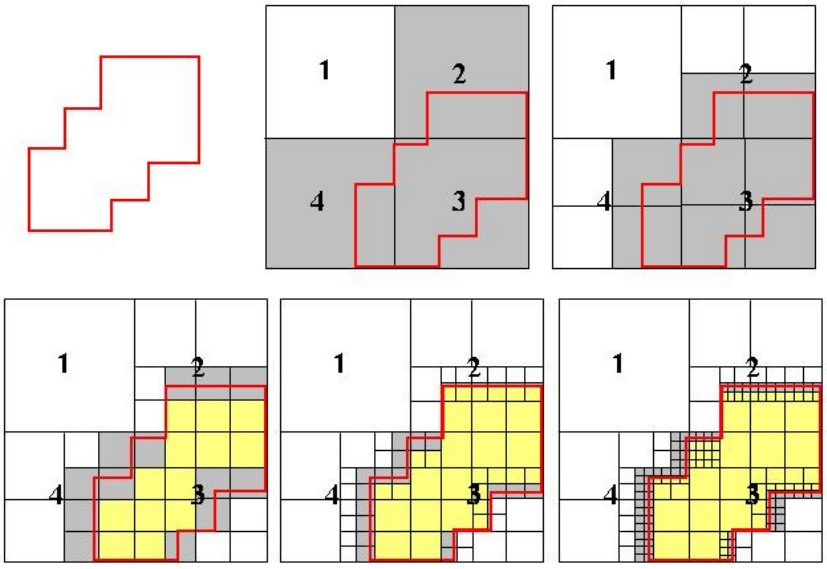


Figura 4.5: Passos para obtenção de uma Quadtree

De acordo com (KOPOSOV; BARTUNOV, 2006), os cálculos do Q3C são muito mais simples que no HTM e no HEALPix e, portanto, eles não limitam o desempenho das consultas. Ainda de acordo com (KOPOSOV; BARTUNOV, 2006), devido ao uso das quadtree, os cálculos são mais rápidos que no HTM no caso de alta pro-

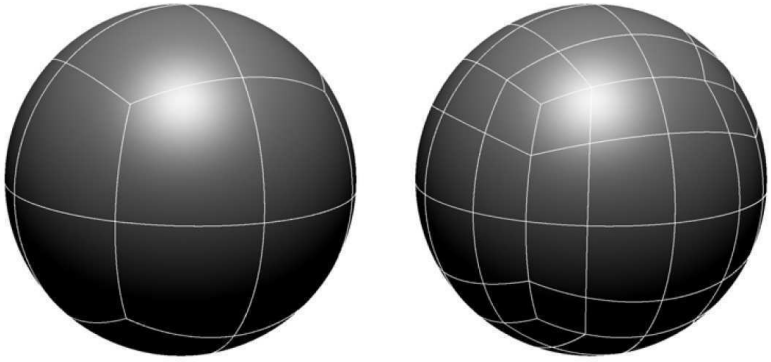


Figura 4.6: Divisão da esfera celeste pelo Q3C

fundidade de segmentação. Os pixeis finais não têm áreas iguais (como no HTM e HEALPix), uma vez que a propriedade de áreas de pixeis iguais é absolutamente desnecessária na indexação de banco de dados.

A estrutura ZONES, proposta em (GRAY; SZALAY; AL., 2004), divide a esfera celeste em zonas (anéis de DECs constantes), como na Figura 4.7. Em geral são milhares de zonas e cada uma delas tem um número. Os objetos são agrupados em suas respectivas zonas, de acordo com a sua posição (RA), facilitando sua localização. A cobertura da figura ultrapassa a esfera, mas só um pouco, considerando que, na prática, sempre usa-se zonas muito finas

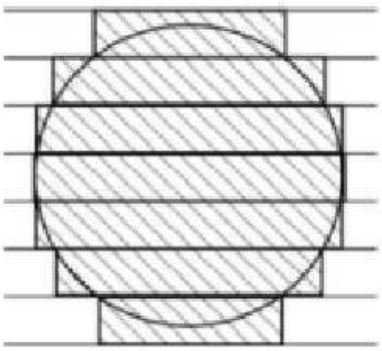


Figura 4.7: A esfera celeste aproximada pela união de intervalos de zonas. Fonte: (FAN et al., 2013)

Segundo (FAN et al., 2013), esta divisão simples tem uma grande vantagem sobre os sistemas hierárquicos mais complicados, pois qualquer zona tem apenas dois vizinhos diretos, ao contrário do HTM, HEALPix e Q3C. Além de ser muito mais fácil de implementar em qualquer linguagem de programação e se encaixar bem com os motores de bancos de dados.

Na Tabela 4.1, é apresentado o resumo das vantagens e desvantagens das diferentes tesselações apresentadas na Seção 4.2.1 usadas para particionar esfera

celeste e usadas para acelerar as consultas sobre os catálogos.

Index.	Abordagem	Vantagens	Desvantagens
HTM	Divisão hierárquica da esfera celeste em triângulos.	**1** - Suporta trigonometria esférica. **2** - Desempenho aceitável mesmo em bancos de dados astronômicos, contendo várias centenas milhões de objetos.	**1** - Faz muitos cálculos complexos, pois a área e a forma de cada pixel é diferente, e possui um esquema que limita sua performance. **2** - O BD para o qual foi desenvolvido não é *Open Source (Microsoft SQL Server)*. **3** - Cada triângulo possui 12 vizinhos. **4** - Problemas de orientação, isto é, todos os triângulos não estão orientadas na mesma direção.
HEALPix	Divisão hierárquica da esfera celeste em quadriláteros curvilíneos.	**1** - Áreas iguais para elementos discretos da partição. **2** - A Numeração dos pixeis é aninhada.	**1** - A forma dos pixeis varia, mas não é um problema com amostragem alta. **2** - A maioria dos pixeis tem 8 vizinhos, mas 8 deles tem 7 vizinhos. **3** - Problemas de orientação dos vizinhos aninhados.
Q3C	Divisão hierárquica da esfera celeste em quadrantes.	**1** - Mais rápido que o HTM no caso de alta profundidade de segmentação por causa do uso da quadtree. **2** - Desenvolvido para BD *open-source* (PostgreSQL). **3** - Fácil de usar.	**1** - Devido à divisão hierárquica, cada quadrante possui vários quadrantes vizinhos. **2** - No casamento é necessário filtrar os limites para encontrar os possíveis correspondentes.
ZONES	Divisão da esfera celeste em zonas (anéis de DEC).	**1** - Fácil de implementar em qualquer linguagem. **2** - Não necessita de extensões para consultas SQL. **3** - Ideal para cross-matchings, dada sua eficiência ao usar pequenos raios, como os da astronomia. **4** - No casamento usa-se os limites explícitos das coordenada, encontrando-se as possíveis correspondências antecipadamente.	**1** - Processamento limitado no desempenho. O algoritmo perde eficiência quando o raio é maior que a altura da zona.

Tabela 4.1: *Resumo das principais vantagens e desvantagens encontradas nas literaturas (ZHAO et al., 2009), (FAN et al., 2013), (DU et al., 2014), (WAY et al., 2012), (GRAY; SZALAY; AL., 2004), (KOPOSOV; BARTUNOV., 2012) (O'MULLANE et al., 2001) (SZALAY et al., 2005).*

4.2.2 Algoritmos de Casamento de Catálogos Astronômicos

Diversos trabalhos que apresentam algoritmos de *cross-matching* foram estudados durante o desenvolvimento desta tese. Classificamos os algoritmos dois tipos:

- aqueles que usam apenas a posição como critério para realizar o casamento (FU; FINK; AL., 2012) (KOPOSOV; BARTUNOV, 2006) (FIOC, 2012);
- e os que também usam outros atributos no casamento (ROHDE et al., 2005) (Sutherland; Saunders, 1992) (BUDAVÁRI; SZALAY, 2008) (ROHDE; GALLAGHER; DRINKWATER, 2012);

4.2.2.1 Algoritmos que utilizam apenas a posição como critério do casamento

Em (FU; FINK; AL., 2012), para tratar o problema de casamento entre um catálogo com objetos conhecidos e uma nova imagem do céu obtida, os autores assumem que as arestas da imagem são paralelas às direções de *ra* e *dec*. Logo, necessitam encontrar os casamentos para todos os objetos da imagem. Especificamente, para cada objeto *p* da imagem, procura-se por um objeto *q* no catálogo, tal que: entre todos os objetos na imagem, *p* é o mais próximo de *q*; entre todos os objetos do catálogo, *q* é o mais próximo de *p*; e a distância entre *p* e *q* é no máximo de 1 arcseg (1/3600 grau). Inicialmente, os objetos do catálogo são indexados por faixas, recuperando-se em memória todos os objetos do catálogo que estão na área da imagem estendida em 1 arcseg nos quatro lados, conforme a Figura 4.8. As faixas recuperadas são divididas em sub-faixas de 1 arcseg de largura, e os objetos que se encontram dentro das sub-faixas são ordenados por *ra*. Para cada objeto da imagem, o seu objeto correspondente só pode estar distante no máximo de 1 arcseg, e para tanto considera-se apenas os objetos da sua sub-faixas e das duas adjacentes. Calcula-se as distâncias entre cada um dos objetos das três sub-faixas ao objeto da imagem. A condição final para o casamento é: para cada objeto *p* da imagem, assume-se que o seu objeto do catálogo mais próximo é *q*; se o objeto mais próximo de *q* também é *p*, então *q* casa com *p*.

Um outro exemplo que usa critérios semelhantes de casamento de catálogos astronômicos pode ser encontrado na implementação do esquema de indexação do céu conhecido como Q3C (KOPOSOV; BARTUNOV, 2006), ele é bastante utilizado pela comunidade de astronomia. Foram escritas funções específicas para integrar o Q3C no PostgreSQL [2]. Dentre elas uma função SQL que realiza o casamento entre dois catálogos, a função *q3c_join*. Primeiramente, os objetos de um dos catálo-

[2] Disponível em http://code.google.com/p/q3c/

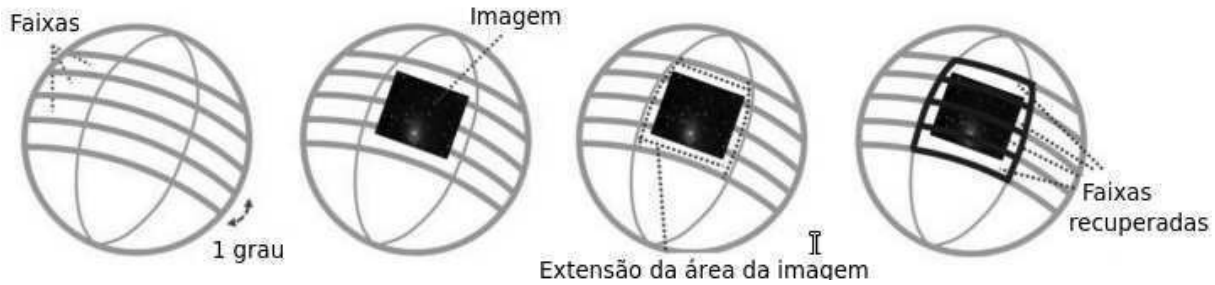


Figura 4.8: Seleção dos objetos do catálogo para serem carregados na memória.

gos devem ser indexados pela estrutura de indexação espacial Q3C, e então, uma consulta do tipo *select * from catalogo1, catalogo2 where q3c_join(catalogo1.ra, catalogo1.dec,catalogo2.ra,catalogo2.dec,0.00027)* deve ser executada, passando como parâmetros as variáveis correspondentes às posições dos objetos dos catálogos 1 e 2, e um raio de busca (neste exemplo é 0,00027 grau ou 1 segundo de arco). Supondo que existam as tabelas *catalogo1* e *catalogo2* e assumindo que a indexação Q3C foi criada no *catalogo2*, para cada registro do *catalogo1*, a função q3c_join() é convertida em quatro sub-consultas de intervalos correspondentes aos quadrantes da indexação do Q3C, projetados para aproximar ao círculo da busca de objetos candidatos ao casamento. Além disso, uma vez que esses quadrantes ocupam uma área maior que o raio em questão, a função faz uma filtragem adicional com base na distância deste raio. Se qualquer objeto do *catalogo1* estiver dentro desses limites, então o casamento é feito. Uma comparação entre esse algoritmo e o que foi desenvolvido durante o doutorado é apresentado no capítulo 7.

Em (FIOC, 2012), o autor descreve um método de casamento probabilístico a ser aplicado em um par de catálogos através das posições dos seus objetos e erros posicionais. A probabilidade de associação de um objeto com uma fonte de outro catálogo, ou que não há casamento, é derivada sob duas hipóteses exclusivas: em primeiro lugar, o caso clássico de associações de *vários-para-um*, e depois o mais realista, mas mais difícil problema de associações, de *um-para-um*. O método, implementado em Fortran95, denomina-se *Aspects*. *Aspects* é uma sigla em francês para "Association positionnelle/probabiliste de catalogues de sources" ("Associação probabilística/posicional de catálogos de fontes", em Inglês). A palavra francesa "*Aspect*" (no plural na sigla) tem o mesmo significado geral como a palavra correspondente em Inglês. Curiosamente, significa, em particular, a posição relativa de corpos celestes (FIOC, 2012).

Dados dois catálogos K e K' de n e n' objetos, respectivamente, *Aspects* calcula, para qualquer objeto $M_i \in K$ e $M'_j \in K'$, a probabilidade de casamento entre M'_j e M_i. Para determinar esta probabilidade de associação, denotada por $P(A_{i,j} \mid C \cap C')$

em (FIOC, 2012), onde $A_{i,j}$ refere-se à associação de i com j, C corresponde às coordenadas e às incertezas posicionais de todos os objetos de K e C' são todas aquelas de K'. *Aspects* também calcula as probabilidades $P(A_{i,0}|C \cap C')$ e $P(A_{0,j}|C \cap C')$ de M_i ou M'_j não terem candidatos o casamento. As três hipóteses exclusivas consideradas nesses cálculos são:

- $H_{v:1}$: Um objeto de K casa com pelo menos um de K', mas um objeto de K' casa com vários de K (**associações *K-K' vários-para-um***).
- $H_{1:v}$: Um objeto de K' casa com pelo menos um de K, mas um objeto de K casa com vários de K (**associações *K-K' um-para-vários***).
- $H_{1:1}$: Um objeto de K casa com no máximo um de K' e vice-versa (**associações *K-K' um-para-um***).

4.2.2.2 Algoritmos que permitem informações não posicionais no critério do casamento

Informações sobre a posição nem sempre são suficientes para estabelecer casamentos de forma confiável. Nestes casos, podem também ser utilizadas informações adicionais a partir das medições não-posicionais. Portanto, existem algoritmos de *cross-matching* que consideram, além da posição dos objetos, propriedades físicas como classificação estrela-galáxia, magnitude, cor, *redshift* e movimento próprio.

Em (ROHDE et al., 2005), os autores apresentam os resultados da aplicação de técnicas de aprendizagem de máquina automatizada para o problema de casamento de objetos de diferentes catálogos. O estudo apresenta o casamento posicional entre dois catálogos, cuja operação atingiu a 44% dos objetos. Porém, ao usar os algoritmos de aprendizagem de máquina, tais como máquina de vetores de suporte (SVM), *perceptron* e redes neurais, e considerando ainda outros atributos do objeto, aumentaram de 44% para 72% de objetos casados. Na validação do modelo, os autores mostram que conseguiram acertar $99,12\%$ por cento dos casamentos através de SVM.

Outra abordagem comum na literatura para melhorar o casamento por proximidade é a apresentada em (Sutherland; Saunders, 1992), na qual desenvolve-se um modelo probabilístico. O seu método é ideal desde que sejam conhecidas as informações de magnitudes e dos erros de posição. No artigo propõe-se uma fórmula para calcular a probabilidade de que qualquer candidato seja a identificação correta, de forma que este calculo também seja válido no caso de múltiplos candidatos. Na conclusão, os autores enfatizam que este método deve ser tratado como um complemento

para identificações espectroscópicas em vez de um substituto.

Em (BUDAVÁRI; SZALAY, 2008), os autores apresentam um formalismo probabilístico para o casamento de objetos astronômicos em várias observações. A abordagem Bayesiana, simétrica em todas as observações, é a base de um *framework* unificado para o casamento, onde não só informação espacial, mas as propriedades físicas, tais como cores, *redshift* e luminosidade, também podem ser consideradas de forma natural. No artigo é fornecida uma receita prática para implementar um algoritmo recursivo eficiente para avaliar o fator de Bayes sobre um conjunto de catálogos com erros circulares conhecidos nas posições.

(ROHDE; GALLAGHER; DRINKWATER, 2012) demonstra que através do emprego de um formalismo Bayesiano preditivo é possível utilizar todas as informações disponíveis para auxiliar na obtenção de casamentos mais confiáveis e ainda obter conclusões sem distorções. Distorções são evitadas porque distribuições de previsão são calculadas em todas as configurações de casamentos, ao contrário de outras abordagens que escolhem uma única configuração mais provável de casamento.

Nas soluções propostas nesta tese utilizamos apenas as coordenadas espaciais devido à sua simplicidade pois elas estão presentes em todos os catálogos. Enquanto que nem todos os catálogos possuem as demais informações. O uso de outras informações para melhorar os nossos resultados estão previstos como trabalhos futuros.

4.3 Particionamento e tratamento de fronteiras em algoritmos de clusterização

Nos algoritmos de clusterização que funcionam de forma distribuída, o conjunto de dados de entrada é dividido em partes menores e, em seguida processado paralelamente por alguma plataforma distribuída, como por exemplo, Hadoop Map/Reduce ou Spark. No entanto, a escolha de diferentes mecanismos de particionamento afeta a eficiência da execução e do balanceamento de cada nó. O particionamento faz com que objetos próximos alocados em partições diferentes ignorem a existência um do outro. Dependendo do método da clusterização, caso a vizinhança seja importante para produzir o resultado final, esse problema deve ser tratado no particionamento ou no tratamento das fronteiras. Nesta seção veremos algumas das soluções existentes na literatura.

DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) (ESTER et al., 1996) é um algoritmo bem conhecido para agrupamento baseado em densidade pois pode identificar os grupos com formas arbitrárias e lidar com conjuntos

de dados ruidosos. No entanto, com a quantidade crescente de dados, o algoritmo DBSCAN rodando em uma única máquina tem de enfrentar o problema de escalabilidade. O trabalho apresentado em (EL-SONBATY; ISMAIL; FAROUK, 2004) melhora a velocidade do DBSCAN usando o CLARANS (*Clustering Large Applications based upon Randomized Search*) (NG; HAN, 1994) para particionar o *dataset* e reduzir o espaço de busca de cada partição em vez de verificar todo o *dataset*. GRIDBSCAN (UNCU et al., 2006) constrói um *grid* que atribui os elementos em partições semelhantes e, em seguida, processa o DBSCAN em cada partição separadamente. Estes algoritmos melhoram a eficiência de DBSCAN, mas ainda não são suficientemente eficientes para processar dados em massa.

Uma estratégia Map-Reduce do DBSCAN, abreviada como DBSCAN-MR, foi proposta em (DAI; LIN, 2012) para tratar o problema de escalabilidade do DBSCAN. Os autores do artigo utilizaram um índice distribuído e otimizaram o balanceamento de carga e a eficiência de execução. O DBSCAN-MR é uma abordagem de processamento paralelo que pode ser executado de forma distribuída e não requer um índice global. Ele melhora a escalabilidade do DBSCAN dividindo os dados de entrada em partes menores e enviando-os aos nós para processamento paralelo. O balanceamento de carga de cada nó e a minimização do tempo total de execução são seus principais problemas.

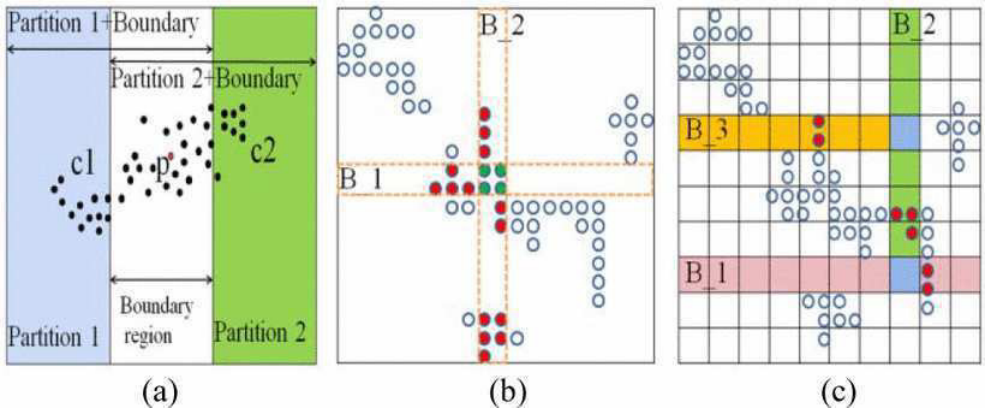


Figura 4.9: Exemplos de fronteiras. Fonte: (DAI; LIN, 2012)

Os elementos do mesmo agrupamento, provavelmente, estão espalhados entre diferentes nós. Quando o conjunto de dados é particionado, elementos das fronteiras devem ser replicados, a fim de fundir os agrupamentos espalhados nos diferentes nós. O número de elementos adicionais da fronteira afetam a eficiência da clusterização em cada nó e a mesclagem dos clusters resultantes de nós diferentes. Por exemplo, como mostrado na Figura 4.9 (a), o *dataset* é dividido em duas partes, a partição 1 (azul) e a partição 2 (verde), e cada parte é estendida para incluir a região

de fronteira (branco). Quando estes dois clusters de diferentes partições, como c1 e c2, são estendidos para a mesma região de fronteira, elementos na região da fronteira podem ajudar a determinar se estes dois clusters devem ser mesclados ou não. No entanto, os elementos da fronteira são replicados e colocados em todas as partições adjacentes, fato esse que aumenta a carga dos nós. Um outro exemplo, como mostrado nas Figura 4.9 (b) e 4.9 (c), onde as posições de partições diferentes são selecionadas, podemos observar que o número de elementos de fronteira na Figura 4.9 (b) é maior do que na Figura 4.9 (c), na qual o *dataset* é dividido em quatro partições. Na Figura 4.9 (b), o conjunto de dados é dividido pela fronteira B_1, e o número de elementos de fronteira é o dobro da quantidade de elementos de sua região (8 * 2 = 16) devido à replicação. Em seguida, o *dataset* é dividido pela fronteira B_2 e seus elementos são replicados em cada lado da fronteira (14 * 2 = 28). Portanto, há 44 (16 + 28) elementos de fronteira na Figura 4.9 (b). Em contraste, o número de elementos de fronteira na Figura 4.9 (c) é 14, e pode ser calculado similarmente (2 * 2 (de B_1) + 3 * 2 (de B_2) + 2 * 2 (de B_3) = 4 + 6 + 4 = 14) (DAI; LIN, 2012).

Podemos notar que uma grande quantidade de elementos de fronteira afetam a eficiência de execução da clusterização, pois estes elementos de fronteira não só aumentam a carga de cada nó, mas também aumentam o tempo de mesclagem dos resultados de diferentes nós. Além disso, o equilíbrio de carga de cada nó é uma preocupação importante para a elaboração do método de particionamento, pois o desequilíbrio de carga anula os benefícios do paralelismo. Pior ainda, toda a tarefa *Map/Recude* falha quando um nó fica sem memória. Para alcançar o equilíbrio de carga e melhorar o desempenho do *framework*, é preciso garantir que cada nó não fique sem memória e que tenhamos o número mínimo de elementos na fronteira. Para atingir estes objetivos, os autores de (DAI; LIN, 2012) projetaram uma abordagem chamada PRBP (*Partition with Reduced Boundary Points*).

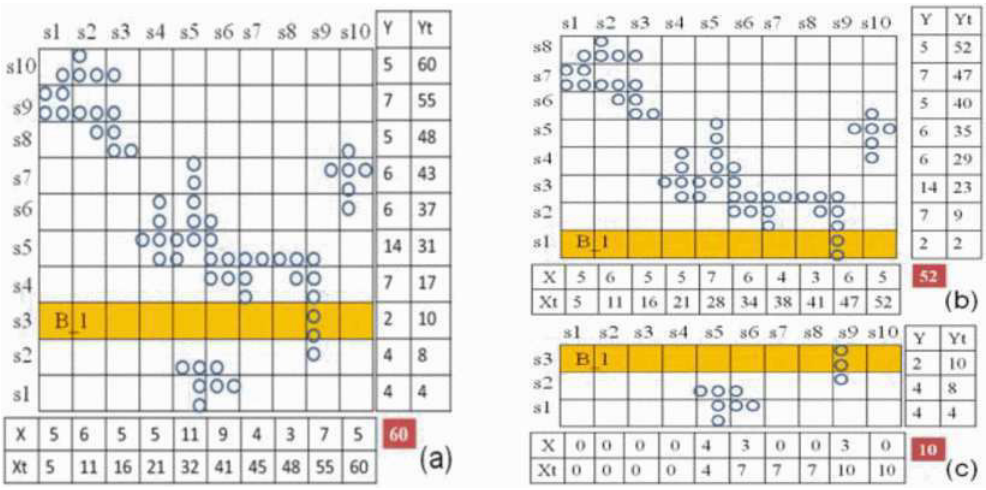
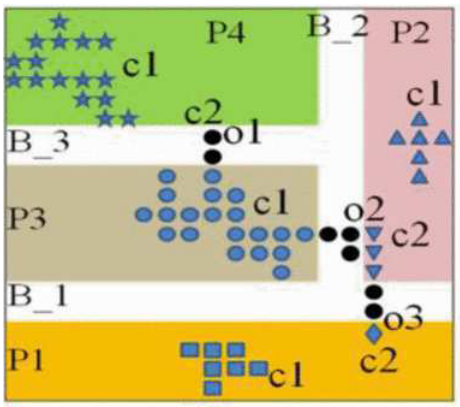


Figura 4.10: Estratégia de Particionamento PRBP. Fonte: (DAI; LIN, 2012)

A Figura 4.10 ilustra o particionamento *PRBP*. Na Figura 4.10 (a), para cada dimensão, a área dos dados é dividida em fatias com larguras iguais de tamanho 2ε, minimizando, assim, o número de elementos na fronteira, pois o *DBSCAN* estende o cluster caso encontre um elemento no raio ε. Logo, são necessários pelo menos 2ε na fronteira para conter informações suficientes para a mesclagem. As quantidades de elementos em cada fatia são registradas na colunas *X* e *Y*. E as quantidades acumuladas de elementos das fatias, por dimensão, são registradas nas colunas *Xt* e *Yt*. A fatia (Y_s3 = 2) com o menor número de elementos entre todas as dimensões é selecionada e torna-se a fronteira (B_1). (b) e (c) mostram os resultados desta etapa do particionamento, na qual a fronteira é replicada em cada partição e as informações de cada fatia são atualizadas para serem divididas ainda mais. O espaço é dividido recursivamente até que o tamanho estimado dos dados de cada partição caibam no espaço de memória de nós, evitando, assim, o problema de esgotamento de memória (DAI; LIN, 2012).



Local(P1,P2,P3,P4)	
key	value
P1 point index	P1c1 or P1c2
P2 point index	P2c1 or P2c2
P3 point index	P3c1
P4 point index	P4c1

Boundary(B_1,B_2,B_3)		
area	key	value
P3	o1	P3c1,isCore
P4	o1	P4c2,isCore
P2	o2	P2c2,isCore
P3	o2	P3c1,isCore
P1	o3	P1c2,isCore
P2	o3	P2c2,isCore
:	:	:

(b)

Figura 4.11: Resultado da clusterização do DBSCAN-Map. Fonte: (DAI; LIN, 2012)

A Figura 4.11 apresenta o resultado da clusterização do DBSCAN-Map. Em (a), o mesmo *dataset* da Figura 4.10 é dividido em quatro partes, P1, P2, P3 e P4 e eles são agrupados pelo DBSCAN-Map separadamente. Os resultados das clusterizações locais P1c1, P2c1, P3c1 e P4c1 são gerados e armazenados localmente e os clusters que envolvem as fronteiras, como mostrado em (b), são enviados ao sistema HDFS como dados de entrada do DBSCAN-Reduce, no qual a chave é o nome da fronteira comum entre as partições. Maiores detalhes estão presentes em (DAI; LIN, 2012).

Outros algoritmos de clusterização importantes nos trabalhos relacionados são FoF (*Friends-of-Friends*, amigos de amigos, em português) e dFoF (*distributed Friends-of-Friends*, *amigos-de-amigos* distribuído, em português) (KWON et al., 2010). Ele são importantes para esta tese pois suas estratégias são semelhantes a do algo-

ritmo SCIBoundary apresentado no Capítulo 6. *FoF* é um algoritmo de clusterização, semelhante ao DBSCAN, que aceita uma lista de objetos (*pid, x, y, z*) como entrada e retorna uma lista de tuplas de clusters (*pid, ClusterID*). Para calcular os agrupamentos, o algoritmo examina apenas a distância entre os objetos. *FoF* define dois objetos como amigos, se a distância entre eles é inferior a ε. Dois objetos são *amigos-de-amigos* se eles são alcançáveis no percorrimento do grafo induzido pelo relacionamento de amizade (KWON et al., 2010).

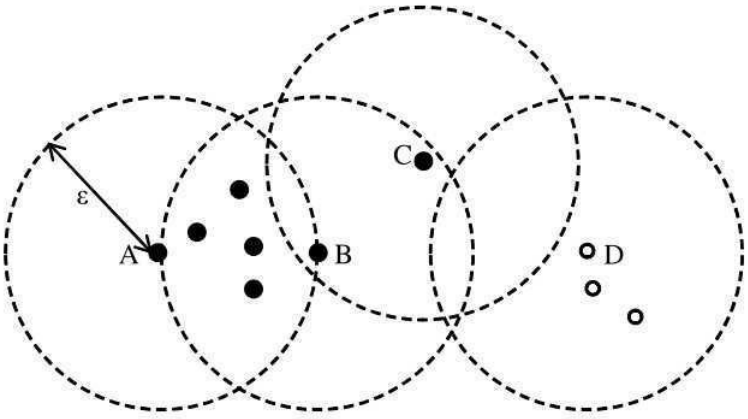


Figura 4.12: Algoritmo de Clusterização FoF. Fonte: (KWON et al., 2010)

No exemplo da Figura 4.12, dois objetos são considerados amigos, se a distância entre eles é inferior a um ε: A e B são amigos e B e C também são amigos, mas A e C não são. A relação de amizade é simétrica se a distância é simétrica. A relação amigo de amigo (FoF) é definida entre dois objetos, se eles são amigos ou se eles estão contidos no fecho transitivo da relação com o amigo (por exemplo, A e C é um par amigo de amigo através de B). Na figura, a relação FoF induz uma cluster de objetos: todos os objetos negros estão em um cluster e todos os brancos em outro.

No algoritmo $dFoF$ os dados são primeiramente particionados, depois agrupados localmente e, finalmente, os clusters locais são mesclados aos das partições vizinhas através do processamento da fronteira. O *dFoF* é dividido em três etapas: clusterização local, mesclagem hierárquica e nova rotulagem (KWON et al., 2010).

Durante o particionamento, uma região contígua do espaço é atribuída a cada nó para melhorar a probabilidade de que os objetos do mesmo cluster estejam no mesmo nó. A Figura 4.13 (a) ilustra uma divisão espaço 2D em quatro partições P1 a P4. Para determinar essas regiões uniformes, *dFoF* bissecciona recursivamente o espaço ao longo de todas as dimensões, até que o número estimado de objetos por região esteja abaixo do limite de memória do nó, de tal modo que o processamento local possa ser realizado inteiramente em memória. Uma vez que os dados são parti-

cionados, o algoritmo FoF original é executado dentro de cada partição. A saída desta fase é gravada em disco e consiste em um conjunto de pares: (pid, cid), onde pid é uma identificação de objeto e cid é um ID único e global do cluster. Cada objeto de entrada é rotulado com um ID do cluster. Os objetos com distância ε da fronteira de cada partição vão para a próxima fase. Eles servirão para identificar os clusters gerados localmente e que precisam ser mesclados com outros maiores (KWON et al., 2010).

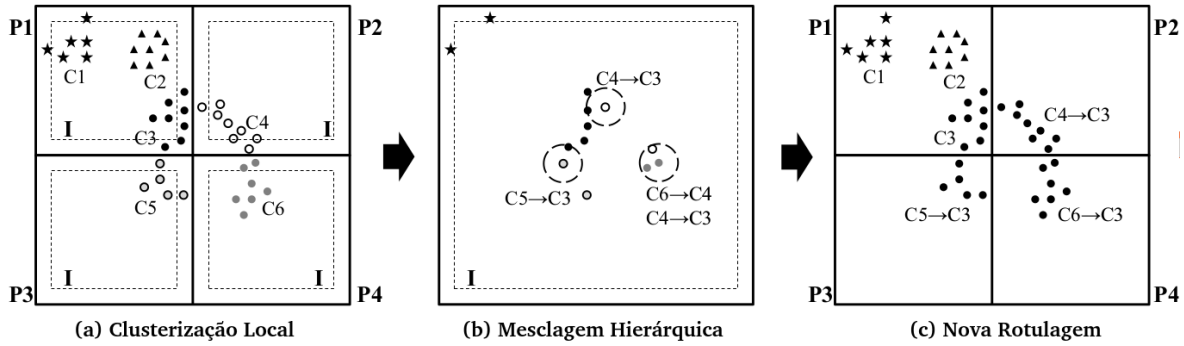


Figura 4.13: Etapas do algoritmo dFoF. Fonte: (KWON et al., 2010)

Para identificar os clusters que abrangem várias partições, objetos da fronteira devem ser examinados. Se objetos de partições adjacentes estão a uma distância ε um do outro, seus clusters devem ser mesclados. A Figura 4.13 ilustra as etapas de mesclagem para quatro partições $P1$ a $P4$. As áreas externas aos quadrados tracejados, Pi, representam as regiões das fronteiras e suas extremidades tem distância ε. Na Figura 4.13 (a), a etapa de clusterização local identifica um total de seis clusters rotulados de $C1$ a $C6$. Cada cluster possui objetos ilustrados com forma e tom de cinza diferente. No entanto, existem apenas três clusters globais neste *dataset*. Esses clusters são identificados durante o processo de mesclagem hierárquica. Os clusters C3, C4, C5, C6 e são mesclados porque seus objetos próximos à fronteira possuem distância ε um do outro. Apenas os objetos dentro de cada Pi, mas fora cada região I, são necessários para determinar que estes clusters devem ser mescladas. (b) demonstra a fase de mesclagem hierárquica. Nota-se que apenas os objetos das fronteiras em (a) são consideradas durante a fase de mesclagem. Após a mesclagem, três mapeamentos de cluster são gerados: ($C4$, $C3$), ($C5$, $C3$), e ($C6$, $C3$). Tais mapeamentos são usados para rotular novamente os clusters locais durante a fase de nova rotulagem tal como ilustrado em (c). Maiores detalhes do algoritmo são encontrados em (KWON et al., 2010).

4.4 Considerações Finais

Neste capítulo apresentamos os principais trabalhos relacionados com esta tese. Primeiramente abordamos as propostas de algoritmos de clusterização que são usados no problema de resolução de entidade. Em seguida, comentamos sobre os trabalhos que contribuem com soluções que apoiam o casamento por localização em astronomia através da representação do céu fazendo uso de estruturas de dados, a fim de facilitar a localização dos objetos e os seus vizinhos no espaço. Discutimos as características e apresentamos as vantagens e desvantagens de cada uma delas.

Também apresentamos diversos algoritmos de casamento estudados durante o desenvolvimento desta tese. Classificamos os trabalhos em duas vertentes: aqueles que usam apenas a distância como critério para realizar o casamento; e os que usam outros atributos no casamento. Dentre os trabalhos relacionados aos algoritmos de casamento em astronomia, não identificamos nenhum que trate do problema da escala. Propomos no Capítulo 6 uma solução para este problema.

E, finalmente, abordamos os trabalhos cuja ideia central consiste em criar algoritmos de clusterização que funcionam de forma distribuída e dão ênfase ao particionamento dos dados de entrada, bem como do tratamento das fronteiras, assuntos estes relacionados à nossa proposta do Capítulo 6. No entanto, existem algumas diferenças quanto à natureza dos dados e, consequentemente, o objetivo final dos algoritmos apresentados na Seção 4.3 é diferente do objetivo da nossa proposta. Enquanto os algoritmos aqui apresentados estão interessados em unir os objetos próximos para identificar os grupos com formas arbitrárias, nós estamos interessados em formar agrupamentos de objetos próximos, mas que pertençam a diferentes catálogos.

5 NACLUSTER: RESOLVENDO ENTIDADES A PARTIR DE MÚLTIPLOS CATÁLOGOS

5.1 Revisitando o Problema

Levantamentos espaciais usam instrumentos poderosos para navegar o espaço e identificar objetos de interesse dentro da região pesquisada. Os objetos são caracterizados individualmente com coordenadas espaciais, identificando a sua posição, além de outros atributos descritivos. Gerar uma visão integrada da área desejada com base em catálogos produzidos por diferentes levantamentos enfrenta um difícil problema de casamento de catálogos. Devido às variações na calibração dos instrumentos de captura, a posição de um único objeto espacial pode variar de um catálogo para outro. Além disso, em determinadas regiões densas do espaço, este problema é agravado por um grande número de potenciais candidatos ao casamento para cada objeto.

No capítulo 3 fizemos um estudo do caso de uso da unificação dos catálogos astronômicos e apresentamos a dificuldade na realização do casamento de objetos espaciais na astronomia (objetos celestes). Conforme mostrado no Capitulo 4, as abordagens tradicionais para lidar com esse problema usam uma distância limite ε, definida pelo usuário, para reduzir o número de candidatos correspondentes. Além disso, elas adotam uma abordagem para casar n catálogos aos pares. No entanto, quando se deseja calcular um casamento entre três ou mais catálogos, um processo mais cuidadoso deve ser aplicado, como averiguar se a ordem com que os catálogos são escolhidos para realizar o casamento pode produzir resultados diferentes ou se existe transitividade.

Neste Capítulo, propomos o NACluster, um algoritmo de clusterização não-supervisionado para realizar o casamento de vários catálogos espaciais. Usamos como cenário o domínio de Astronomia e os resultados experimentais são apresentados no Capítulo 7.

5.1.1 Premissas do Trabalho

Dentre as principais premissas do nosso problema podemos destacar:

- A realização do casamento de objetos de mesma natureza:
  - Os objetos a serem casados devem ser do mesmo tipo. Como exemplo, na natureza da astronomia, os objetos são corpos celestes que possuem

as coordenadas *RA*, variando de 0 a 360 graus, e *DEC*, variando de -90 a 90 graus. Caso o casamento seja de objetos de natureza geoespacial, cada entidade pode ser associada à um ponto localizado na Terra, cujas coordenadas são latitude, variando de 0 a 90 graus, e longitude, de zero a 180 graus para leste ou para oeste, a partir do Meridiano de Greenwich.

- Existência de um critério simples de casamento baseado no posicionamento dos objetos:
  - Dentre os vários atributos existentes nos catálogos espaciais, a posição dos objetos, composta por coordenadas, é comum a todos eles. Nosso desafio é utilizar apenas as coordenadas para realizar o casamento.
- Casamento não sensível ao raio de busca:
  - Desejamos que usar valores altos para o raio na realização do casamento não influencie negativamente na qualidade do casamento, ou seja, sua saída não produza ruídos provocados pelo raio.
- Desconhecimento do número exato de entidades:
  - Não queremos que seja preciso dar como entrada a quantidade de entidades a ser formada. Ela deve ser dinâmica e configurada de acordo com os objetos.

5.2 Formalização do Problema

O problema de resolução de entidades espaciais é formalizado através das Definições 2, 3, 4. Todas as definições desta tese também podem ser consultadas no Apêndice A.

Definição 2. *Dado um conjunto $S = \{C_1, ...C_n\}$ com n catálogos espaciais C_i, $1 \leq i \leq n$, tal que $C_i = \{o_{1i}, o_{2i}, ..., o_{mi}\}$, onde o_{ji}, $1 \leq j \leq m$, é um objeto espacial do catálogo i; o resultado da resolução de entidade $ER(S)$ é o conjunto E, tal que $E = \{e_1, e_2, ..., e_k\}$, com $k \geq max\ |C_i|$, onde $|C_i|$ corresponde ao número de objetos em um catálogo C_i, e e_p, $1 \leq p \leq k$, é uma entidade e representa um objeto espacial. Adicionalmente, $e_p = \{o_{i1}, o_{i2}, ..o_{ir}\}$, tal que $r \leq n$ e para cada catálogo C_j existe no máximo um $o_{ij} \in e_p$, onde $1 \leq i \leq m$.*

Definição 3. *Uma função binária d correspondente à distância Euclidiana entre objetos de catálogos e entidades é definida como $d : C_i \times E \longrightarrow \mathbb{R}$.*

Definição 4. *Uma função injetora M de mapeamento entre catálogos e entidades é definida como $M : C_i \longrightarrow E$, tal que, para cada $o_{ji} \in C_i$ e para todo $1 \leq i \leq n$, existe um $e_p \in E$, onde $d(o_{ji}, e_p) \leq \varepsilon$ e ε corresponde ao erro máximo de localização entre objetos de dois catálogos. Adicionalmente, $d(o_{ji}, e_p)$ é o valor mínimo dentre todas as entidades e_p.*

Proposição do Problema 1. *Determinar M tal que a Definição 4 seja verdade para um conjunto S e um dado valor de ε.*

Invariante 1. *Sejam $o_i \in C_i$ e $q_j \in C_j$ dois objetos espaciais. Seja ainda uma função M, conforme a Definição 4. Desta forma, $M(o_i) = M(q_j) = e_p$ se $C_i \neq C_j$.*

Neste capítulo, propomos uma abordagem de clusterização que segue as definições acima, onde um cluster é uma entidade e_p formada por objetos celestes o_{ji} de catálogos C_i diferentes, onde $1 \leq i \leq n$. Logo, no máximo temos n objetos por cluster. A clusterização é feita a partir da função de mapeamento M, onde cada objeto é associado a um cluster e_p mais próximo e cuja distância máxima é ε. Todos os objetos pertencentes ao cluster e_p representam o mesmo objeto real.

5.3 Algoritmo

NACluster é um acrônimo para **N-way CAtalog Clustering**, um algoritmo de clusterização não supervisionado para realizar casamentos entre múltiplos catálogos espaciais. Um conjunto S de *n* catálogos é fornecido como entrada e a saída é uma clusterização composta por k clusters. Neste contexto, um cluster corresponde ao conjunto de mapeamentos $M(o_{ji}) \longrightarrow e_p$, conforme a Definição 4. A Invariante 1 aborda uma restrição nos critérios de casamento entre os objetos, logo, é importante que todos os objetos associados ou mapeados a um cluster devam vir originalmente de catálogos diferentes. Além disso, cada cluster deve representar um objeto espacial na região observada, segundo a Definição 2. Assim, não se sabe de antemão quantos clusters serão produzidos ao final da execução do algoritmo. Estas restrições nos abstém de adotar um algoritmo de clusterização tradicional, como *k-means*. Comparações mais específicas com o *k-means* podem ser vistas na seção 5.3.2.

O NACluster é uma adaptação do *K-means*, onde os agrupamentos representam as entidades para as quais os objetos de catálogos devem ser mapeados. Assim, no NACluster, para cada objeto dos catálogos envolvidos, procura-se o cluster cuja distância entre seu centroide e o objeto seja menor que um valor dado, chamado de $epsilon$ (ε).

A ideia do algoritmo (detalhado no Algoritmo 1) é comparar cada objeto com centroides de clusters distantes até ε, definido pelo usuário, usando a distância Euclidiana $d(o_i, C_a)$ de um objeto o_i para um centroide C_a. Quando existe um centroide C_a, tal que $d(o_i, C_a) < \varepsilon$, então este centroide é classificado como potencial candidato a ser mapeado ao objeto o_i. Dentre os potenciais candidatos, aquele com menor distância ao objeto o_i é mapeado a ele. Esse mapeamento, no entanto, só pode ser aplicado quando não existe um outro objeto o_j no cluster de C_a que faça parte do mesmo catálogo de o_i, segundo a Invariante 1.

Algoritmo 1 NACluster

```
1: Input: Conjunto S de catálogos
2: Output: Todos os clusters e seus objetos e centroide;
3: procedure CLUSTERING(S)
4:     indexedCentroids ← initializeClusters(largestCatalog); ▷ Cria os primeiros clusters
   com centroides tendo as mesmas coordenadas dos objetos do maior catálogo. Retorna a variável
   que indexa os centroides e contem seus respectivos clusters
5:     for all obj ∈ S do
6:         candidates ← rangeQuery(obj,indexedCentroids,ε); ▷ Busca indexada por centroides em raio
   ε. Retorna uma lista de tuplas <centroide,distancia>.
7:         indexedCentroids ← searchCentroid(obj,candidates,indexedCentroids); ▷ Ver Algoritmo 2
8:     end for
9: end procedure
```

No caso de um objeto o_j do mesmo catálogo de o_i já existir no cluster de C_a, é preciso tratar a colisão de mapeamento.

Tratamento de colisão de Mapeamento:

1. se $d(o_j, C_a) > d(o_i, C_a)$ então é necessário remover o objeto o_j do cluster de C_a, inserir o_i neste cluster, e procurar um outro cluster para o_j;

2. se $d(o_j, C_a) < d(o_i, C_a)$ então o algoritmo busca dentre os demais centroides candidatos a serem mapeados ao objeto o_i um novo cluster para realizar o mapeamento (detalhado no Algoritmo 2).

Caso nenhum cluster seja encontrado na distância ε, um novo cluster com centroide C_b é criado para o ponto o_i, no qual sua posição é o centroide. O algoritmo termina quando todos os objetos de todos os catálogos forem avaliados.

A implementação do NACluster foi feita em Java. Para indexar os centroides usamos a *PH-tree*, apresentada na seção 2.4.1, reduzindo, assim, o espaço de busca. Os experimentos realizados com o execução da implementação podem ser vistos no Capítulo 7.

A implementação do algoritmo possui código aberto e está disponível à comunidade no endereço https://github.com/vinipires/NACluster.

Algoritmo 2 Função recursiva que busca o cluster apropriado para alocar o objeto obj do catálogo $obj.cat$ na lista de centroides candidatos $candidates$. Ao final, a variável $indexedCentroids$, que indexa os centroides e contém os clusters, é atualizada e retornada como saída.

```
1: Input: o objeto, a lista de candidatos e a variável indexada de centroides.
2: Output: a variável indexada de centroides atualizada
3: function SEARCHCENTROIDE(obj,candidates,indexedCentroids)
4:     if candidates.size = 0 then
5:         create cluster;
6:         cluster.add(obj);
7:         cluster.updateCentroid();                ▷ Atualiza as coordenadas do centroide do cluster
8:         indexedCentroids.add(cluster);           ▷ Adiciona o cluster e indexa seu centroide
9:     else
10:        minDistance ← Double.MAX_VALUE;
11:        for all centroid ∈ candidates do
12:            distance ← candidates.getDistance(centroid);
13:            if distance < minDistance then
14:                minCentroid ← centroid
15:                minDistance ← distance
16:            end if
17:        end for
18:        cluster ← indexedCentroids.getCluster(minCentroid);   ▷ Retorna o cluster correspondente
19:        if ∄ o ∈ cluster| o.cat=obj.cat then ▷ Não existe objeto no cluster do mesmo catálogo de obj
20:            cluster.add(obj);
21:            cluster.updateCentroid();            ▷ Atualiza as coordenadas do seu centroide
22:        else                                     ▷ Objeto o presente no cluster é do mesmo catálogo de obj
23:            oldObj ← o;
24:            oldDistance ←computeDistance(oldObj,minCentroid);
25:            if minDistance < oldDistance then
26:                cluster.remove(oldObj);
27:                cluster.add(obj);
28:                cluster.updateCentroid();
29:                candidates.clear();
30:                minDistance ← Double.MAX_VALUE;
31:                candidates ← rangeQuery(oldObj,indexedCentroids,ε);   ▷ Busca indexada por
   centroides em raio ε a partir de oldObj. Retorna uma lista de tuplas <centroide,distancia>.
32:                for all centroid ∈ candidates do
33:                    if centroid = minCentroid then
34:                        candidates.remove(centroid);
35:                    end if
36:                end for
37:                indexedCentroids ← searchCentroid(obj,candidates,indexedCentroids);
38:            else
39:                candidates.remove(minCentroid);
40:                indexedCentroids ← searchCentroid(obj,candidates,indexedCentroids);
41:            end if
42:        end if
43:    end if
44:    Return indexedCentroids;
45: end function
```

5.3.1 Índice Espacial sobre Catálogos

Algoritmos de clusterização que necessitam encontrar centroides próximos aos objetos podem fazer computações desnecessárias ao realizar essa busca. Ingenuamente compara-se a distância de cada objeto a todos os centroides, e assim, verifica-se qual é o mais próximo. A complexidade computacional dessa busca ingênua é da ordem de $O(n \times p)$, onde n é o número de objetos e p é a quantidade de centroides. Para reduzir essa complexidade é necessária uma estrutura de dados que possa indexar os centroides e reduzir o número de comparações para se obter os centroides vizinhos a uma distância ε.

O NACluster utiliza uma estrutura de indexação espacial multidimensional para acesso rápido às regiões do espaço. A estrutura adotada requer suporte eficiente à atualização, pois o centroide muda de posição cada vez que um objeto é inserido em seu cluster.

Estruturas bem conhecidas, como a KD-Tree, tem um alto custo de atualização, na ordem de $O(n)$ no pior caso e *O(log n)* no caso médio (ZÄSCHKE; ZIMMERLI; NORRIE, 2014), e a árvore precisa ser rebalanceada a cada atualização. Logo ela não é ideal para ser usada no NACluster.

O NACluster usa a PH-tree (ZÄSCHKE; ZIMMERLI; NORRIE, 2014) na indexação dos centroides, uma vez que ela permite que atualizações sejam feitas com baixa complexidade computacional ($O(1)$ no melhor caso e *O(log n)* no pior caso), de tal forma que rebalanceamentos não são necessários.

A *PH-Tree* suporta dois tipos de consultas, *point queries* e *range queries*. *Point queries* verificam a existência ou não de um determinado objeto na árvore. *Range queries* buscam por todos os objetos dispostos dentro de uma área retangular definida pelo ponto do canto inferior esquerdo e pelo ponto do canto superior direito. Dessa forma, indexando a posição dos centroides de cada cluster, podemos realizar, a partir de cada objeto, a busca por centroides em um raio ε. Mais detalhes sobre a PH-Tree podem ser vistos na seção 2.4 do Capítulo 2.

5.3.2 Adaptação do K-Means

Conforme visto no Capítulo 2, o *k-means* é um algoritmo de agrupamento que tem por objetivo classificar um conjunto de elementos de entrada em *k* clusters. A estratégia de casamento do NACluster é semelhante ao mapeamento de objetos a grupamentos no *k-means* nos seguintes aspectos:

- Os dois algoritmos usam o conceito de centroides em clusters, nos quais a posição de cada centroide é a média da posição dos objetos do cluster;
- Tanto o *k-means* quanto o NACluster definem centroides iniciais. Em geral, no *k-means* defini-se *k* centroides aleatoriamente. No NACluster define-se *x* centroides iniciais, tal que *x* é o número de objetos do maior catálogo;
- Ambos utilizam a abordagem de alocar o objeto ao cluster de centroide mais próximo.

Apesar de existirem características semelhantes entre as duas abordagens, existem várias diferenças. Dentre elas:

- Relaxamento do *k*: diferentemente do *k-means*, o número de clusters formados, *k*, não é dado como entrada. No NACluster, a quantidade final de clusters só é descoberta após toda a execução.
- Casamento determinístico na maioria dos casos: dado um conjunto S de *n* catálogos, o NACluster tem, para o mesmo conjunto a cada execução, os mesmos centroides iniciais, ou seja, a posição dos objetos do maior catálogo, e produzirá a mesma sequência de clusters, desde que a distância entre um objeto aos dois centroides mais próximos seja a mesma. Neste caso, o algoritmo escolhe aleatoriamente um cluster para o casamento, no entanto este é um caso raro e é abordado na Seção 5.3.3. No *k-means*, os centroides iniciais são aleatórios e a sequência na formação dos clusters é diferente na maioria das vezes.
- Existem restrições de tamanho do cluster e tipos de objetos que podem pertencer a ele: no NACluster, o número de objetos mapeados para um agrupamento não pode ultrapassar a quantidade de catálogos existentes. Cada cluster pode conter apenas um objeto de cada catálogo. No *k-means* não existe restrição quanto ao tamanho do cluster e o tipo de catálogo que ele pode alocar.
- Tratamento de colisão: no NACluster é possível que um objeto seja removido do cluster ao qual ele tenha sido adicionado anteriormente e seja alocado em outro cluster caso nele ocorra uma colisão de objetos do mesmo catálogo, ou seja, um objeto tentar ser adicionado em um cluster onde já existe outro objeto do mesmo catálogo. Neste caso, aquele que for mais próximo do centroide permanece no cluster. No *k-means* não existe tratamento de colisão e objetos diferentes catálogos podem fazer parte do mesmo cluster.

Com tantas diferenças, podemos concluir que o *k-means* não iria satisfazer os pré-requisitos de casamento do NACluster. A falta das restrições no *k-means*, bem

como o k fixo, fariam com que um cluster não representasse um objeto do mundo real.

5.3.3 Situações particulares da execução do NACluster

Para que o leitor entenda melhor nossa proposta, apresentamos duas diferentes situações que podem ocorrer durante a execução do NACluster.

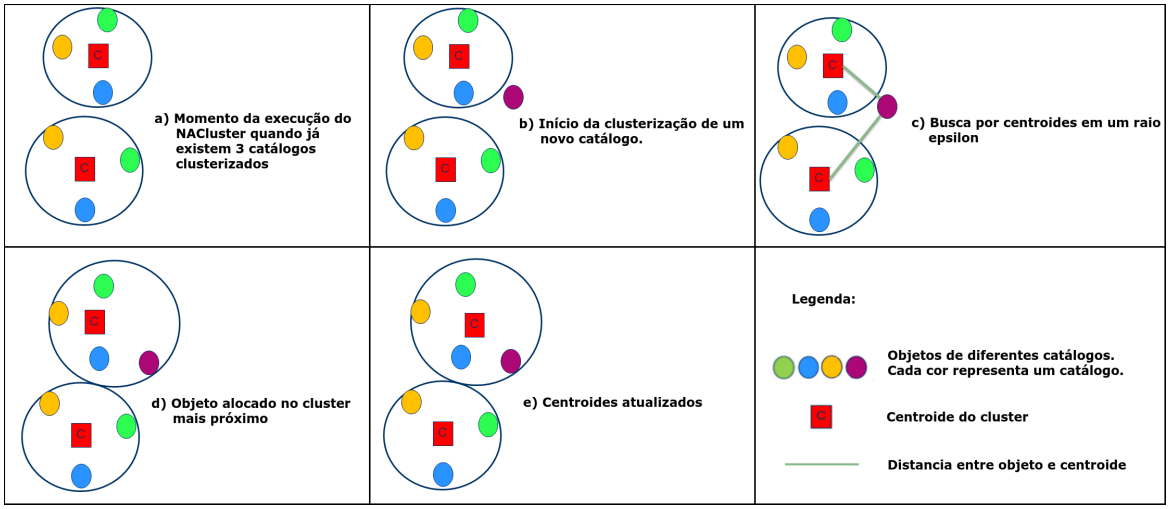


Figura 5.1: Situação 1 de casamento.

A primeira situação é mostrada na Figura 5.1. Suponha que estamos fazendo o casamento de quatro catálogos. Cada elipse representa um objeto no espaço e cada cor corresponde a um catálogo diferente.

O momento ”*a*” da Figura 5.1 ilustra uma situação que pode ocorrer em um ponto da execução do NACluster nessa região do espaço. Existem dois clusters formados, cada um com centroide ”*C*” e três objetos de diferentes catálogos. No instante *”b”*, surge um objeto de um quarto catálogo. Esse objeto busca, em ”*c*”, por centroides em um raio ε, encontra os dois clusters e em ”*d*” aloca-se ao mais próximo. Depois da nova alocação o centroide precisa ser atualizado. Essa atualização ocorre no momento ”*e*” e resolve-se o problema do casamento para esse objeto segundo o NACluster.

A segunda situação é mostrada na Figura 5.2 através dos momentos no intervalo de ”*a*” até ”*h*”. Abaixo comentamos cada um deles:

a) Momento em que a execução do NACluster está ocorrendo nessa região do espaço e existem dois clusters formados, cada um com centroide ”*C*” e quatro objetos de diferentes catálogos.

b) Instante em que um novo objeto precisa ser mapeado para seu grupamento.

c) O objeto busca por centroides em um raio ε e encontra dois clusters.

d) De acordo com o NACluster, é necessário verificar se o cluster de centroide mais próximo possui objeto do mesmo catálogo e, em ”*d*”, essa verificação é constatada.

e) Para que a explicação fique mais fácil, nomeamos os objetos o_1 e o_2. Podemos perceber que a distância de o_1 ao centroide é menor que a distância do objeto o_2 que já faz parte do cluster.

f) o_2 é removido, o_1 é inserido, atualiza-se a posição do centroide e busca-se por outros centroides em um raio ε para alocar o objeto removido o_2.

g) Percebe-se que já existe um outro objeto do mesmo catálogo no cluster, e a distância do objeto o_1 ao cluster candidato é maior do que a distância do objeto alocado.

h) Finalmente, cria-se um novo cluster para alocar o_2, pois o mesmo não tinha opções para ser alocado.

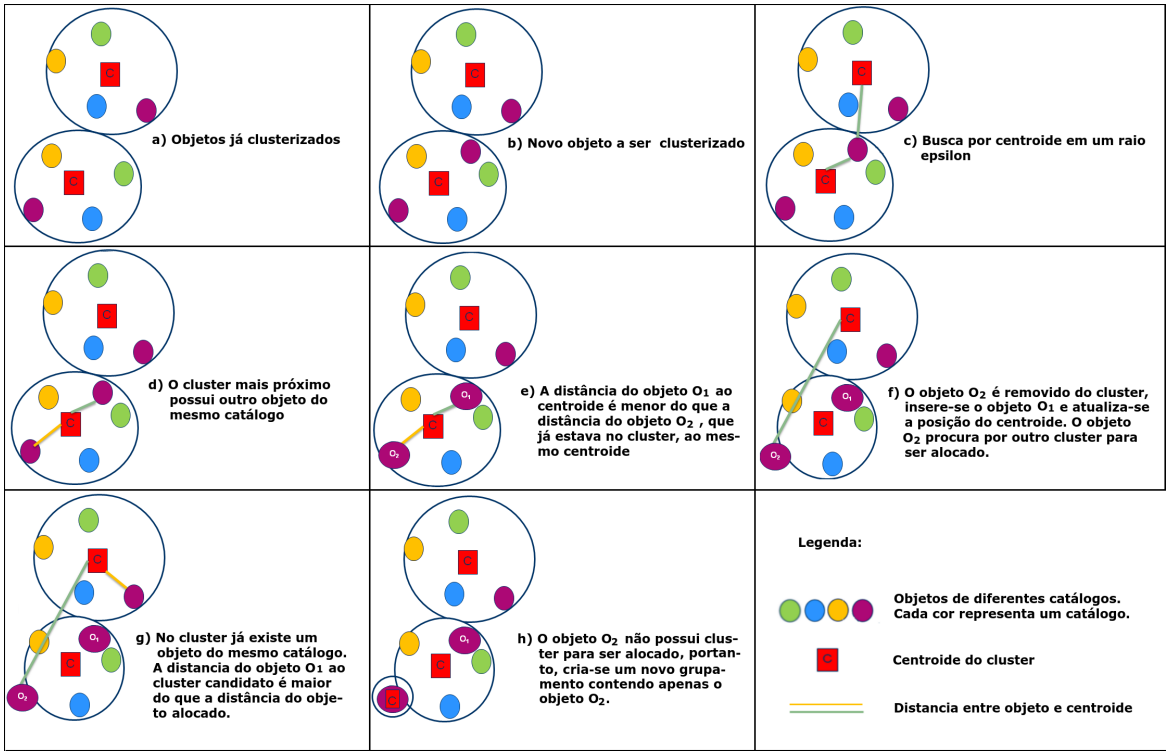


Figura 5.2: Situação 2 de casamento.

5.3.4 Complexidade

Graças à estratégia de indexação espacial através da *PH-Tree*, o número real de comparações entre objetos e centroides é reduzido para uma região local, fazendo com que o algoritmo seja viável na prática e apresentando um enorme potencial para paralelização. A complexidade de uma execução completa do NACluster com a PH-tree é reduzida a $O(n \times l)$, onde n é o número total de objetos do conjunto de catálogos S usado como entrada e l é a quantidade de centroides vizinhos dos objetos. No pior caso, todos os centroides deveriam ser vizinhos de todos os objetos do conjunto S, e cada objeto poderia refazer sua alocação aos clusters vizinhos. Neste caso, a complexidade seria $O(n \times l^2)$, onde l é o numero total de centroides.

5.4 Grau Médio de Ambiguidade

Com o objetivo de encontrarmos alguma correlação entre a qualidade da clusterização do NACluster e a ambiguidade existente no agrupamento, propomos uma métrica para avaliar a ambiguidade do ponto de vista dos objetos, ou seja, o quanto cada objeto é ambíguo em relação aos clusters.

Considerando as definições apresentadas na formalização do problema (Seção 5.2) e que uma entidade e_p é um cluster resultante na execução do NACluster, apresentamos as definições a seguir.

Definição 5. *Uma função $Cand$ de mapeamento entre catálogos e entidades é definida como $Cand : C_i \longrightarrow E$, tal que, para cada $o_{ji} \in C_i$ e para todo $1 \leq i \leq n$, existe um ou mais $e_p \in E$, onde $d(o_{ji}, e_p) \leq \varepsilon$ e ε é a distancia máxima.*

Em outras palavras, a função $Cand$ mapeia cada objeto a um ou mais clusters candidatos à alocá-lo. A diferença da função $Cand$ para a função M, apresentada na Definição 4, é que enquanto M mapeia cada objeto à apenas uma entidade, $Cand$ mapeia cada objeto à uma ou mais entidades.

O Grau de Ambiguidade pode ser definido através da quantidade de clusters candidatos.

Definição 6. *Seja $|Cand(o_{ji})| = a$, o seu grau de ambiguidade é $G(o_{ji}) = a - 1$.*

O Grau de Ambiguidade de um objeto o_{ji} é proporcional ao número de clusters e_p cujos centroides tenham distância menor ou igual a ε, situação esta ilustrada na Figura 5.3. Portanto, de acordo com a Definição 6, um objeto com a clusters candidatos a alocá-lo, tem o seu *grau de ambiguidade* em relação aos a clusters no valor de $G(o_{ji}) = a - 1$, pois ele deve ser alocado em ao menos um deles.

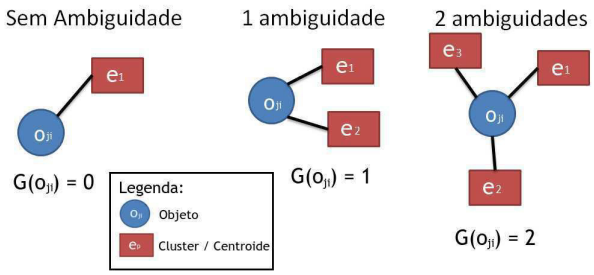


Figura 5.3: Grau de Ambiguidade do objeto o_{ji}

A métrica proposta para medir a ambiguidade é formalizada na Definição 7.

Definição 7. *Seja* $\sum_{i=1}^{n} |C_i|$ *a quantidade total de objetos do conjunto de catálogos, o grau médio de ambiguidade é* $\overline{G} = \frac{\sum_{j=1}^{m} \sum_{i=1}^{n} G(o_{ji})}{\sum_{i=1}^{n} |C_i|}$*, tal que* m *é a quantidade total de objetos do catálogo* i*.*

Nos experimentos do Capítulo 7 apresentamos um estudo sobre a correlação a qualidade do casamento e o grau médio de ambiguidade proposto.

5.5 Considerações Finais

Neste capítulo foi apresentada a primeira contribuição desta tese, o NACluster, uma solução para a resolução de entidades de dados espaciais.

O algoritmo proposto realiza o casamento de n catálogos através da clusterização de objetos vizinhos. Na busca da vizinhança há uma indexação para reduzir o espaço de busca. Cada cluster produzido representa um objeto real, ou seja, uma entidade, e não deve possuir mais de um objeto do mesmo catálogo. Assim, não se sabe de antemão quantos clusters serão produzidos ao final da execução. Esta restrições nos abstém de adotar um algoritmo de clusterização tradicional, como *k-means*.

Com uma complexidade na ordem de $O(n \times l)$, onde n é o número de objetos do conjunto S usado como entrada e l é a quantidade de centroides vizinhos dos objetos, o NACluster é viável para execução. Seus experimentos podem ser vistos no Capitulo 7.

6 PARALLELNACLUSTER

Este capítulo apresenta três contribuições deste trabalho. A primeira delas é o algoritmo de clusterização *ParallelNACluster*, uma solução para a resolução de entidades em larga escala a partir de múltiplos catálogos de astronomia. A segunda contribuição é o algoritmo *SCIBoundary* para o tratamento das fronteiras, de forma que o resultado do casamento ao final do ParallelNACluster seja semelhante ao centralizado, mesmo após o particionamento do *dataset*. A terceira contribuição apresentada neste capítulo é *AODP*, um workflow para particionamento dos dados em disco local e execução do casamento dos mesmos em ambiente distribuído através do *ParallelNACluster*.

6.1 Visão Geral do Problema

Para realizar o casamento entre bilhões de objetos através do NACluster de forma centralizada é necessário ter uma máquina com uma grande quantidade de memória RAM. Em nossos experimentos a versão centralizada realizou o casamento de 1 Bilhão de objetos utilizando cerca de 500 GB de memoria RAM em 2 horas e 41 minutos em uma máquina com 48 processadores. Dessa forma, existe uma limitação física na maioria das máquinas usadas pela comunidade que impedem a realização de casamentos de grandes catálogos através do NACluster. Neste ano de 2016, um computador com a quantidade de memória nesta ordem de grandeza custa muito caro, tornando esse tipo de execução inviável também financeiramente. Todas essas características nos motivam a criar uma versão distribuída do NACluster, para que possam ser feitos casamentos em larga escala de múltiplos catálogos de astronomia através de um cluster formado por máquinas baratas.

Antes de entrarmos na abordagem, precisamos entender, em linhas gerais, sobre arquitetura distribuída e o processamento paralelo. Segundo (MATTOS, 2008), a evolução do hardware e a diminuição dos seus custos, os avanços nas tecnologias de comunicação nas redes locais e nas redes de longa distância, permitiram o surgimento dos conceitos de clusters para o processamento paralelo. O cluster é um tipo de sistema de processamento paralelo ou distribuído que consiste de um conjunto de computadores interconectados trabalhando em conjunto para executar aplicações e integrando recursos computacionais. Seu objetivo é fazer com que todo o processamento da aplicação seja distribuído aos computadores, de forma que pareça um único computador. Com isso, é possível realizar processamentos que até então somente computadores de alto desempenho seriam capazes de fazer.

Cada computador do cluster é também chamado de nó. Na área de banco de dados é comum a comunidade usar sistemas distribuídos com a arquitetura *master/slave* (Figura 6.1), onde o mestre (*master*) consiste em um único nó que é responsável por atribuir tarefas aos demais nós (escravos ou *slaves*). O *master*, por sua vez, é controlado por outro processo executado pelo usuário. Outro papel importante do *master* é o de verificar o estado dos *slaves* para saber se estão ativos. Caso algum não responda em um prazo predefinido, o *master* assume que o mesmo falhou (PINTO, 2015).

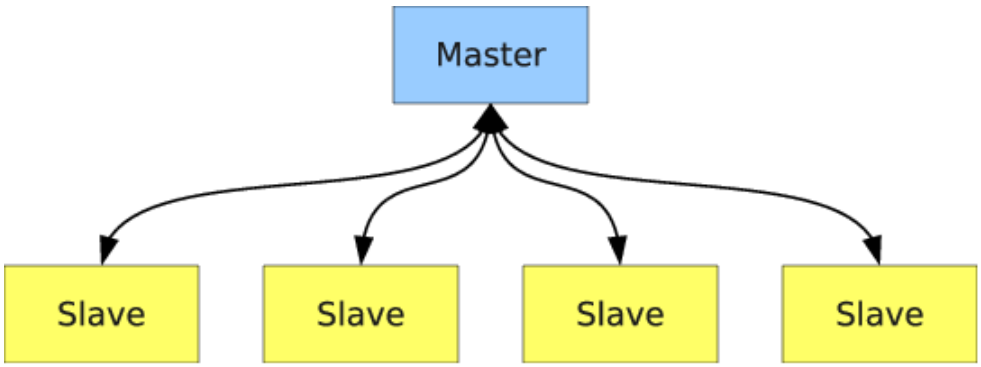


Figura 6.1: Arquitetura distribuída Master/Slave

Processar dados em paralelo requer o particionamento dos mesmos. Dessa forma, os dados devem ser divididos em partes menores, chamadas partições. O particionamento distribui objetos pelos nós de processamento induzindo acesso distribuído quando referências a objetos em nós distintos são realizadas. A necessidade de particionamento vai de encontro com a noção de proximidade utilizada pelo método de clusterização. É importante que o aspecto físico de alocação de dados em partições não prejudique a noção lógica de proximidade entre os objetos, necessária ao correto processo de clusterização. De modo a tratar esse problema, introduz-se o conceito de fronteira que determina critérios lógicos de pertinência de objetos em partições. As fronteias são os locais onde são feitas as divisões no particionamento. Levando em consideração que dois objetos são vizinhos quando a distância entre eles é no máximo ε, o tratamento das fronteiras deve permitir que os objetos vizinhos localizados em partições diferentes, bem como vizinhos dos vizinhos, e assim sucessivamente, sejam processados juntos em alguma etapa posterior da aplicação e o resultado desse processamento seja usado de forma que o particionamento não interfira no resultado final da clusterização. A não realização do tratamento das fronteiras pode influenciar diretamente na clusterização. No Capítulo 4 vimos duas formas diferentes de tratamento das fronteiras geradas pelo particionamento. Para esta tese, consideramos duas categorias de objetos:

- os objetos influenciados pelas fronteiras, ou seja, aqueles cuja distância seja ε à

alguma das fronteiras, bem com seus vizinhos, os vizinhos dos vizinhos, e assim sucessivamente;

- e os demais objetos, isto é, aqueles não influenciados pela fronteira.

Existem diversas formas de particionar os dados e a alocação de partições aos nós do cluster interfere no balanceamento de carga. Balanceamento de carga é uma técnica para distribuir a carga de trabalho uniformemente no sistema distribuído a fim de otimizar a utilização de recursos, maximizar o desempenho, minimizar o tempo de resposta e evitar sobrecarga. O seu objetivo é equilibrar a carga de todas as máquinas (KRUEGER; CHAWLA, 1991). Ao desenvolver uma aplicação de processamento paralelo, deve-se estar atento a estes conceitos.

Nesta tese, consideramos um modelo de distribuição e paralelismo em que existem k nós de processamento de dados, onde em cada nós há disponível uma quantidade M de memória, um número p de núcleos de processamento e um volume V de espaço em disco local. Um processo é alocado a um nó de processamento se naquele nó existe uma partição P dos dados que deseja processar.

Precisamos definir alguns conceitos:

Definição 8. *(Partição de um Catálogo) Uma partição P_i de um catálogo C representa um subconjunto de C de tal forma todos os objetos em P_i encontram-se entre duas fronteiras espaciais f_i e f_{i+1}, de tal forma que a coordenada espacial associada a f_i tem um valor inferior à coordenada espacial associada à f_{i+1}.*

Definição 9. *(Balanceamento entre partições) Dadas duas partições P_i e P_j, dizemos que estão balanceadas se o número de objetos em P_i, $\mid P_i \mid$, é aproximadamente igual ao de P_j, $\mid P_j \mid$. Logo, $\mid P_i \mid + \delta = \mid P_j \mid$, tal que δ é um valor tolerável na diferença da quantidade de objetos das duas partições.*

Podemos fazer uma analogia do balanceamento entre as partições apresentado na Definição 9 com os histogramas *equi-depth*, ou equi-frequência, nos quais suas faixas possuem aproximadamente a mesma altura, ou frequência.

Neste capitulo, estamos interessados em técnicas que permitam a paralelização do algoritmo NACluster. Suas características criam alguns desafios a serem superados na paralelização:

- Os objetos vizinhos distantes em até ε devem ser executados juntos, portanto devem estar associados à mesma partição lógica.
- A alocação dos objetos em diferentes nós de forma desbalanceada pode fazer com que algumas máquinas executem o NACluster por mais tempo. Ou seja,

enquanto as máquinas com mais objetos continuariam a clusterização, as demais terminariam o casamento e ficariam ociosas até o processo terminar por completo.

- O particionamento de um conjunto de dados gera fronteiras, f_i e f_{i+1}, entre partições, uma vez que as fronteiras precisam ser conhecidas e tratadas.

6.1.1 O problema de clusterização sob o efeito de dados particionados (CDP)

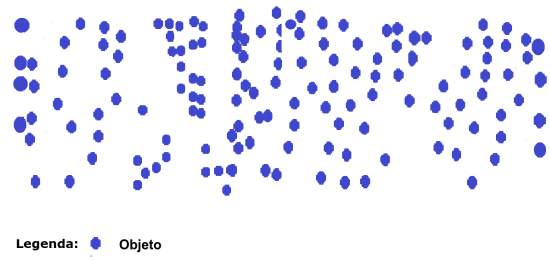
Dado um conjunto de dados C e um critério de proximidade cp, o problema CDP está em determinar um agrupamento G de tal forma que maximize a proximidade de objetos em grupos, considerando a distribuição de dados pelos nós de um cluster.

Para resolver o problema, podemos usar o modelo *MapReduce*. Neste caso, o NACluster pode ser processado como uma função Map e o tratamento associado ao particionamento pode ser feito na etapa Reduce. Neste sentido, faz-se necessário identificar os dados para os quais o tratamento é necessário.

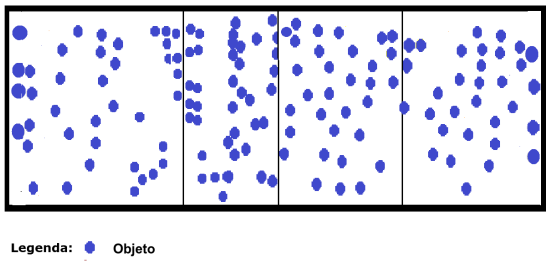
O modelo MapReduce a ser proposto como solução deste problema é adequado para ser executado no framework de processamento distribuído Apache Spark[1]. Spark (ZAHARIA et al., 2012) é um cluster framework que realiza o processamento em memória com o objetivo de superar os sistemas baseados em disco, como Hadoop [2]. Damos o nome de *ParallelNACluster* à solução proposta nos próximos parágrafos. Resumidamente, ela é dividida em quatro etapas principais, podemos acompanhá-las pelas Figuras 6.2 e 6.3:

- **Particionamento** - fase em que os dados são particionados de acordo com os critérios apresentados anteriormente nos desafios da paralelização. O particionamento produz partições balanceadas e as fronteiras são conhecidas. A Figura 6.2b ilustra o particionamento a partir dos dados da Figura 6.2a.
- **Map** - etapa em que executamos independentemente o NACluster em cada partição. Em seguida, identificamos os clusters que contêm objetos influenciados por cada fronteira, com o objetivo de mapear esses objetos para uma nova partição. A Figura 6.2c ilustra os não influenciados pela fronteira (os azuis) e influenciados pela fronteira (os demais). Mais detalhes serão apresentados nas próximas seções. Os clusters não influenciados pela fronteira formados em cada partição, depois da execução do NACluster, são guardados para serem utilizados na última etapa (União).

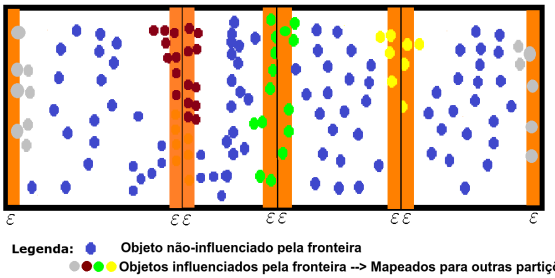
[1] http://spark.apache.org/
[2] http://hadoop.apache.org/



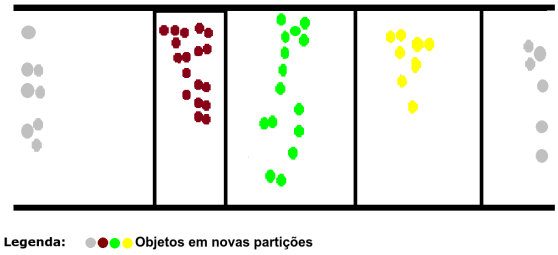
(a) Dados Originais



(b) Particionamento



(c) **Map** - NACluster aplicado em cada partição e os objetos influenciados pela fronteira são mapeados para uma nova partição e enviados ao Reduce



(d) Reduce - NACluster em cada nova partição

Figura 6.2: Etapas do Particionamento ao Reduce

- **Reduce** - passo em que efetuamos o NACluster nas novas partições geradas, as quais contêm os objetos influenciados por cada fronteira. Esses objetos precisam ser executados juntos para obedecer os critérios de proximidade exigidos na clusterização do NACluster. A Figura 6.2d ilustra as novas partições e seus objetos.

- **União** - fase em que unimos os clusters produzidos nas etapas Map e Reduce e geramos o resultado final. Na Figura 6.3 o cilindro azul representa os clusters que não recebem influencia da fronteira, gerados pelo NACluster aplicado em cada partição na etapa Map. O cilindro vermelho ilustra os clusters gerados na etapa Reduce e que têm influencia da fronteira. E o cilindro preto representa o resultado final, ou seja, a união dos resultados das duas etapas anteriores.

O ParallelNACluster é uma solução distribuída que pode ser utilizada por toda a comunidade através do uso de máquinas baratas. O particionamento permite que sejam utilizados *datasets* de tamanho na ordem de bilhões de objetos, tornando nossa proposta uma solução atual e robusta, que funciona mesmo com o grande aumento do volume de dados, bastando apenas aumentar o número de partições, utilizando o mesmo *cluster*, à medida que aumenta o tamanho dos dados

As próximas subseções identificam as principais características que norteiam as soluções em cada etapa.

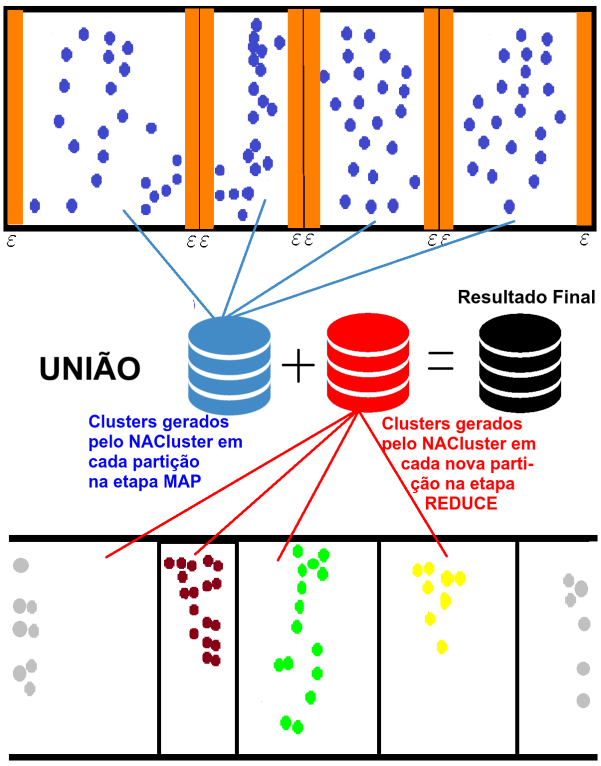


Figura 6.3: Etapa União

6.1.2 Particionamento

Desejamos realizar um particionamento do *dataset* de maneira que os objetos vizinhos sejam associados à mesma partição lógica. Além disso, as partições precisam ser balanceadas, ou seja, cada partição deve ter um número semelhante de objetos.

6.1.3 Formalização do Problema

A proposta do particionamento para este trabalho vem de (GASPAR; PORTO, 2014). Em (GASPAR; PORTO, 2014), propõe-se o FRANCE (FRAgmeNtador de Catálogos Espaciais), um algoritmo iterativo para particionar dados em histogramas *equi-depth*. Formalizamos o particionamento através da Definição 10.

Definição 10. *(Particionamento) Dado um objeto* $o_{ji} < x, y, a_1, a_2, ..., a_t >$ *tal que* a_z, $1 \leq z \leq t$, *é um atributo de* o_{ji}, *e* $x \in X$ *e* $y \in Y$, *tal que* X *e* Y *são as dimensões espaciais; um particionamento* P *é uma lista de valores* $< v_1, v_2, ..., v_g >$ *tal que* $v_h \in x$ *e* $1 \leq h \leq g$. *Desta forma, uma partição* P_r, *tal que* $1 \leq r \leq g-2$ *apresenta objetos vizinhos em uma região do espaço delimitada por* P_r *e* P_{r+1}.

6.1.4 Implementação

O FRANCE calcula iterativamente os pontos de fragmentação segundo uma das dimensões espaciais, RA, se preocupando em manter as partições com uma quantidade equivalente de elementos (dado um δ), mesmo em catálogos com densidade não uniforme (como o Espaço).

As partições são geradas iterativamente até que a quantidade de elementos seja próxima em δ do balanceamento perfeito. Consideramos perfeito, balanceamentos em que as partições apresentam n/g elementos, onde n é o total de elementos e g a quantidade de partições.

O algoritmo tem como entrada o mesmo tipo de *dataset* usado no NACluster. Primeiramente, divide-se esse *dataset* em duas partições, cada uma cobrindo metade dos dados. Uma variável *diff* é definida como metade do tamanho de cada partição. Para cada partição, enquanto não atingir um particionamento com n/g objetos (com uma tolerância de δ definida a priori), ela será reduzida por $diff$. Para cada passo, $diff = diff/2$. Quando a partição tiver a quantidade de objetos dentro do limite de δ, ela será fixada. O algoritmo executa recursivamente no espaço disponível entre as partições fixas.

O valor de δ é alterável no código e, no momento, no FRANCE, ele se apresenta como 0,5% da quantidade de objetos desejada. Logo, todas as partições geradas terão $(n/g) \pm 0,005 \times (n/g)$ elementos. Com esse δ em 0,5% temos uma variação muito pequena no tamanho das partições.

O FRANCE gera como saída os valores da coordenada x onde ocorreram as divisões do espaço. A partir desses valores, podemos definir formalmente uma fronteira através da Definição 11.

Definição 11. *Uma fronteira f_i definida segundo um valor de particionamento v_i pertencente a P, contem um conjunto de objetos O_{f_i} cujos valores da coordenada x de particionamento estão entre $v_i - \varepsilon \leq x \leq v_i + \varepsilon$.*

O pseudocódigo é apresentado no Algoritmo 3 e o seu passo a passo é ilustrado na Figura 6.4, onde os retângulos vermelhos correspondem às partições com uma quantidade de elementos maior que $n/x + \delta$ e que ainda precisam ser divididas para conter o número de objetos ideal. Os retângulos verdes representam as partições que contêm a quantidade de objetos dentro do limite de $n/x + \delta$ e não precisam mais ser subdivididas, ou seja, elas são partições fixadas. Para maiores detalhes, a implementação do FRANCE que fizemos em java está disponível à comunidade na

internet[3].

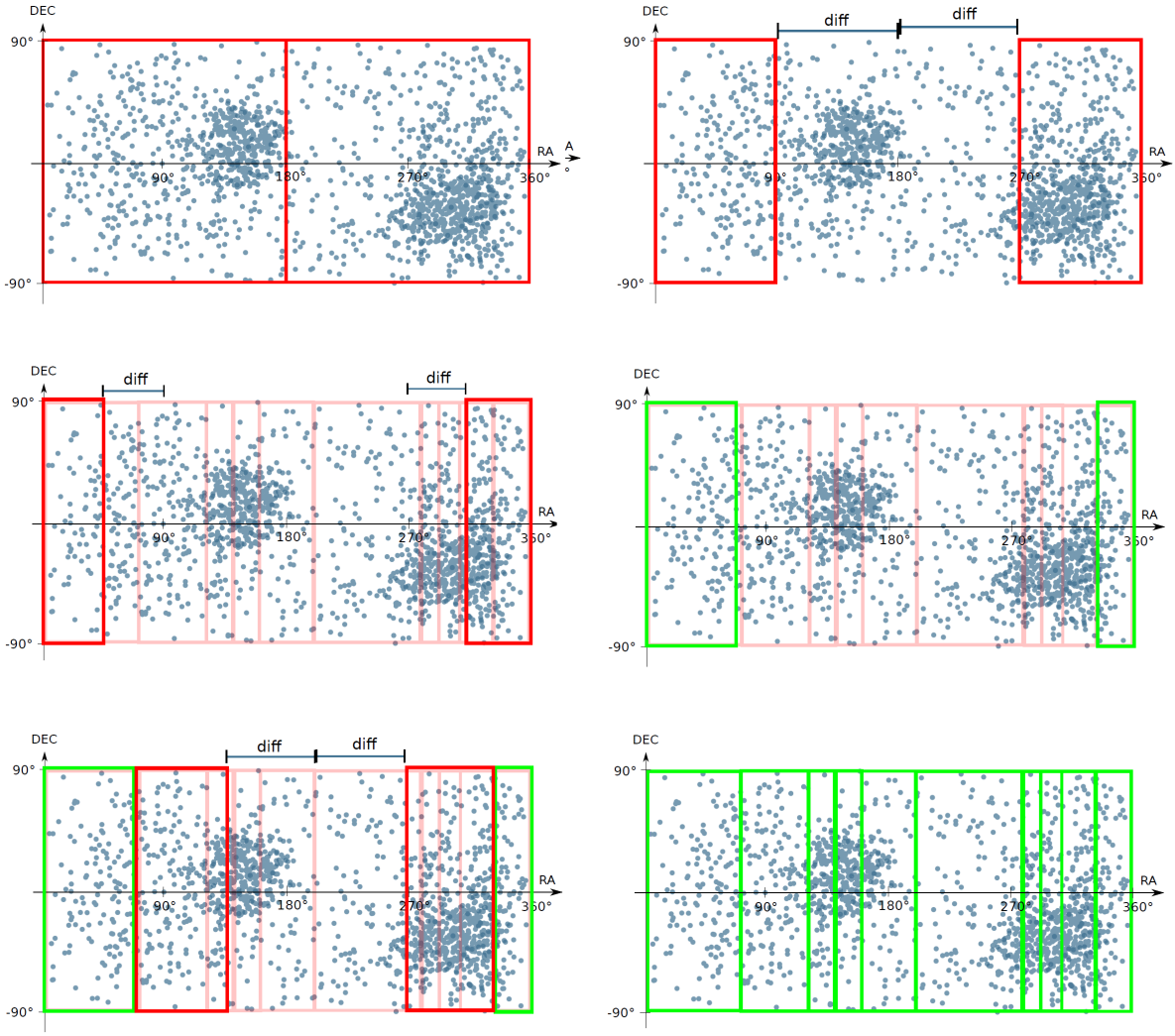


Figura 6.4: Passo a passo do FRANCE

O software FRANCE deve ser executado anteriormente ao ParallelNACluster e sua saída é um arquivo contendo os valores em *RA* correspondentes às divisões feitas no espaço. Esse arquivo é uma das entradas de nossa implementação paralela. É importante ressaltar que como o FRANCE precisa da visão completa dos dados, não pode ser paralelizado e executa de forma centralizada em apenas uma máquina.

6.1.5 Map

Uma vez que as fronteiras tenham sido definidas, podem-se gerar as partições de dados a partir das associação de cada objeto do catálogo a um intervalo f_i e f_{i+1}. As partições assim criadas podem ser distribuídas pelos nós do cluster para

[3]http://github.com/vinipires/FRANCE

Algoritmo 3 FRANCE

1: **Entrada 1**: *Dataset* contendo um conjunto de catálogos
2: **Entrada 2**: Quantidade de partições
3: **procedure** FRANCE($dataset$,x) ▷ x é a quantidade partições desejada
4: $partitions \leftarrow$ *dataset*$/2$ ▷ Duas partições de mesmo tamanho
5: $n \leftarrow$quantidade de objetos do *dataset*
6: $diff \leftarrow |partitions[0]/2|$
7: **for all** $partition \in partitions$ **do**
8: **while** existir mais que $(n/g) \pm \delta \times (n/g)$ objetos **do** ▷ $\delta = 0{,}5\%$
9: Partição é reduzida por $diff$
10: $diff \leftarrow diff/2$
11: **end while**
12: **end for**
13: **end procedure**
14: **Saída**: fronteiras;

processamento. Assim, o NACluster pode ser executado em cada partição de forma isolada. A estratégia de processar isoladamente funciona para o NACluster, pois sua operação consiste em comparar cada objeto com possíveis centroides à uma distância ε. Portanto, é um processamento de vizinhança, e enquanto os vizinhos estiverem na mesma partição, o algoritmo tem um comportamento semelhante ao centralizado. A diferença aparece quando o vizinho está em outra partição.

Uma abordagem ingênua de paralelização seria aplicar o NACluster em cada partição e ao final do processamento unir o resultado. No entanto, existe um problema nessa abordagem, pois como cada partição seria executada isoladamente, a clusterização dos objetos da fronteira não levaria em conta os vizinhos da partição ao lado, mesmo que estivessem a menos que ε de distância. Isso faria com que ao final da execução paralela, usando essa abordagem, os clusters formados próximos de cada fronteira não reproduzissem a característica de casar por proximidade todos os dados, tal como faz o NACluster centralizado. Portanto, concluímos que os casamentos próximos à fronteira devem ser tratados de forma que sejam realizados considerando os objetos vizinhos dos dois lados da divisão ao mesmo tempo.

O problema da fronteira pode ser enunciado da seguinte forma: dadas duas partições vizinhas P_i e P_j, uma fronteira f_k, $f_k \subset P_i$ e $f_k \subset P_j$, e dois objetos $o_i \in Pi$, e $o_j \in P_j$, se distância$(o_i, o_j) < \varepsilon$ então o NACluster deve considerar f_k tal que $f_k \supset \{o_i, o_j\}$. Desta forma, para tratar o problema da fronteira, criamos a partir de P_i e P_j a fronteira f_k para processamento.

Como o particionamento do FRANCE é feito em apenas uma dimensão, existe somente uma fronteira entre duas partições vizinhas. A fronteira é ilustrada na Figura 6.5. Como a região onde pode aparecer ambiguidade na nossa definição de casamento está à uma distância ε do ponto referencial, e considerando que esse referencial é a fronteira, todos os elementos pertencentes à região distante ε dela

precisam ser tratados nas duas partições. Na Figura 6.5, essa região está destacada.

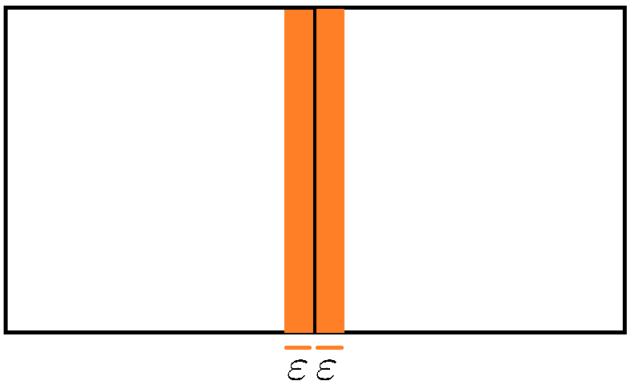


Figura 6.5: Fronteira entre duas partições

Definimos abaixo os passos para nossa estratégia de paralelização:

1. Aplicar o FRANCE no *dataset* de entrada e definir as fronteiras.
2. Realizar o particionamento obedecendo as fronteiras definidas anteriormente e mapear os dados à sua respectiva partição.
3. Executar o NACluster sobre cada partição.
4. Identificar os clusters pertencentes à região de cada fronteira.
5. Tratar todas as fronteira, de forma a detectar todos os clusters formados por objetos dessa região, bem como aqueles nos quais seus casamentos poderiam ser influenciados pelos objetos da fronteira.
6. Para cada fronteira, enviar os clusters influenciados por ela (dos seus dois lados) à uma nova partição.
7. Os demais clusters de fora da fronteira devem ficar armazenados para uso no último passo.
8. Realizar o NACluster em cada uma das novas partições que contém os clusters influenciados.
9. Unir o resultado da clusterização em cada uma das partições.

Os clusters influenciados pela fronteira são aqueles cuja definição é influenciada pelos objetos na fronteira. Definimos objeto da fronteira, objeto influenciados pela fronteira, bem como cluster influenciado pela fronteira nas Definições 12, 13, 14, respectivamente.

Definição 12. *(Objeto da Fronteira) Seja O_{f_i} uma fronteira, um objeto da fronteira é o_l tal que $o_l \in O_{f_i}$.*

Definição 13. *(Objeto influenciado pela fronteira) Seja uma entidade $e_p \subset O_{f_i}$ e um objeto $o_l \in O$ e $o_l \notin O_{f_i}$. o_l é influenciado pela fronteira $O_{f_i} \iff d(o_l, e_p) \leq \varepsilon$, onde d é a função de distância definida na Definição 3.*

Definição 14. *(Cluster influenciado pela Fronteira) Seja o_l um objeto influenciado pela fronteira O_{f_i}. Uma entidade e_j é um cluster influenciado pela fronteira O_{f_i} se $o_l \in e_j$.*

6.1.5.1 SCIBoundary: Identificando os clusters influenciados pela fronteira

Ao algoritmo de tratamento da fronteira, cujo objetivo é selecionar os clusters influenciados por ela, damos o nome de SCIBoundary, um acrônimo para **S**elect **C**lusters **I**nfluenced by **B**oundary.

Vamos expor a intuição de como podemos visualizar o tratamento das fronteiras. Essa visualização é mostrada através das componentes conexas de um grafo. Elas revelam existência de influência dos objetos da fronteira na formação dos clusters próximos à eles.

Definição 15. *Grafo de Clusters (GC): Um GC é um grafo $G = (V, E)$ de tal forma que V é um conjunto de vértices, e cada vértice $v \in V$ corresponde a um cluster, e as arestas $e \in E$, aos pares de clusters. Em um GC, uma aresta existe entre dois vértices v_i e v_j se existe um objeto $o_i \in v_i$ tal que a distancia$(o_i, v_j) < \varepsilon$, onde v_j é representado por seu centroide no cálculo da distância.*

As componentes conexas são quaisquer subgrafos conexos maximais de G, mais precisamente, são os maiores conjuntos de nós, tal que todos os nós conseguem alcançar os demais. Dado um grafo GC, formado a partir do resultado da execução do NACluster em cada partição, podemos observar todas as componentes conexas que têm origem na fronteira e detectar todos os clusters que tem influência da fronteira na sua formação.

Na Figura 6.6 temos essa representação. Caso processemos, juntos, todos os clusters que formam as componentes conexas que tem origem na fronteira de duas partições vizinhas em uma segunda etapa resolveremos o problema da fronteira.

A estratégia de processar as componentes conexas que têm origem na fronteira é viável, pois ela é uma região muito pequena (enquanto uma partição possui graus de extensão, uma fronteira é em torno de 5 ordens de grandeza menor) e os catálogos com os quais temos trabalhado não são densos a ponto de ter uma quantidade maior de objetos próximos à fronteira do que distribuídos no restante do catálogo.

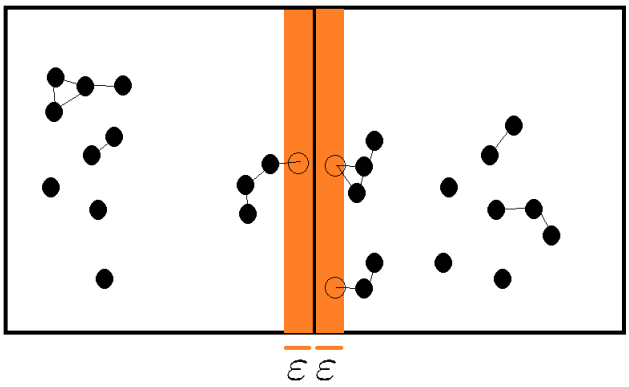


Figura 6.6: Visualização do grafo formado por clusters vizinhos depois da execução do NACluster em cada partição.

Para executar o SCIBoundary, devemos considerar que as partições e as fronteiras são nomeadas conforme mostrado na Figura 6.7. Os nomes das partições que representam os objetos fora da fronteira são formados por Px, onde x corresponde ao número da partição. Os nomes das fronteiras (em azul na Figura 6.7) são formados pelo nome da partição da esquerda unido com o nome daquela que fica à direita. Por exemplo, a fronteira que fica entre as partições $P1$ e $P2$ recebe o nome de $p1p2$.

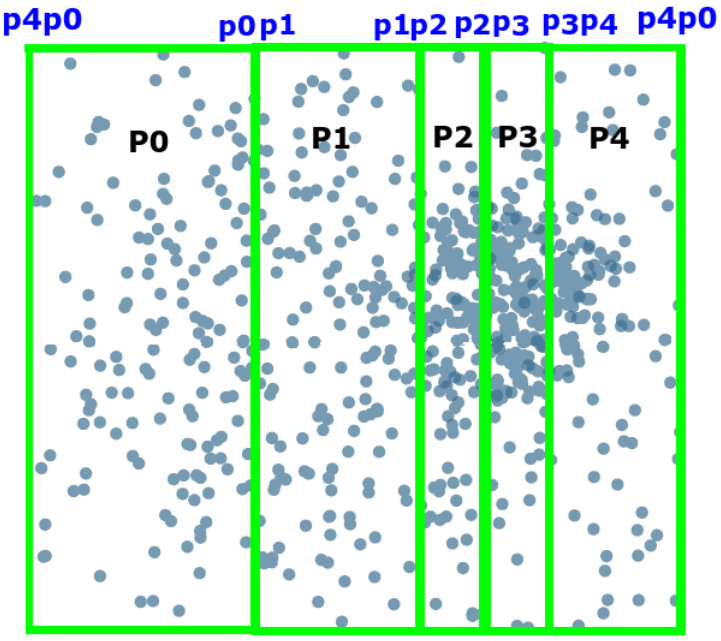


Figura 6.7: Nome das partições e fronteiras após o particionamento.

Para facilitar o entendimento do SCIBoundary, apresentamos os atributos dos objetos e clusters através da Figura 6.8, os quais *coordenadas*, *idObjeto*, *idCatalogo* e *centroide* são autoexplicativos e o atributo *boundary* é um tipo booleano, onde seu valor *true* indica que é um objeto, ou cluster, influenciado pela fronteira, e *false*

indica o contrário. O atributo *partitionName* serve para guardar o nome da partição ao qual o objeto, ou o cluster, pertence.

Antes da execução do SCIBoundary é necessário que nossa versão paralela realize os seguintes procedimentos:

1. Cada objeto do conjunto de catálogos deve ter o atributo *boundary* inicializado com *false* e o (*partitionName*) com o nome da partição ao qual ele pertence (nesse momento todos os objetos são considerados de fora da fronteira).

2. Como são conhecidas as fronteiras definidas pelo FRANCE, para cada partição deve-se identificar e marcar *true* no atributo *boundary* dos objetos presentes entre a posição de cada fronteira até uma distância ε dela. O valor do atributo *partitionName* destes objetos deve ser alterado para o nome da fronteira, mapeando, assim, o objeto da fronteira para uma nova partição.

3. Realizar o NACluster em cada partição. Obs: até este momento, o atributo *boundary* dos clusters formados deve possuir o valor *false*.

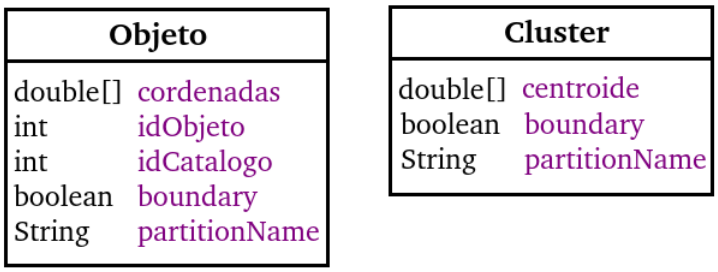


Figura 6.8: Atributos dos Objetos e Clusters

O SCIBoundary recebe como entrada uma lista de objetos vizinhos da fronteira. A implementação contém recursividade e seu pseudocódigo é apresentado pelo Algoritmo 4. Resumidamente, ela é marcada pelos seguintes passos:

- **Passo 1** - Para cada objeto da lista, se o objeto pertencer à região da fronteira, buscar clusters (centroides) em um raio ε. Caso contrário, marcá-lo como objeto da fronteira e então realizar a busca por clusters.

- **Passo 2** - Para cada cluster encontrado, caso ele não pertença à região da fronteira ou não seja influenciado por ela, sua variável *boundary* recebe *true*, ou seja, ele é marcado como um cluster influenciado pela fronteira e mapeado para a partição com o mesmo nome da fronteira. Os objetos deste cluster são

Algoritmo 4 SCIBoundary

1: **Input**: $objList$ lista de objetos (objetos podem ou não pertencer à fronteira), $partitionName$ (nome da partição relativa à fronteira) e $indexedCentroids$, a variável indexada de centroides (com seus respectivos clusters).
2: **Output**: a variável indexada de centroides atualizada ($indexedCentroids$)
3: **function** SCIBOUNDARY($objList$,$partitionName$,$indexedCentroids$)
4: **for all** $obj \in objList$ **do**
5: **if** $obj.boundary == true$ **then** ▷ Objeto da fronteira.
6: $clustersList \leftarrow$ rangeQuery(obj,$indexedCentroids$,ε); ▷ Busca indexada por centroides em raio ε a partir de obj. Retorna a lista de clusters presentes no raio.
7: **else** ▷ Objeto não pertence à fronteira.
8: $obj.boundary \leftarrow true$
9: $obj.partitionName \leftarrow partitionName$ ▷ Objeto influenciado pela fronteira é mapeado para a nova partição.
10: $clustersList \leftarrow$ rangeQuery(obj,$indexedCentroids$,ε); ▷ Busca indexada por centroides em raio ε a partir de obj. Retorna a lista de clusters presentes no raio.
11: **end if**
12: **for all** $cluster \in clustersList$ **do**
13: **if** $cluster.boundary == false$ **then**
14: $cluster.boundary \leftarrow true$
15: $cluster.partitionName \leftarrow obj.partitionName$ ▷ Cluster influenciado pela fronteira é mapeado para a nova partição
16: $objsInCluster \leftarrow cluster.getObjects()$ ▷ Retorna objetos pertencentes ao cluster
17: $indexedCentroids \leftarrow$ SCIBoundary($objsInCluster$,$cluster.partitionName$,$indexedCentroids$);
18: **end if**
19: **end for**
20: **end for**
21: **Return** $indexedCentroids$;
22: **end function**

recuperados e colocados em uma lista. Essa lista é usada como entrada em uma nova chamada do SCIBoundary e deve-se voltar ao passo 1.

O passo a passo do SCIBoundary é apresentado na Figura 6.9. Podemos examiná-la juntamente com os passos descritos acima. As bolas azuis representam os objetos, os retângulos vermelhos equivalem aos clusters (centroides) da fronteira e influenciados por ela, e os retângulos pretos correspondem aos clusters (centroides) fora da fronteira ou não influenciados por ela. Podemos observar que no momento "a" da Figura 6.9, possuímos os clusters (em preto) formados pelo NACluster, classificados inicialmente como fora da fronteira, e um objeto (em azul) pertencente à ela. Todos eles presentes na partição *P1*. No momento "b", o objeto procura por centroides de clusters em um raio ε e encontra dois deles. Estes dois clusters passam a ter a cor vermelha no instante "c", ou seja, passam a ser considerados como clusters influenciados pela fronteira e são mapeados para partição *p1p2*, e em "d" a componente conexa começa a se formar. No momento "e", os objetos do cluster encontrado são recuperados e inicia-se uma busca por outros clusters (centroides) fora da fronteira em "f". No instante "g", o novo cluster encontrado transforma-se em cluster influenciado pela fronteira, tornando-se vermelho. De "h" até "n", os passos são semelhantes aos ocorridos anteriormente. Ao final da execução do SCIBoundary em "n", todos os clusters encontrados a partir da fronteira ficaram vermelhos e foram mapeados para a partição de mesmo nome da fronteira *p1p2*, mostrando que eles são influenciados por ela.

Ao final da execução do SCIBoundary em partições vizinhas teremos dois conjuntos de dados:

- **A**: contendo os clusters que não são afetados pela fronteira, representados pelas bolas azuis na Figura 6.10.
- **B**: contendo os clusters afetados pela fronteira, representados pelas bolas vermelhas na Figura 6.10.

Os clusters do conjunto **A** devem ser guardados para serem usados no resultado final e os clusters do conjunto **B** devem ser enviados para a etapa Reduce.

Um exemplo ilustrativo da classificação feita pelo SCIBoundary em todo o *dataset* é apresentado na Figura 6.11. Neste exemplo, o *dataset* possui 4 partições inicialmente. Ao final do SCIBoundary, o *dataset* passa a ter 8 partições, as 4 já existentes anteriormente ($P1$, $P2$,$P3$ e $P4$) e as 4 novas partições correspondentes às fronteiras ($p1p2$, $p2p3$, $p3p4$ e $p4p1$). A cor das bolas próximas às fronteiras representa a partição a qual os clusters pertencem.

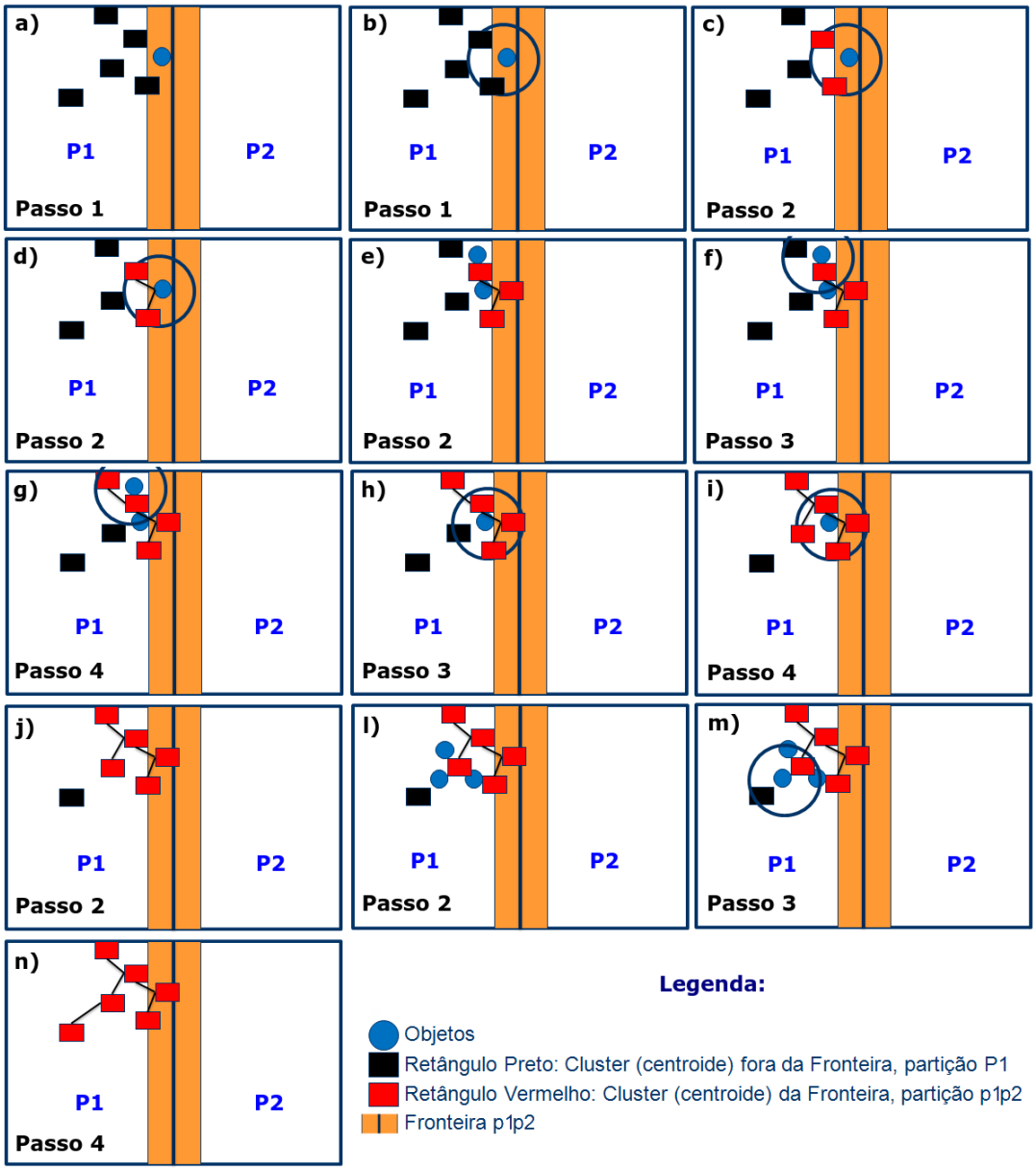


Figura 6.9: Passo a passo do SCIBoundary

6.1.6 Reduce

As novas partições criadas pelo SCIBoundary contendo os clusters influenciados pela fronteira são processadas na etapa Reduce. A Figura 6.12 ilustra as partições formadas. Cada bola representa um cluster influenciado pela fronteira e sua cor representa sua partição. Cada elipse expõe os clusters que devem ser reexecutados, desta vez juntos na mesma partição.

Realizar o NACluster nas fronteiras é preciso para corrigir os casamentos dos objetos influenciados por elas na etapa anterior. A influência da fronteira faz com que o casamento ocorra, na etapa Map, sem considerar os vizinhos da partição ao

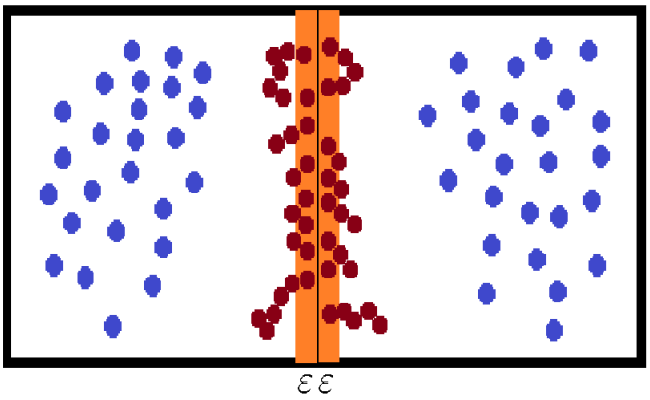


Figura 6.10: Visualização de duas partições vizinhas após a execução do SCIBoundary.

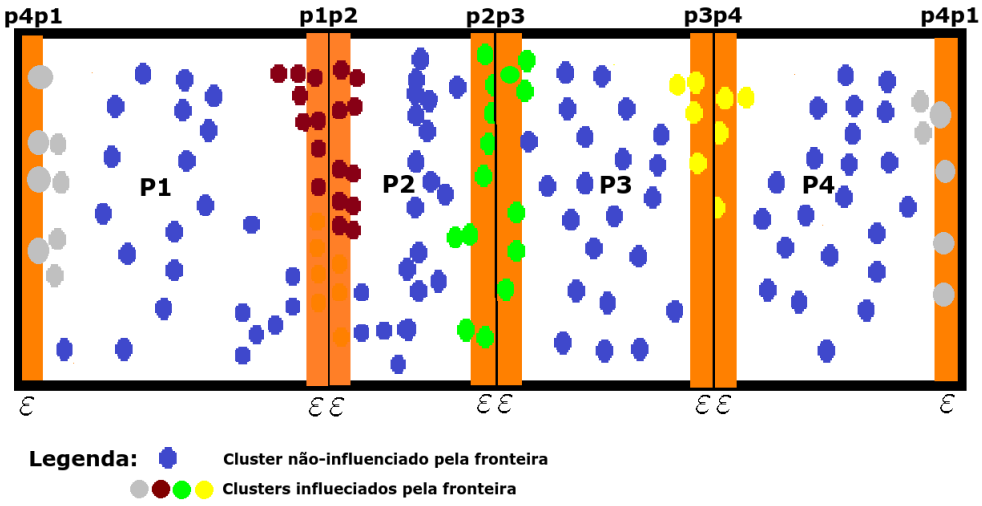


Figura 6.11: Exemplo ilustrativo dos clusters em todo o dataset após a execução do SCIBoundary.

lado.

O Reduce recebe como entrada o par <chave,valor>, onde a chave é o *id* da nova partição e o valor é a lista dos clusters dessa partição. Para cada partição, são recuperados os objetos pertencentes à todos os seus clusters e é executado o NACluster sobre esses objetos, produzindo, assim, um novo casamento.

Uma observação acerca desta etapa é que apesar dos objetos das fronteiras de *RA* 0.0 e 360.0 executarem juntos no Reduce, eles não estão sendo adaptados para corrigir seus clusters. A implementação foi feita dessa forma para facilitar o tratamento dessa fronteira em trabalhos futuros.

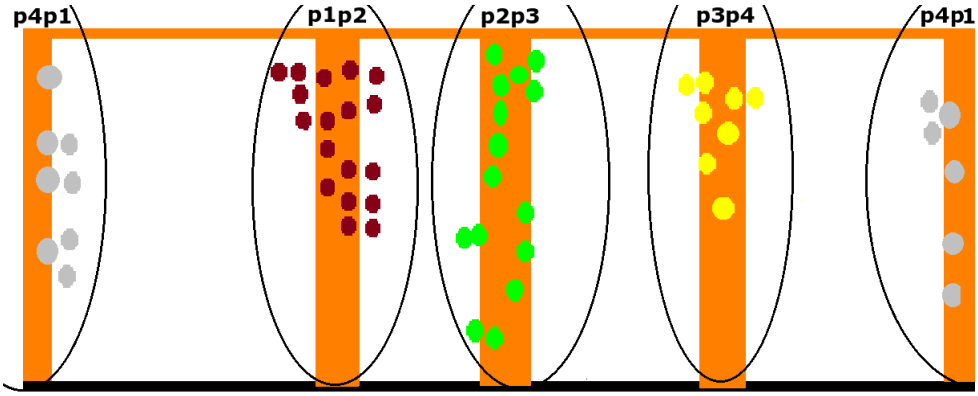


Figura 6.12: Exemplo ilustrativo dos clusters em todo o dataset após a execução do SCIBoundary.

6.1.7 União

O resultado final do ParallelNACluster será a união dos:

- clusters não influenciados pela fronteira, armazenados previamente na etapa *Map*;
- com os novos clusters produzidos pela etapa Reduce.

A conclusão dessa etapa garante que o resultado do ParallelNACluster seja semelhante ao resultado do NACluster Centralizado aplicado ao mesmo *dataset*.

6.2 Implementação

Nesta seção apresentamos os detalhes da implementação para Apache Spark do ParallelNACluster e da implementação do workflow que executa o processo completo desde o pré-processamento dos dados em disco local até a execução do ParallelNACluster em ambiente distribuído.

6.2.1 ParallelNACluster

A implementação do algoritmo paralelo possui código aberto e está disponível à comunidade no endereço https://github.com/vinipires/ParallelNACluster. A Figura 6.13 apresenta os detalhes da implementação. Os RDDs são apresentados em amarelo, e os retângulos azuis representam as transformações spark. Essas transformações recebem um RDD como entrada e gera um novo RDD na saída. As setas

estão de acordo com as entradas e saídas de cada transformação e obedecem a sequência de execução.

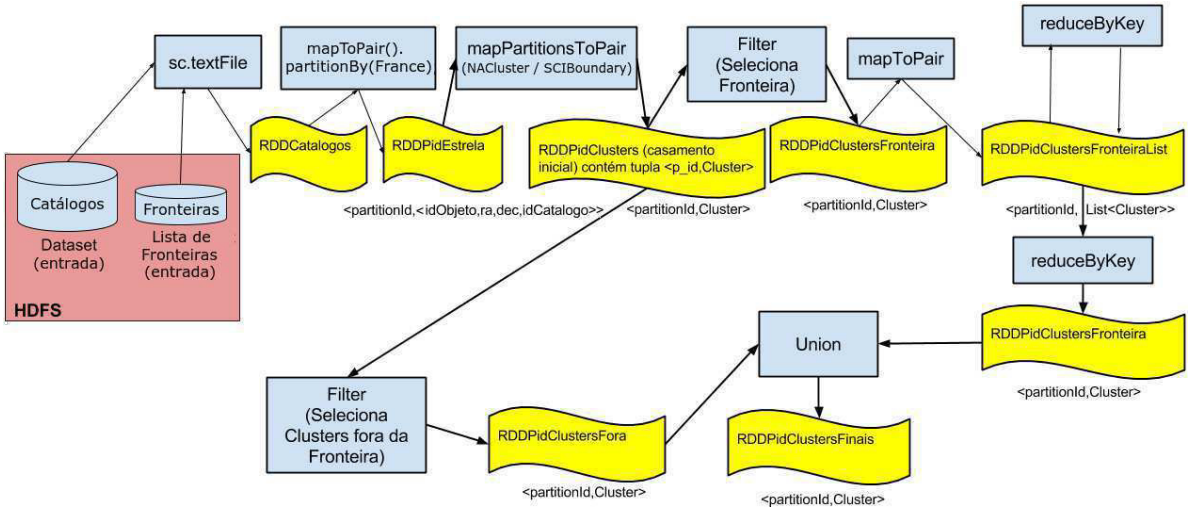


Figura 6.13: Modelo de Implementação do ParallelNACluster para Apache Spark

No primeiro momento, os dados dos catálogos são lidos do HDFS e carregados no *RDDCatalogos*. Este RDD contém strings formadas pelo id do objeto, sua posição no espaço (RA e DEC) e o id do catálogo. A lista de fronteiras, também lida do HDFS, é carregada em uma *broadcast variable*[4] para ficar acessível à todo o código. Com o acesso às fronteiras, ou seja, aos valores em RA correspondentes aos locais onde o *dataset* foi dividido, é possível realizar o particionamento do RDD. Portanto, uma transformação *mapToPair*, seguida do *partitionBy*, é aplicada ao RDDCatalogos. Essa transformação gera o RDDPidEstrela contendo um par <chave,valor>, onde a chave é o *id* da partição e o valor é a informação do objeto (*idObjeto,ra,dec,idCatalogo*). O *partitionBy* implementa o FrancePartitioner, criado para particionar os dados de acordo com as fronteiras definidas no arquivo de Fronteiras.

A transformação *mapPartitionsToPair* realiza, em cada partição, o casamento do NACluster e o SCIBoundary. O NACluster gera os clusters e o SCIBoundary identifica aqueles influenciados pela fronteira e os mapeia para a partição correspondente à ela, conforme detalhado na subseção 6.1.5.1. Essa transformação gera o RDDPidClusters, onde cada um dos seus elementos é da forma <chave,valor>. A chave é o id da partição e o valor é cluster formado pelo casamento. É importante ressaltar que esse RDD tem o dobro do número de chaves do anterior, consequentemente o dobro do número de partições, pois além das partições que existiam antes,

[4]Broadcast Variable em Spark permite ao programador manter uma variável somente leitura em cache em cada máquina ao invés de enviar uma cópia do mesmo juntamente com as tarefas.

passou-se a considerar as novas partições correspondentes à cada fronteira.

Com a execução do SCIBoundary, temos dois tipos de clusters no RDD-PidClusters: aqueles influenciados pela fronteira e os não influenciados por ela. Duas transformações do tipo *Filter* são aplicadas à esse RDD para dividi-lo em dois. A primeira operação *Filter* seleciona os clusters que são influenciados pela fronteira e gera o *RDDPidClustersFronteira*. O RDDPidClustersFronteira tem o mesmo formato do RDD anterior, ou seja, é um par <chave,valor> do tipo <partitionId,Cluster>. A segunda operação *Filter* aplicada ao *RDDPidClusters* seleciona os clusters que não são influenciados pela fronteira e gera o *RDDPidClustersFora*, também do tipo <partitionId,Cluster>. É importante ressaltar que este último já contém o casamento correto segundo os critérios do NACluster, pois seu casamento não sofre influência de nenhuma das fronteiras. Portanto, esse RDD será usado somente depois do processamento da fronteira.

O *RDDPidClustersFronteira* precisa se adequar à etapa Reduce, sua estrutura definida por <partitionId,Cluster> precisa ser modificada para < partitionId, List<Cluster> >, ou seja, para cada partição, precisamos ter a lista de clusters associados. Com o intuito de transformar esse RDD na forma desejada, a operação *mapToPair* é aplicada e cria-se o *RDDPidClustersFronteiraList* que contém as tuplas do tipo < Particao, List<Cluster> >. Neste primeiro momento, cada List<Cluster> contém apenas um elemento, ou seja, para cada tupla <Particao,Cluster> existente anteriormente cria-se uma lista com apenas esse cluster para que depois seja possível fazer o reduceByKey e criar a tupla < Particao, List<Cluster> >, onde cada chave contém a lista de todos os clusters da partição correspondente.

Preparamos o *RDDPidClustersFronteiraList* para o formato desejado através do primeiro *reduceByKey*. Portanto, podemos implementar a segunda transformação *reduceByKey* com a finalidade de executar o NACluster em cada uma das fronteiras e corrigir os clusters formados anteriormente. Essa operação sobrescreve o RDDPidClustersFronteira.

Para finalizar o ParallelNACluster, realizamos a transformação *Union* que retorna a união de RDDPidClustersFronteira e RDDPidClustersFora. Dessa forma, concluímos todo o processo de clusterização.

6.2.2 AODP: Um Data Flow de particionamento

O ParallelNACluster depende do resultado da execução do FRANCE. A Figura 6.14 ilustra essa dependência e mostra o FRANCE recebendo como entrada o conjunto de catálogos e o número de partições desejado. A saída produzida por

ele contém um arquivo com as quatro fronteiras geradas que dão origem às cinco partições. Esse arquivo, juntamente com o conjunto de catálogos, são usados como entrada no ParallelNACluster.

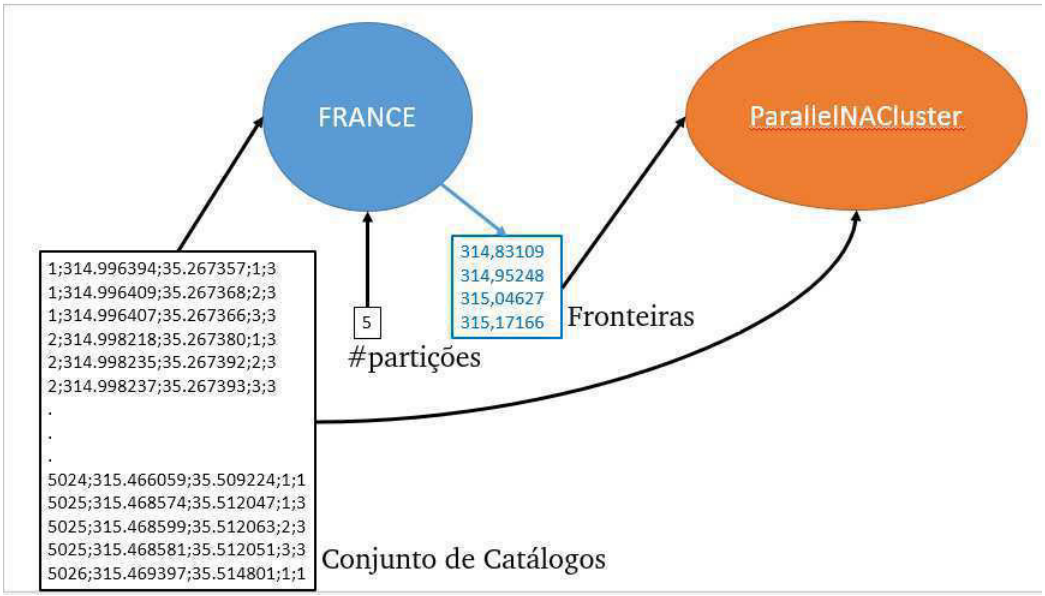


Figura 6.14: Modelo de Implementação do ParallelNACluster para Apache Spark

O FRANCE executa de forma centralizada, logo o conjunto de catálogos usado como entrada precisa estar no disco local da máquina onde ele é executado. No entanto, o ParallelNACluster executa de forma distribuída e seus arquivos de entrada, precisam estar em um sistema de arquivos distribuído, como o HDFS. Portanto, para executar o ParallelNACluster é necessário realizar uma sequência de passos anteriores à sua execução. Eles são:

- tornar conhecidas as fronteiras entre as partições: para identificar as fronteiras em um conjunto de catálogos é preciso realizar seu particionamento com o software que implementa o FRANCE e executa de forma centralizada. A sua saída deve ser usada como entrada do ParallelNACluster;
- copiar os arquivos de entrada do ParallelNACluster para o sistema de arquivos distribuído: o conjunto de catálogos e o resultado do FRANCE deve ser copiado para o HDFS.

Esse pré-processamento executado manualmente requer tempo e atenção na manipulação dos arquivos. Qualquer erro compromete o resultado final do casamento, pois caso usemos o arquivo de fronteiras que não se refira ao mesmo conjunto

de catálogos usado como entrada no ParallelNACluster, as partições ficarão desbalanceadas e pode ocorrer erro na alocação da memória. Portanto, precisamos de um processo automatizado que realize o pré-processamento dos dados e os direcione ao ParallelNACluster conforme mostrado na Figura 6.14.

Com o objetivo de resolver esse problema, desenvolvemos um *workflow*, chamado AODP (Application Oriented Data Partitioning), que organiza todas as etapas necessárias para a execução ParallelNACluster. Ele realiza o pré-processamento dos dados de diferentes catálogos, que estão em disco local, de forma que eles possam ser usados como entrada do ParallelNACluster em ambiente distribuído.

O AODP foi desenvolvido para executar através do QEF (PORTO et al., 2007), *Query Evaluation Framework* e está disponível à comunidade no endereço https://github.com/vinipires/AODP. O QEF é um framework que provê um ambiente para definição e execução de planos de consulta de forma distribuída, e permite a comunicação entre os componentes de execução de consulta e o acesso a fontes de dados heterogêneas. Os planos de consulta são representados por um QEP – *Query Execution Plan* – e constituídos de operadores que comunicam-se entre si para a obtenção de um resultado.

Basicamente, o workflow possui a estratégia FRANCE de particionamento, a cópia dos dados no HDFS e a execução do ParallelNACluster. Ele é apresentado graficamente pela Figura 6.15. Os círculos centrais representam cada um dos operadores: TupleScan, FRANCE, CopyToHdfs e ParallelNACluster.

O operador **TupleScan** é responsável por obter o *dataset* do disco local e o deixar disponível para os demais operadores.

O operador **FRANCE** é responsável pelo particionamento dos dados em histogramas *equi-depth*, conforme apresentado na subseção 6.1.2. Ele recebe como entrada o *dataset* formado por um conjunto de catálogos armazenado no disco local e gera como saída um arquivo contendo os intervalos em *RA* (Ascensão Reta) do particionamento, isto é, os valores em *RA* onde o espaço foi cortado. Como o FRANCE é uma visão dos dados, ele não pode ser paralelizado e executa localmente.

O terceiro operador, **CopyToHdfs**, recebe dois arquivos como entrada, o mesmo *dataset* e a lista de partições produzida pelo operador anterior. O objetivo do CopyToHdfs é copiar esses arquivos, que estão no disco local, para o HDFS e deixá-los disponíveis à todos os nós do cluster, tornando possível a execução da versão paralela do NACluster.

O último operador, **ParallelNACluster**, executa sobre o cluster Hadoop

YARN[5], uma plataforma de gerenciamento de recursos de computação em clusters responsável pelo agendamento de *jobs* no ambiente distribuído. O operador recebe como entrada o conjunto de catálogos e o arquivo contendo a lista de partições, todos eles armazenados no HDFS. Como saída, este operador gera o resultado do casamento produzido pela versão paralela do NACluster.

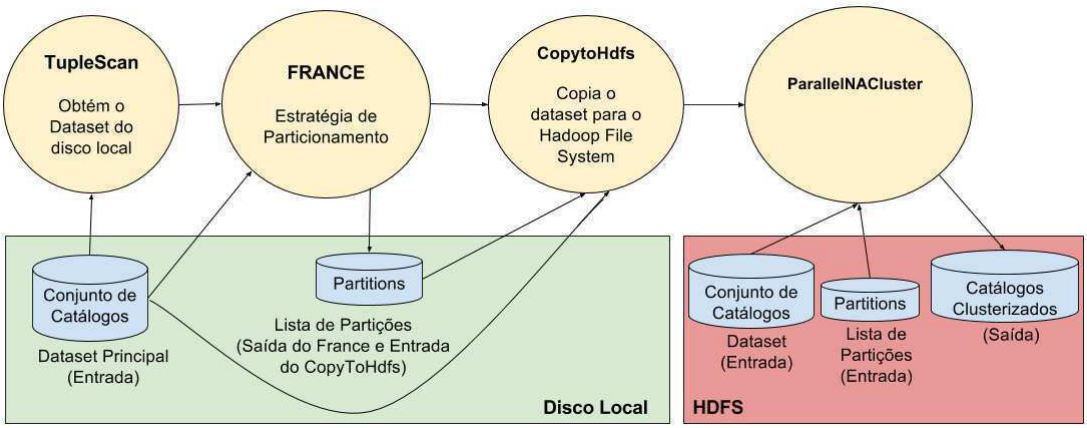


Figura 6.15: AODP: Um workflow de particionamento.

6.3 Considerações Finais

Iniciamos este capítulo demonstrando que em casos de grande volume de dados, o NACluster, apresentado no Capítulo 5, possui limitações físicas, pois precisaria de computadores com demasiada memória RAM para ser executado, algo que nos dias de hoje custa muito caro. Para baratear o processo e facilitar seu uso na comunidade que necessita processar grandes volumes de dados, propomos o ParallelNACluster, um algoritmo de clusterização que executa de forma distribuída e propõe-se a dar resultados semelhantes ao NACluster.

A solução paralela proposta requer alguns cuidados especiais. O primeiro deles, o particionamento, precisa preservar a vizinhança e manter o balanceamento das partições. Para atender este requisito, usamos o FRANCE (FRAgmeNtador de Catálogos Espaciais) para realizar o particionamento em uma dimensão. O segundo cuidado especial é com as fronteiras produzidas pelo particionamento. O NACluster precisa da vizinhança para realizar o casamento segundo seu algoritmo. No entanto, particionar os dados com o FRANCE, faz com que a vizinhança da fronteira de duas partições vizinhas seja separada. Portanto, conhecer as fronteiras e os objetos próximos em partições vizinhas torna-se essencial para a execução da clusterização nestes

[5]disponível em http://hadoop.apache.org/

objetos. Para esse tratamento especial das fronteiras propomos o SCIBoundary, um algoritmo recursivo que objetiva encontrar os objetos influenciados elas.

Para automatizar e acelerar a execução do pré-processamento dos dados, propomos a terceira contribuição deste capítulo, o *AODP*, um workflow para particionamento dos dados em disco local e execução do casamento dos mesmos em ambiente distribuído através do *ParallelNACluster*.

7 EXPERIMENTOS

Este capítulo apresenta os experimentos realizados nesta tese. Eles são divididos em duas partes. Primeiramente apresentamos os experimentos com o NACluster e em seguida com o ParallelNACluster.

Na Seção 7.1 discutimos como simulamos o *Golden Standard* e os conjuntos de catálogos a serem usados como entrada nas implementações. O objetivo das simulações é avaliar a qualidade do NACluster.

Em seguida, na Seção 7.2, analisamos os resultados dos experimentos do NACluster aplicados a diversos conjuntos de catálogos. No primeiro experimento, selecionamos três conjuntos de catálogos, aplicamos o NACluster sobre eles e medimos a qualidade dos casamentos através da precisão, abrangência e medida-F. Em seguida, na Subseção 7.2.1, propomos uma métrica para avaliar a ambiguidade do ponto de vista dos objetos, ou seja, o quanto cada objeto é ambíguo em relação aos clusters, com o objetivo de encontrarmos alguma correlação entre a qualidade da clusterização do NACluster e a ambiguidade existente no agrupamento. Na Subseção 7.2.2 realizamos experimentos para medir o tempo de execução do *NACluster* em relação ao aumento do volume do conjunto de catálogos. E na Subseção 7.2.3, realizamos uma análise para verificar se a escolha dos centroides iniciais influencia na saída do NACluster.

Na Seção 7.3 fazemos uma comparação entre um algoritmo de *cross-matching* bastante utilizado na comunidade, o *Q3C Join*, e o NACluster. E, finalmente, na Seção 7.4 apresentamos o último experimento com o NACluster e apontamos um caso particular que pode ocorrer na estratégia de casamento.

Na Seção 7.5 realizamos experimentos com o ParallelNACluster. Avaliamos a qualidade do seu casamento, bem como do casamento produzido pela fronteira. Além disso, analisamos o impacto da mudança dos parâmetros do Apache Spark na submissão da aplicação distribuída e testamos a escalabilidade ao aumentarmos o volume de dados.

E, finalmente, na Seção 7.6 fizemos testes com o AODP e discutimos a proporção de tempo que cada operador usa para executar e identificamos os parâmetros que influenciam diretamente nesta proporção.

7.1 Dados

O conjunto de catálogos S dado como entrada no NACluster aparece implementado em um aquivo no formato texto. Cada linha contém uma tupla com a forma *<idObjeto, ra, dec, idCatalogo>*, cujos elementos são autoexplicativos. Quaisquer catálogos de dados espaciais que sejam de mesma natureza podem ser casados pelo NACluster. Utilizamos os catálogos de astronomia em todos os nossos experimentos, pois os problemas dessa área, apresentados no Capitulo 3, nos motivaram a criar o algoritmo.

Conforme visto no Capítulo 5, durante a execução do NACluster, para cada objeto dos catálogos envolvidos, procura-se o cluster cuja distância entre seu centroide e o objeto seja menor que um valor dado, chamado de ε. Além disso, cada cluster deve representar um objeto espacial na região observada, segundo a Definição 1. Para constatar que cada agrupamento formado pelo NACluster é um objeto espacial, precisa-se de um conjunto de catálogos S que possua um *Golden Standard* do seu casamento, ou seja, um conjunto de catálogos no qual saibamos previamente os casamentos corretos. No entanto, não encontramos nenhum conjunto de catálogos com essa característica, dificultando, assim, medirmos a qualidade do algoritmo. Portanto, surgiu a necessidade de gerarmos catálogos sintéticos e os usarmos para criar um *Golden Standard* com o propósito de avaliar a qualidade do casamento.

Como visto no Capítulo 3, o casamento entre um par de catálogos astronômicos é feito, geralmente, através do uso das coordenadas *RA* e *DEC* dos objetos e de um raio ε. Por exemplo, supõe-se que exista um catálogo *A* e deseja-se realizar seu casamento com um catálogo *B*. Para cada objeto de *A*, procuram-se por objetos próximos em um raio ε. Se neste raio existir algum objeto pertencente ao catálogo *B* ocorre o casamento entre o objeto de *A* e de *B*. Em geral, o raio utilizado é de 1 arcseg e, ao realizar o casamento de um par de catálogos através esse raio, há uma grande chance da maioria dos casamentos estarem corretos.

Na Figura 7.1 apresentamos a distribuição das distâncias entre os objetos dos catálogos *2MASS* e *UCAC3* (descritos na Seção 3.3) que estão em uma distância de até 1 arcseg, ou seja, consideramos as distâncias dos objetos que potencialmente casam entre si. Em média, nessa distribuição, a distância entre um objeto do *2MASS* e um do *UCAC3* é 0,21 arcseg e o desvio padrão é 0,18 arcseg. Partindo do pressuposto que o casamento destes dois catálogos esteja correto, assumimos que essas são distâncias entre o mesmo objeto presente nos dois catálogos e as usamos para construir o *Golden Standard*. Consideramos cada distância da distribuição como um erro na posição do mesmo objeto em diferentes catálogos.

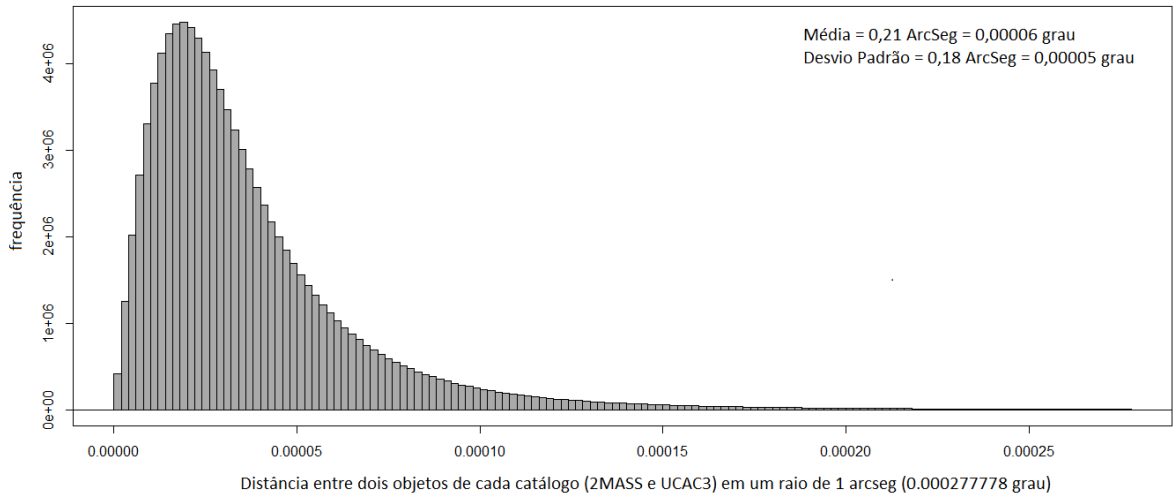


Figura 7.1: Distribuição das distâncias de no máximo 1 arcseg entre os objetos dos catálogos *2MASS* e *UCAC3*

Na criação do *Golden Standard*, tomamos como referência um catálogo real, e criamos catálogos sintéticos a partir dele. Para criar um objeto celeste sintético, tomamos a posição do objeto real e selecionamos aleatoriamente um erro e, dentre os presentes na distribuição da Figura 7.1, a ser adicionado à posição do objeto real. Logo após, geramos aleatoriamente um ângulo θ (de 0 a 360 graus) ao qual esta nova posição estará deslocada em relação à original. Se θ for menor ou igual a 90 graus, por exemplo, o novo objeto terá as seguintes coordenadas:

$$x' = x + e \times cos(\theta) \tag{7.1}$$

$$y' = y + e \times sen(\theta) \tag{7.2}$$

A Figura 7.2 apresenta graficamente a aplicação das Equações 7.1 e 7.2 para gerar o objeto sintético (em branco) com posição (x', y'), a partir do objeto real (em preto) com posição (x, y).

Caso θ seja maior que 90° e menor ou igual a 180° as novas coordenadas serão:

$$x' = x + e \times cos(180^\circ - \theta) \tag{7.3}$$

$$y' = y - e \times sen(180^\circ - \theta) \tag{7.4}$$

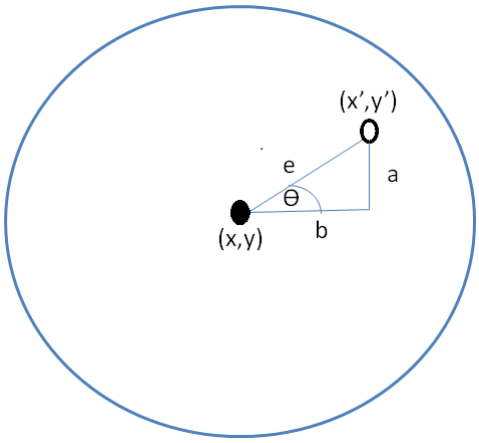


Figura 7.2: Exemplo de um objeto sintético gerado a partir de um objeto real

Se θ for maior que 180° e menor ou igual a 270° as novas coordenadas serão:

$$x' = x - e \times cos(270^\circ - \theta) \tag{7.5}$$

$$y' = y - e \times sen(270^\circ - \theta) \tag{7.6}$$

E, finalmente, caso θ seja maior que 270° e menor ou igual a 360° as novas coordenadas serão:

$$x' = x - e \times cos(360^\circ - \theta) \tag{7.7}$$

$$y' = y + e \times sen(360^\circ - \theta) \tag{7.8}$$

Assim, podemos gerar *n* catálogos sintéticos, a partir de um catálogo real e ter um *Golden Standard*, pois sabemos qual objeto real gerou um sintético, e um casamento entre eles deve ser considerado correto.

Suponha que a Figura 7.3 represente um catálogo real. Cada linha da figura corresponde a um objeto, no qual temos, respectivamente, seu identificador, a ascensão reta e a declinação, separados por ponto e vírgula. Para gerar um conjunto com *n* catálogos, simulamos $n-1$ catálogos a partir do real. Para cada linha, sorteamos um valor de 0 a $n-1$, correspondente à quantidade de objetos sintéticos a serem criados a partir desse objeto real. Cada um dos $n-1$ objetos pertencem a um catálogo diferente.

1;314.9963940000;35.2673570000
2;314.9982180000;35.2673800000
3;314.9947850000;35.2678450000
4;314.9916190000;35.2689510000
5;314.9886660000;35.2703210000

Figura 7.3: Exemplo de um catálogo real com 5 objetos

A Figura 7.4 ilustra a criação de um conjunto simulado de 5 catálogos. Cada linha possui, respectivamente, o identificador do objeto (inteiro), *RA*, *DEC*, identificador do catálogo (varia de 1 a n) e a quantidade de objetos com o mesmo identificador. Para cada linha do catálogo original sorteamos um valor de 0 a 4. O primeiro objeto possui identificador 1 e o valor 0 foi sorteado. Por isso não foi criado nenhum objeto sintético no conjunto de catálogo e apenas repetiu-se o original. A segunda linha do catálogo real possui o identificador 2 e o valor sorteado foi 4. Portanto, criou-se quatro objetos sintéticos pertencentes, respectivamente, aos catálogos 2, 3, 4 e 5. A criação do restante do conjunto de catálogos se deu de forma semelhante.

Podemos observar que o identificador do objeto sintético é o mesmo do objeto ao qual ele é originado. O primeiro e o último parâmetro de uma linha do conjunto de catálogos são usados para detectar casamentos corretos. Por exemplo, quando o primeiro parâmetro é 2 e o último é 5, quer dizer que existem 5 objetos com o identificador 2. Ou seja, o *Golden Standard* diz que o casamento do objeto 2 pelo NACluster, deve ser um agrupamento contendo cinco objetos com *idObjeto* igual a 2. Portanto, em nossos experimentos, consideramos um cluster correto quando ele tem a mesma quantidade de objetos definida pelo *Golden Standard* e todos eles têm o mesmo identificador.

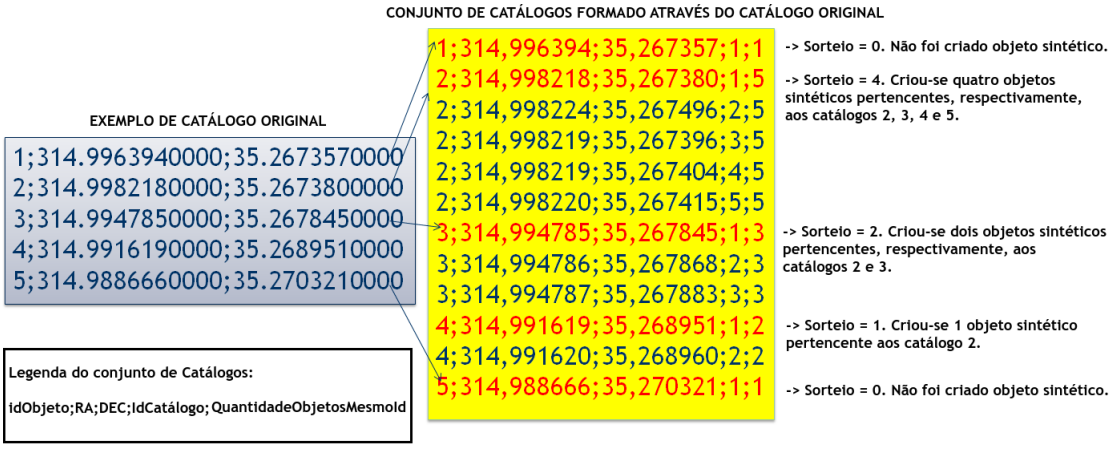


Figura 7.4: Criando um conjunto simulado de 5 catálogos

7.2 Experimentos com o NACluster

Os experimentos desta seção foram realizados com a infra-estrutura do *DEXL LAB*[1]. O hardware utilizado para os testes com o NACluster possui 756 GB de RAM e quarenta e oito processadores Intel Xeon E5-2690 de 2,60 Ghz equipado com CentOS 7 e Java 1.8. Todos os *datasets* de entrada são carregados em memória principal no NACluster a cada experimento realizado.

Realizamos experimentos com conjuntos de catálogos simulados gerados a partir dos catálogos 2MASS, UCAC3 e WISE (Detalhes sobre estes catálogos podem ser encontrados no Capítulo 3).

No primeiro experimento, selecionamos 100 milhões de objetos do catálogo *2MASS*, cuja área coberta corresponde aos pontos escuros da Figura 7.5, e geramos 4 conjuntos de catálogos, através da abordagem mostrada na Seção 7.1. Os conjuntos simulados contêm, respectivamente, 3, 4, 5 e 6 catálogos, e 200, 250, 300 e 350 milhões de objetos nesta ordem. A distribuição dos elementos do último catálogo é ilustrada na Figura 7.6. Em cada conjunto, o catálogo com identificador 1 corresponde ao original.

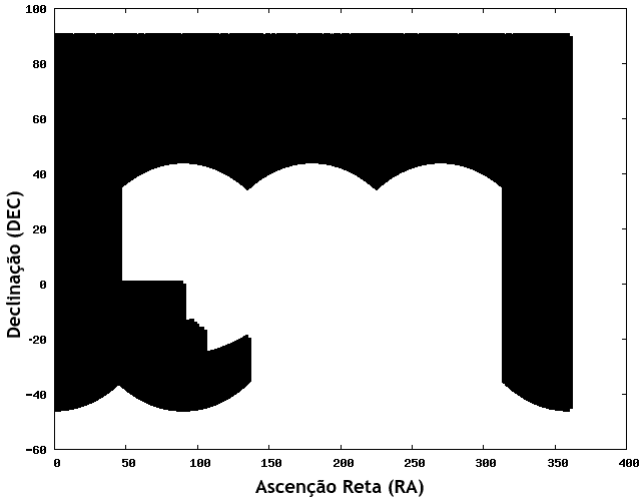


Figura 7.5: Área coberta pelo conjunto de catálogos gerado a partir do *2MASS*

O NACluster foi avaliado através da "qualidade" dos agrupamentos produzidos pelo algoritmo. Em particular, usamos precisão, abrangência (*recall* em inglês) e medida-f na avaliação do casamento gerado pelo NACluster. Lembramos ao leitor que precisão é a proporção de um conjunto de objetos retornados que é realmente relevante; abrangência é a proporção de objetos relevantes que foram retornados; e medida-f é a média harmônica entre precisão e abrangência (MANNING; RAGHAVAN;

[1] Extreme Data Lab - http://dexl.lncc.br

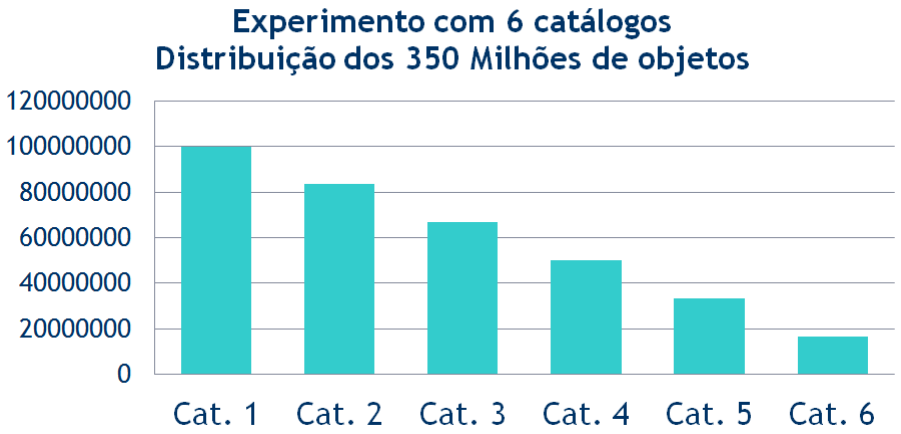


Figura 7.6: Distribuição dos 350 milhões de objetos

#Catálogos	#Objetos	Precisão	Abrangência	Medida-F	#Cand
3	200 Milhões	$0,99973$	$0,99973$	0,99973	31,5 mil
4	250 Milhões	$0,99967$	$0,99967$	0,99967	42 mil
5	300 Milhões	$0,99967$	$0,99967$	0,99967	42 mil
6	350 Milhões	$0,99960$	$0,99960$	0,99960	42 mil

Tabela 7.1: *Métricas de qualidade do NACluster aplicado a diferentes conjuntos de catálogos baseados no 2MASS*

SCHüTZE, 2008) (ver Seção 2.3.1 para maiores detalhes sobre essas métricas).

As Tabelas 7.1, 7.2 e 7.3 apresentam as medidas de qualidade dos resultados do NACluster aplicados, respectivamente, aos conjuntos simulados de catálogos gerados a partir do *2MASS*, *WISE* e *UCAC3* quando comparados ao *Golden Standard*. Cada linha corresponde a uma execução em um conjunto maior de catálogos. A primeira coluna, *#Catálogos*, apresenta a quantidade de catálogos no $dataset$ de entrada e a segunda coluna, *#Objetos*, mostra a quantidade aproximada de astros presentes. As próximas colunas, *Precisão*, *Abrangência* e *Medida-F*, cujos nomes são autoexplicativos, apresentam seus respectivos valores. E, finalmente, a coluna *#Cand* representa a quantidade aproximada de objetos que possuem mais de um cluster, em um raio ε, candidato a alocá-los.

Na Tabela 7.1 observamos que a precisão, abrangência e medida-f permanecem por volta de 0,999, mesmo aumentando a quantidade de catálogos presentes nos conjuntos. Explicamos esse acontecimento através dos dados da última coluna, pois a grande maioria dos objetos encontram apenas um centroide no raio de busca ε, ou seja, eles possuem apenas um cluster para ser alocado, e o casamento tende a ser correto. Na primeira linha da tabela, apenas 31,5 mil objetos, $0,0126\%$ dos 250 milhões, possuem mais de um cluster candidato a alocá-lo. A medida-f não é 1,0 pois

#Catálogos	#Objetos	Precisão	Abrangência	Medida-F	#Cand
3	100 Milhões	0,994690	0,994694	0,994692	256 mil
4	125 Milhões	0,993607	0,993619	0,993612	375 mil
5	150 Milhões	0,992846	0,992859	0,992852	484 mil
6	175 Milhões	0,991349	0,991363	0,991356	637,5 mil

Tabela 7.2: Métricas de qualidade do NACluster aplicado a diferentes conjuntos de catálogos baseados no *UCAC3*

alguns casamentos dos 31,5 mil objetos foram diferentes do *Golden Standard*.

Para o segundo experimento, selecionamos 50 milhões de objetos do *UCAC3*, cuja área coberta é representada pelos pontos escuros da Figura 7.7. Geramos 4 conjuntos S, simulando variações do mesmo catálogo através da abordagem mostrada na Seção 7.1. Os conjuntos simulados contêm, respectivamente, 3, 4, 5 e 6 catálogos. Em todos eles, o catálogo com identificador 1 corresponde ao original. Os resultados do experimento são apresentados na Tabela 7.2.

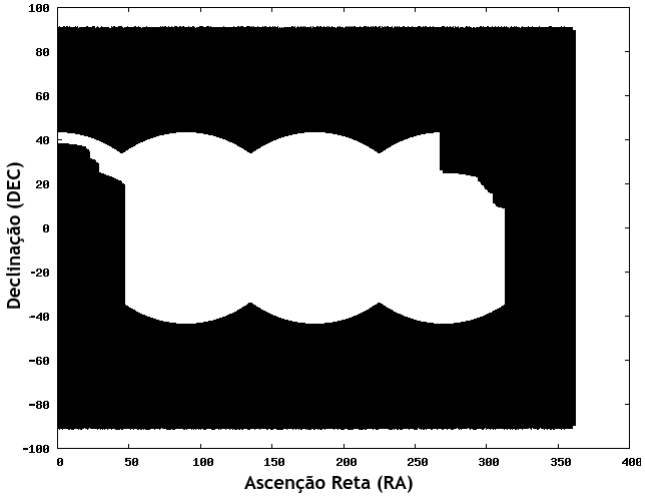


Figura 7.7: Área coberta pelo conjunto de catálogos gerado a partir do *UCAC3*

A Tabela 7.2 possui valores de precisão, abrangência e medida-f por volta de 0,99, mas podemos observar que a medida que aumentamos a quantidade de catálogos presentes nos conjuntos, os valores vão decrescendo na ordem de 0,001. A explicação para este fenômeno é dada pelos valores da última coluna, pois percebemos que a quantidade de objetos com mais de um cluster candidato a alocá-los é maior a cada linha, aumentando, assim, a suscetibilidade a erros no casamento. Para visualizar este cenário apresentamos a Figura 7.8, onde a medida que aumentamos o número de objetos que possuem mais de um cluster candidato a alocá-lo (#Cand), menor é a qualidade da clusterização. Esse é um problema de ambiguidade e é assunto da próxima seção.

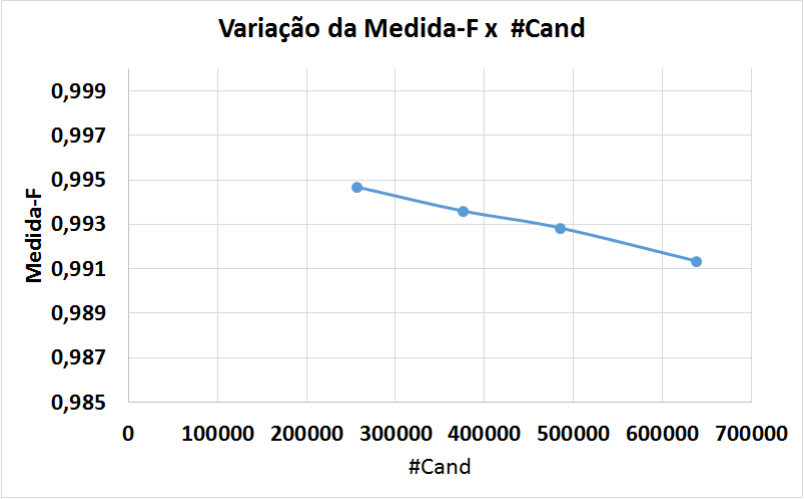


Figura 7.8: Variação da Medida-F em função do o número de objetos que possuem mais de um cluster candidato a alocá-lo durante a execução do NACluster sobre os conjuntos de catálogos (UCAC3)

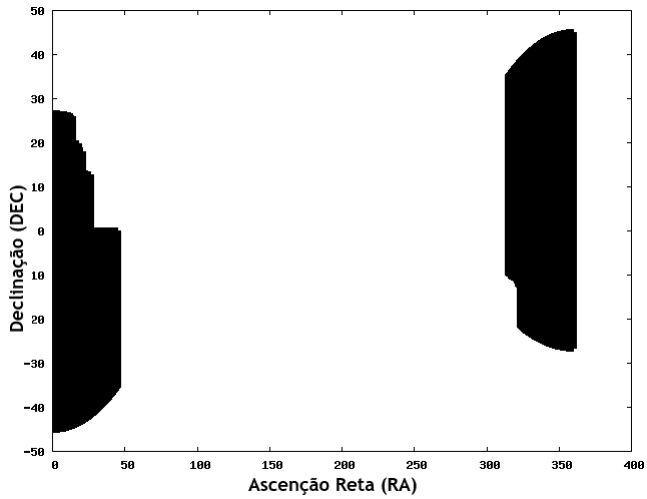


Figura 7.9: Área coberta pelo conjunto de catálogos gerado a partir do *WISE*

Um outro experimento similar foi feito com conjuntos de catálogos gerados a partir de 50 milhões de objetos do WISE (WRIGHT et al., 2010), cuja região do espaço é ilustrada pela Figura 7.9. A Tabela 7.3 apresenta os resultados, nos quais a precisão, abrangência e medida-f são todas 1,0, mesmo aumentando a quantidade de catálogos no conjunto. A qualidade do casamento é máxima, pois a coluna *#Cand* tem valor 0 para todos os *datasets*. Portanto, cada objeto encontra em ε apenas o cluster correto para ser alocado. Concluímos que o catálogo original é esparso e os objetos são bem distantes uns dos outros.

Os resultados dos experimentos induzem que os valores de precisão, abrangência e medida-f estão diretamente relacionados à quantidade de centroides encontrados para cada objeto dentro dos limites de busca do raio ε. A partir dessa relação,

#Catálogos	#Objetos	Precisão	Abrangência	Medida-F	#Cand
3	100 Milhões	1,0	1,0	1,0	0
4	125 Milhões	1,0	1,0	1,0	0
5	150 Milhões	1,0	1,0	1,0	0
6	175 Milhões	1,0	1,0	1,0	0

Tabela 7.3: Métricas de qualidade do NACluster aplicado a diferentes conjuntos de catálogos baseados no *WISE*

definimos o conceito de ambiguidade na próxima seção.

7.2.1 Medida-F x Grau Médio de Ambiguidade

Com o objetivo de encontrarmos alguma correlação entre a qualidade da clusterização do NACluster e a ambiguidade existente no agrupamento, propomos a métrica grau médio de ambiguidade na Seção 5.4 para avaliar a ambiguidade do ponto de vista dos objetos, ou seja, o quanto cada objeto é ambíguo em relação aos clusters.

Temos a hipótese de que quanto maior o grau médio de ambiguidade no conjunto de catálogos, pior é a qualidade do casamento. Até então, em todos os experimentos, tivemos apenas altos valores de medida-f, entre 0,99 e 1,0. Aplicamos a métrica no mesmo conjunto de 3 catálogos usado no experimento da Tabela 7.1 e tivemos como resultado o valor 0,017. Acreditamos que a medida-f, com valor próximo de 1, tem correlação com o grau médio de ambiguidade, que está próximo de 0. Portanto, precisamos de conjuntos de catálogos que possuem altos valores de $\overline{\mathrm{G}}$ para comparar com a qualidade do seu casamento.

Para um conjunto de catálogos ter $\overline{\mathrm{G}}$ maior que 1, cada objeto deve possuir pelo menos dois centroides em um raio ε. Essa característica está ligada à densidade do catálogo original, em relação a um dado ε. Caso ele seja denso, simular um conjunto de catálogos a partir dele fará com que tenhamos muitos centroides próximos uns dos outros, dado que as posições dos centroides iniciais são as mesmas dos objetos do catálogo original. Portanto, geramos diferentes níveis de densidade no catálogo original e a partir dele simulamos o conjunto de catálogos.

Chamamos de nível de densidade a quantidade de astros gerados para cada objeto do catálogo original. Por exemplo, quando temos nível de densidade igual a 1, para cada objeto do catálogo original geramos um novo objeto à uma distância aleatória dentre as definidas na Figura 7.1. Ao termos o nível de densidade igual a 2, criamos dois novos objetos a partir do objeto do catálogo original, e assim por diante. O processo de criação de objeto é semelhante ao apresentado na Seção 7.1. A diferença é que o objeto simulado deve pertencer ao mesmo catálogo do original.

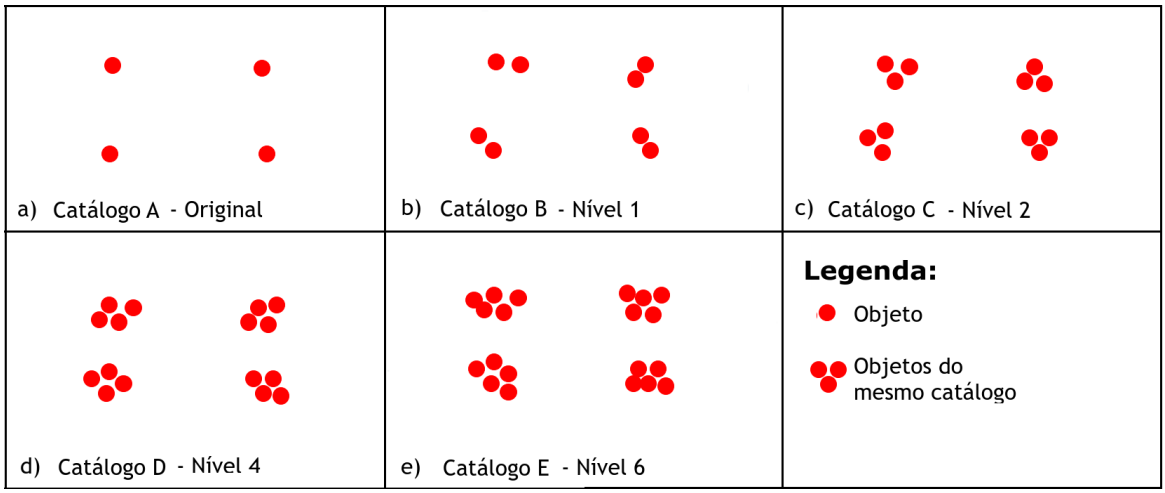


Figura 7.10: Gerando densidade no catálogo

A Figura 7.10 ilustra a criação de catálogos densos simulados. Em *a* apresentamos um exemplo do catálogo original com 4 objetos. Todos os demais são criados a partir dele. Em "*b*", "*c*", "*d*" e "*e*", para cada objeto de *A* geramos *y* novos objetos, onde *y* corresponde ao nível de densidade e, assim, a união dos objetos do catálogo original com astros criados formam o novo catálogo.

Ainda com relação à Figura 7.10, *B* tem 8 objetos, nos quais cada um deles possui outro astro com distância menor ou igual a 1 arcseg. Se gerarmos um conjunto de catálogos através de *B* e o usarmos como entrada no NACluster, na inicialização dos centroides será criado um centroide para cada objeto do maior catálogo. Como B será o maior catálogo deste conjunto, teremos pelo menos dois centroides candidatos para cada objeto. Dessa forma, o conjunto de catálogos gerado terá $\overline{\mathrm{G}} \geq 1$.

O catálogo *C* tem 12 objetos, os mesmos 4 de *A* e mais 8 criados a partir de *A*. Se gerarmos um conjunto de catálogos através de *C* e o usarmos como entrada no NACluster, no instante da criação dos centroides iniciais será criado um centroide para cada objeto do maior catálogo. Como C será o maior catálogo deste conjunto, teremos pelo menos três centroides candidatos para cada objeto. Dessa forma, o conjunto de catálogos gerado terá $\overline{\mathrm{G}} \geq 2$.

O processo de criação de *D* e *E* são semelhantes ao de *C*. Através dessa abordagem, podemos partir de qualquer catálogo real e gerar vários conjuntos de catálogos com diferentes valores de $\overline{\mathrm{G}}$ e constatar a correlação entre $\overline{\mathrm{G}}$ e a qualidade do casamento do NACluster.

Consideramos parte do *2MASS* como nosso catálogo original e geramos densidades com níveis de 0 a 10. A partir desses 11 catálogos, simulamos 11 conjuntos contendo 3 catálogos cada e aplicamos o NACluster sobre eles. A Figura 7.11

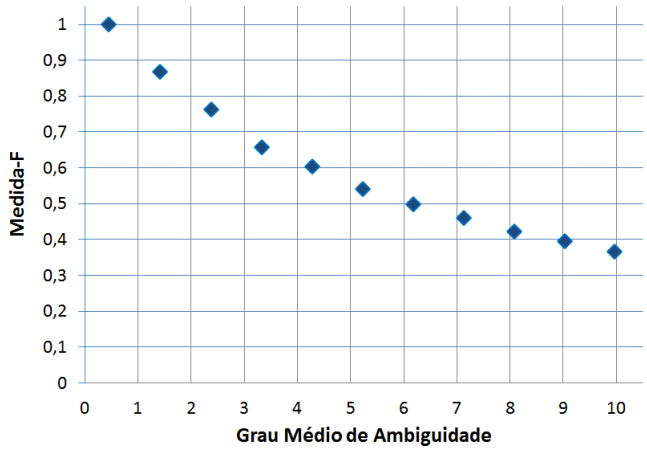


Figura 7.11: Grau médio de Ambiguidade x Medida-F

apresenta a correlação entre a qualidade do NACluster (medida-f) e o grau médio de ambiguidade destes conjuntos. Podemos perceber, que o conjunto gerado a partir do catálogo original possui grau médio de ambiguidade próximo de 0 e medida-f próxima de 1. O conjunto formado a partir do catálogo com densidade nível 1 possui $\overline{G}$ igual a 1,41 e medida-f igual a 0,868. O gráfico mostra que quanto maior o grau médio de ambiguidade, menor é a qualidade do resultado do NACluster. Portanto, a técnica tem um limite de aplicação em relação ao nível de ambiguidade e não é aplicável a qualquer cenário. A taxa de acerto é superior a 60% quando o grau médio de ambiguidade é menor ou igual a 4. Independente disso, o NACluster adiciona ao estado da arte o tratamento correto quando múltiplos catálogos precisam ser casados e não apresenta problemas de transitividade. Para os catálogos reais utilizados, o grau médio de ambiguidade fica entre 0 e 1 e a taxa de acerto é entre 99% e 100%. Finalmente, vale ressaltar que dado um conjunto de catálogos, é possível separar os dados e aplicar o NACluster onde tem menor ambiguidade. E em trabalhos futuros, podemos adicionar na função de distância outros atributos e o NACluster tenderá a funcionar bem em regiões ambíguas.

7.2.2 Volume x Média do Tempo de Execução

Realizamos experimentos para medir o tempo de execução do *NACluster* em relação ao aumento do volume do conjunto de catálogos. Para cada $dataset$ de entrada, executamos 10 vezes o algoritmo e calculamos a média do tempo de execução. O processo de execução corresponde ao tempo de carga do $dataset$ do disco para a memória, juntamente com a realização do casamento e o tempo de gravação em disco da clusterização e da lista de objetos candidatos para cada cluster .

A Figura 7.12 apresenta a relação entre o volume e a média do tempo de execução do NACluster em segundos. Através do gráfico, concluímos que a relação *volume x tempo* tem tendência linear.

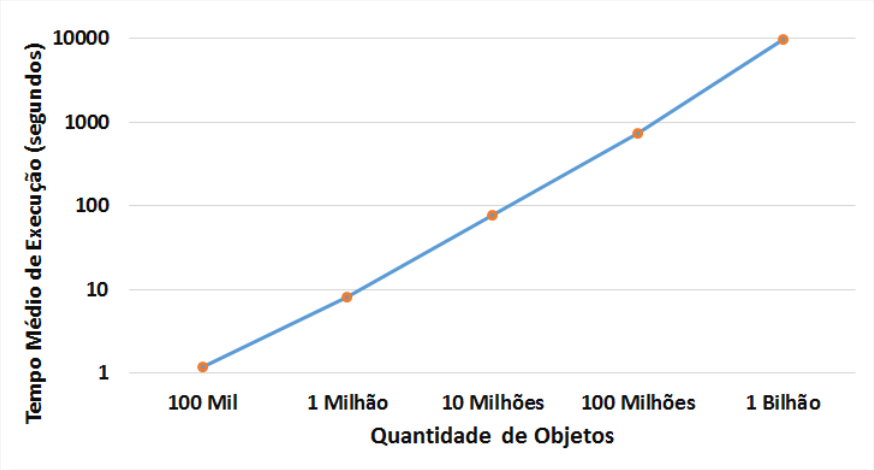


Figura 7.12: Volume x Tempo Médio de Execução do NACluster

7.2.3 A influência dos centroides iniciais no resultado final

O algoritmo NACluster sempre define seus centroides iniciais como sendo a posição dos objetos do maior catálogo, pois sua quantidade corresponde ao número mínimo de clusters do resultado final. Para saber se a escolha dos centroides iniciais influencia na saída do algoritmo, permitimos escolher o catálogo no qual as posições dos seus objetos serão inicializados como centroides. Realizamos experimentos com o NACluster modificado variando todas as possibilidades de inicialização dos centroides. Usamos como *dataset* de entrada, alguns dos conjuntos de catálogos utilizados na Seção 7.2.

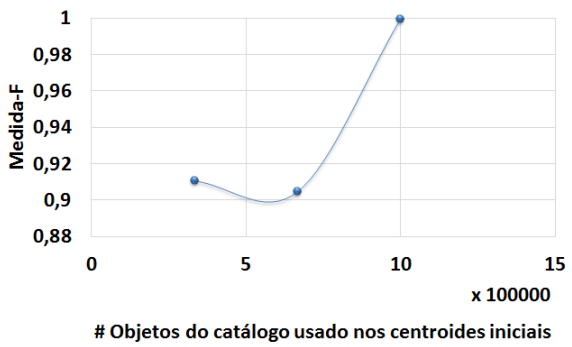
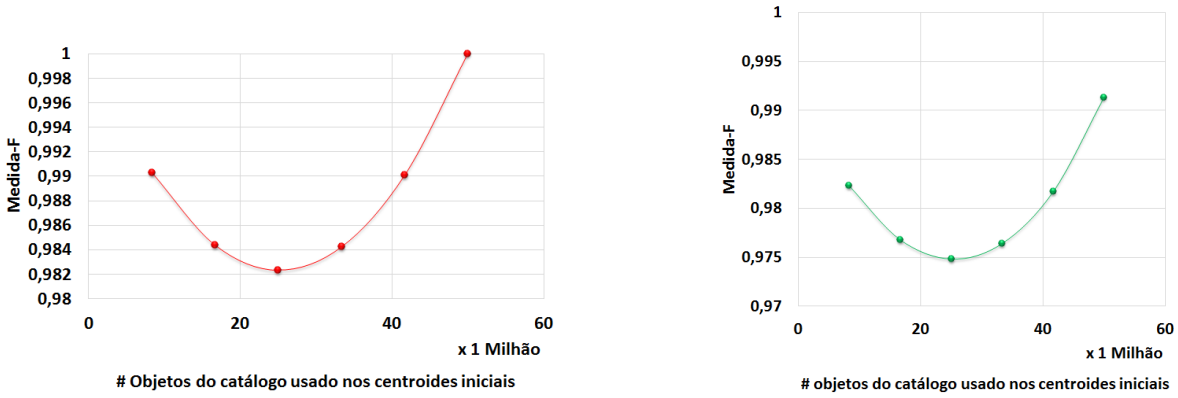


Figura 7.13: Variação dos centroides iniciais (Conjunto de Catálogos 2MASS) x Medida-F

A Figura 7.13 mostra uma comparação entre a escolha dos centroides iniciais e a medida-f do casamento do conjunto de 3 catálogos simulado a partir do 2MASS e com $\overline{G}$ igual a 2,37 (mesmo *dataset* usado na Figura 7.11). No eixo *x* temos o tamanho do catálogo cujas posições dos seus objetos são usadas como centroides iniciais, e no eixo *y* a medida-f. Podemos observar que quando a inicialização dos centroides ocorre pelo maior catálogo, a medida-f do resultado é superior, pois os clusters iniciais possuem centroides com as mesmas posições dos objetos do maior catálogo e, assim, a maioria dos objetos estão próximos dos clusters aos quais serão alocados. Os objetos que não estão próximos dos centroides iniciais sempre criarão novos clusters durante a execução do NACluster. Portanto, ao iniciarmos os centroides pelo maior catálogo, diminuímos a quantidade de criação de clusters ao longo da execução do NACluster e a possibilidade de erro na alocação dos objetos. Nos próximos experimentos com uma versão modificada do NACluster, veremos que o erro na alocação, quando inicializamos os centroides por catálogos menores, pode ser corrigido através de uma segunda iteração do NACluster.



(a) Conjunto de Catálogos WISE

(b) Conjunto de Catálogos UCAC3

Figura 7.14: Variação dos centroides iniciais x Medida-F

As Figuras 7.14a e 7.14b apresentam um comparativo entre a escolha dos centroides iniciais e a qualidade do casamento dos conjuntos de 6 catálogos gerados a partir do *WISE* e *UCAC3*, respectivamente. Assim como na Figura 7.13, constatamos que quando a inicialização dos centroides ocorre pelo maior catálogo o casamento tem melhor qualidade. Em todos os experimentos essa característica sempre se repete, portanto, a ideia de inicializar os centroides pela posição dos objetos do maior catálogo é melhor que inicializá-los pela posição de catálogos menores.

Os resultados dos experimentos acima mostram que a escolha dos centroides iniciais influencia na qualidade do resultado. Por padrão, o NACluster sempre começa com centroides na posição dos objetos do maior catálogo, garantindo, assim, o melhor casamento. No entanto, o NACluster modificado pode ter uma qualidade se-

melhante ao NACluster original quando o primeiro realizar uma segunda iteração. Ou seja, depois que o casamento entre todos os catálogos tiver sido concluído, deve-se remover os objetos dos clusters, manter os centroides, e realizar uma nova iteração do NACluster.

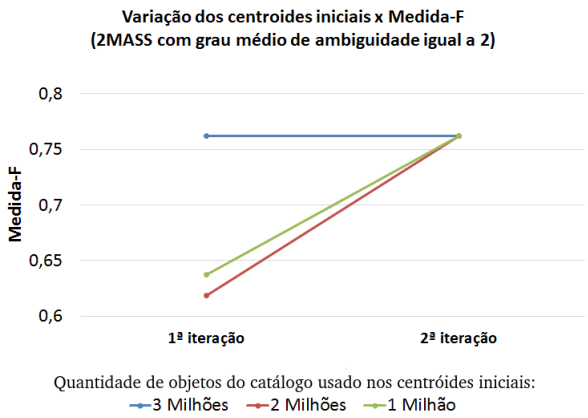


Figura 7.15: Variação dos centroides iniciais (Conjunto de Catálogos UCAC3) x Medida-F

Fizemos experimentos com o NACluster modificado e capturamos a qualidade do casamento ao variarmos os centroides iniciais e realizarmos duas iterações. Sempre que realizamos a segunda iteração no NACluster modificado, qualquer que seja a quantidade de objetos do catálogo pelo qual ocorre a inicialização dos centroides, a medida-f é semelhante ao do NACluster original. Os resultados podem ser vistos nas Figuras 7.15, 7.16 e 7.17. Justificamos esse comportamento através da criação dos clusters referentes ao maior catálogo. Quando a inicialização dos centroides não se dá pela posição dos objetos do maior catálogo, os clusters referentes ao maior catálogo só vão sendo criados durante a execução da primeira iteração e é por isso que a convergência se dá somente na segunda iteração.

A Figura 7.15 ilustra o experimento feito com o NACluster modificado ao usar o mesmo *dataset* da Figura 7.13. Observamos que na primeira iteração, a medida-f varia de acordo com a quantidade de objetos do catálogo usado nos centroides iniciais. No entanto, na segunda iteração, todas as medidas-f convergem para um valor semelhante, independente do tamanho catálogo usado na inicialização dos centroides.

As Figuras 7.17 e 7.16 ilustram os experimentos realizados com o NACluster modificado ao usar, respectivamente, os mesmos *datasets* das Figura 7.14a e 7.14b. O comportamento da medida-f nos dois conjuntos de catálogos é semelhante ao da figura anterior. Ou seja, apesar da medida-f variar de acordo com a quantidade de objetos do catálogo usado nos centroides iniciais na primeira iteração, ao terminar a

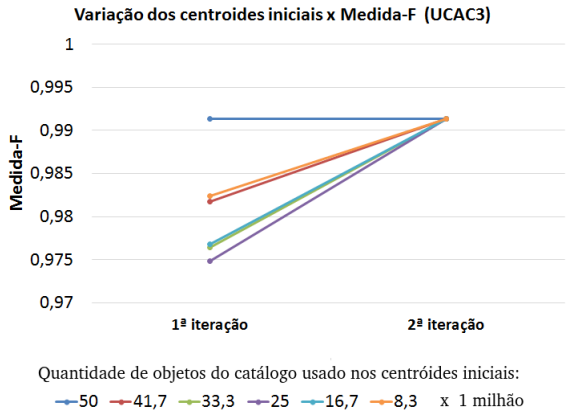


Figura 7.16: Variação dos centroides iniciais (Conjunto de Catálogos UCAC3) x Medida-F

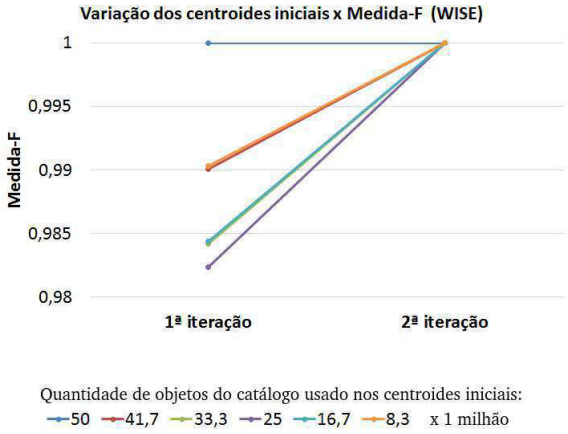


Figura 7.17: Variação dos centroides iniciais (Conjunto de Catálogos UCAC3) x Medida-F

segunda, todas as medidas-f convergem para um valor semelhante, independente do tamanho catálogo usado na inicialização dos centroides. Logo, concluímos que a qualidade do casamento é independente da escolha do catálogo utilizado na inicialização dos centroides, desde que ocorra uma segunda iteração do NACluster. Além disso, o algoritmo precisa de apenas uma iteração para convergir quando a inicialização dos centroides é feita a partir da posição dos objetos do maior catálogo.

7.3 Comparação com Q3C Join

O Q3C Join (KOPOSOV; BARTUNOV., 2012), como a maioria dos algoritmos de *cross-matching*, realiza o casamento de apenas dois catálogos por vez, produzindo, como saída, um catálogo contendo os objetos que casam entre si. Com o objetivo de identificar a diferença de comportamento do NACluster e do *Q3C Join*,

comparamos as saídas geradas por eles tendo como entrada dois catálogos. Realizamos o casamento entre parte do catálogo *2MASS* (1 milhão de objetos) e um catálogo sintético gerado a partir dele (também com 1 milhão de objetos). Na geração do catálogo, para cada objeto do original geramos um novo objeto a uma distância x através de uma distribuição normal com média 0,00066 e desvio padrão 0,00024 grau. A faixa de valores permitida é mostrada na Figura 7.18.

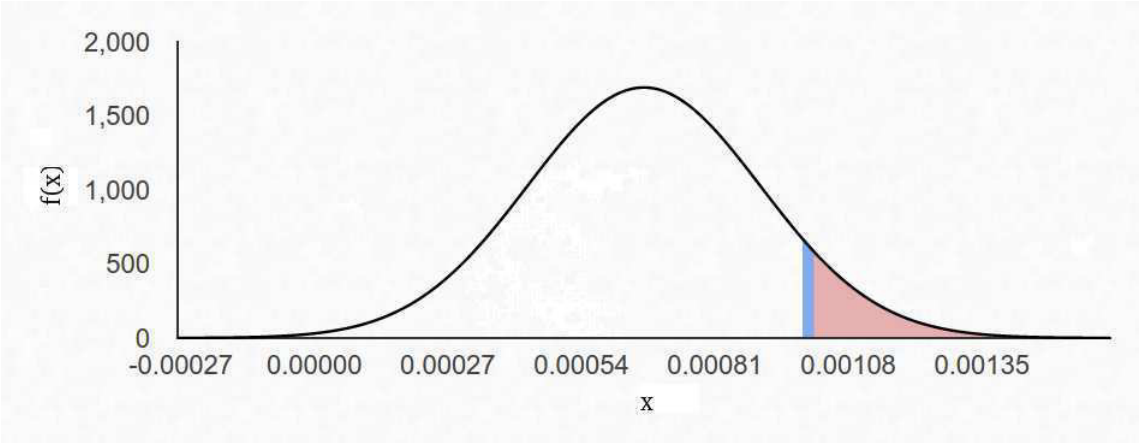


Figura 7.18: Faixa de valores das possíveis distâncias x

Os experimentos desta seção avaliam o comportamento do *NACluster* e do *Q3C Join* ao aumentar o raio de busca utilizado no casamento. Dividimos os testes em duas etapas. Na primeira delas, usamos um raio de 0,001 grau no Q3C para buscar um novo objeto, e no NACluster consideramos um ε de 0,001 grau para buscar centroides vizinhos. Conforme a Figura 7.18, a maioria das distâncias são menores que 0,001. Na segunda etapa, avaliamos a qualidade do casamento ao aumentar o raio do *Q3C Join* e o ε do NACluster para 0,002 grau e, assim, podermos capturar todos objetos ou centroides que, de acordo com o *Golden Standard*, fariam o casamento correto.

As Tabelas 7.4 e 7.5 apresentam a precisão, abrangência e medida-F desses experimentos. Valores em negrito correspondem aos maiores valores. Na Tabela 7.4, nota-se que o *Q3C* com raio igual a 0,001 grau tem melhor qualidade que o NACluster com ε igual a 0,001. No entanto, para abranger todos os casamentos, devemos aumentar o raio de busca. Na Figura 7.18 temos em destaque todos as distâncias possíveis maiores que 0,001 entre dois objetos “iguais” presentes nos dois catálogos. Para não perder nenhuma das possibilidades de acertar o casamento, o raio deve ser maior e abranger todas elas.

Dobramos o raio do *Q3C* e o ε do *NACluster* para 0,002 com a expectativa de melhorar a qualidade do casamento. Ao observar o resultado, apresentado na Tabela 7.5, percebemos que o *NACluster* melhorou em todas as medidas, mas o *Q3C*

Algoritmo	Precisão	Abrangência	Medida-F
Q3C (0,001 grau)	**0,9520**	**0,9644**	**0,9582**
NACluster (0,001 grau)	0,9462	0,9466	0,9464

Tabela 7.4: *NACluster x Q3C em precisão, abrangência e medida-F do casamento (com raio igual a 0,001 grau) entre parte do catálogo 2MASS e o catálogo sintético gerado a partir dele.*

Algoritmo	Precisão	Abrangência	Medida-F
Q3C (0,002 grau)	0,7456	**1,0**	0,8549
NACluster (0,002 grau)	**0,9779**	0,9806	**0,9792**

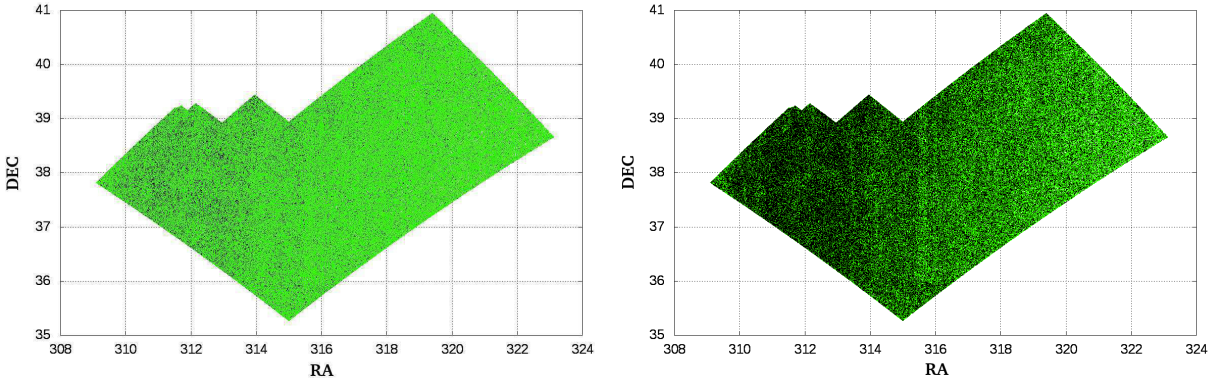
Tabela 7.5: *NACluster x Q3C em precisão, abrangência e medida-F do casamento (com raio igual a 0,002 grau) entre parte do catálogo 2MASS e o catálogo sintético gerado a partir dele.*

piorou bastante em precisão e medida-f. A abrangência do Q3C chegou ao seu valor máximo, pois tivemos o número de acertos igual ao *Golden Standard*, no entanto detectamos muitos erros, prejudicando, assim, a qualidade final do casamento (medida-f baixa).

O aumento dos erros no casamento do *Q3C* pode ser visualizado através da Figura 7.19. As Figuras 7.19a e 7.19b mostram a região do céu coberta pelos catálogos utilizados na comparação. O eixo *x* é a coordenada Ascensão Reta, ou *RA*, e o eixo y é a Declinação, ou *DEC*. Cada ponto em verde é um objeto casado corretamente, ou seja, ele casou com seu par real (de acordo com *Golden Standard*). Cada ponto preto é um objeto casado de forma errada, ou seja, não casou com seu par real. É importante mencionar que os os pontos verdes foram plotados antes dos pontos pretos, bem como, devido à baixa resolução da plotagem, os pontos ocupam mais espaços que seu tamanho real. Assim, a comparação fica mais visível a olho nu e podemos perceber o grande crescimento dos erros no casamento ao aumentarmos o raio usado.

Para efeito de comparação entre a quantidade de erros de casamento do Q3C com raio 0,002 grau e a quantidade de erros de casamento do NACluster com ε 0,002, geramos a Figura 7.20 nos mesmos moldes da Figura 7.19. Podemos observar que a quantidade de erros de casamento no NACluster são bem menores que no Q3C, mesmo utilizando o mesmo raio ou ε.

Finalmente, a Figura 7.21 mostra uma comparação direta da medida-f para esses cenários. Observamos no gráfico que o NACluster melhora o desempenho quando aumentamos o ε e o *Q3C* piora a medida-F ao aumentar o raio. Dentre todos, o NACluster com ε 0,002 tem melhor desempenho.



(a) *Q3C* com raio (0,001) (b) *Q3C* com raio (0,002)

Figura 7.19: Casamento correto (pontos verdes) X Casamento errado (pontos pretos)

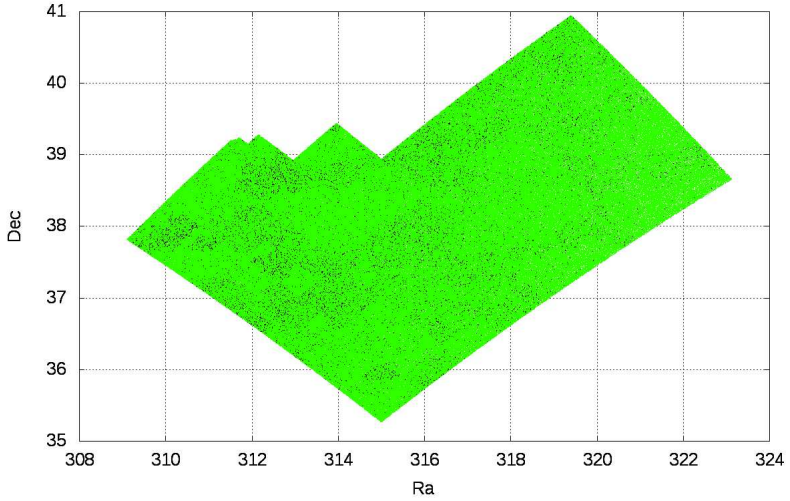


Figura 7.20: Casamentos corretos (pontos verdes) X Casamentos errados (pontos pretos) pelo NACluster com raio 0,002

O resultado dessa comparação nos leva a concluir que, diferente da estratégia do algoritmo NACluster de dar uma solução para a ambiguidade dos casamentos, o *Q3C Join* tem uma estratégia de preservá-la, aceitando duplicidade nos casamentos. Essa característica faz com que o resultado do *Q3C* seja bastante dependente do raio utilizado. Caso seja utilizado um raio maior que a maioria das distâncias entre objetos “iguais” de diferentes catálogos ou eles sejam densos, a saída pode conter muito ruído. No *NACluster* o aumento de ε não piora a qualidade do casamento. É importante mencionar que, apesar do NACluster não preservar a ambiguidade no resultado, a saída da implementação mostra, além dos clusters, os outros objetos com potencial a serem alocados em cada cluster para análise manual por parte do usuário.

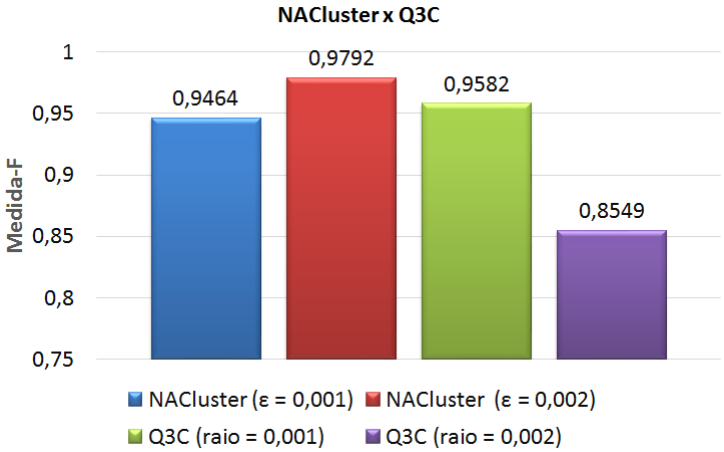


Figura 7.21: Comparação entre NACluster e Q3C Join

7.4 Particularidade no tratamento do casamento

Durante a realização dos testes e análise dos resultados, encontramos uma particularidade no tratamento do casamento. Suponha que tenhamos dois clusters com centroides c_1 e c_2, respectivamente, que distam menos que 2ε um do outro, conforme a Figura 7.22. Na figura, cada círculo em torno de um centroide tem raio ε. Cada cluster contem dois objetos de catálogos diferentes (a cor dos objetos na figura representa a identidade do catálogo). Podemos observar que os dois clusters tem uma interseção na sua área de cobertura. Analisando a interseção podemos identificar os seguintes fatos:

1. na disposição dos objetos nos clusters, o_1 está casando com o_2, mas outras configurações poderiam ser possíveis, por exemplo o_2 casar com o_3 e, então o_1 e o_4 formarem dois novos clusters.

2. o critério de menor distância ao centroide para alocar os objetos pode ficar confuso em situações como essa.

Vejamos um exemplo prático encontrado na execução do NACluster. Suponha os seguintes objetos no esquema (id,RA,DEC,idCatalogo)

- o_1,314.519368,36.991344,1
- o_2,314.518546,36.991600,2
- o_3,314.518200,36.991684,1
- o_4,314.517378,36.991940,2

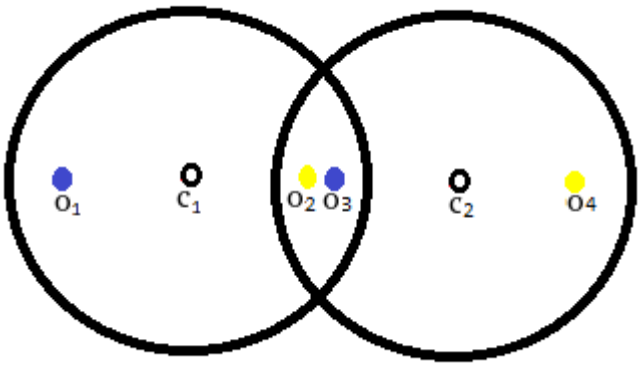


Figura 7.22: Interseção no raio ε entre dois clusters

A inicialização do NACluster cria dois clusters iniciais, conforme mostrado na Figura 7.23:

- O cluster c_1 com centroide c_1 na posição (314.519368, 36.991344) contendo o objeto o_1;
- e o cluster c_2 com centroide c_2 na posição (314.5182, 36.991684) contendo o objeto o_3.

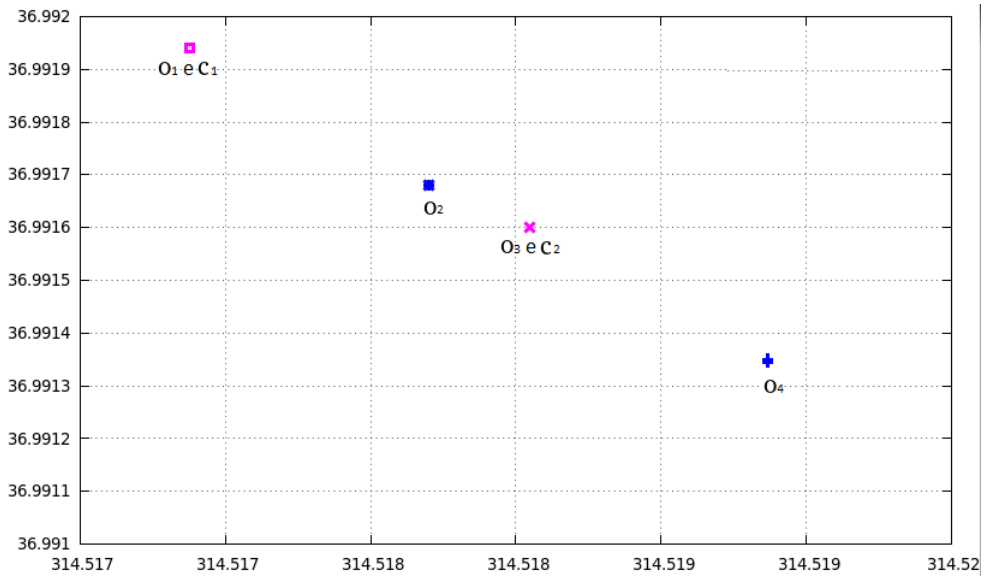


Figura 7.23: Os quatro objetos (o_1, o_2, o_3 e o_4) e o centroide c_1 do cluster contendo somente o objeto o_1; e o centroide c_2 do cluster contendo somente o objeto o_3

Durante a execução do NACluster, ao buscar-se os centroides mais próximos dos objetos o_2 e o_4, os dois clusters c_1 e c_2 são candidatos à alocá-los. Então, neste modelo, teríamos o cluster c_1 com o objeto o_1 do catálogo 1 e o cluster c_2 com

os objetos o_3 do catálogo 1, e os objetos o_2 e o_4 do catálogo 2. Como não podemos ter dois objetos do mesmo catálogo em um mesmo agrupamento, pelo NACluster, o objeto o_2 está mais perto do centroide c_2 e seria alocado ao cluster c_2, enquanto o objeto o_4 procuraria outro cluster, e alocar-se-ia ao c_1. Logo, o NACluster formaria os seguintes clusters:

- O cluster com centroide c_1 na posição (314.518373, 36.991642) contendo os objetos o_1 e o_4;
- e o cluster com centroide c_2 na posição (314.518373, 36.991642) contendo os objetos o_3 e o_2.

Observando o que aconteceu, percebemos que ao calcular a média das posições dos objetos alocados nos clusters, os centroides convergiram para o mesmo ponto. O cenário é ilustrado na Figura 7.24. Os centroides c_1 e c_2 do gráfico possuem a mesma posição no espaço, mas as elipses que identificam os dois clusters são ilustrativas, e os objetos dentro da elipse menor não pertencem ao cluster da elipse maior.

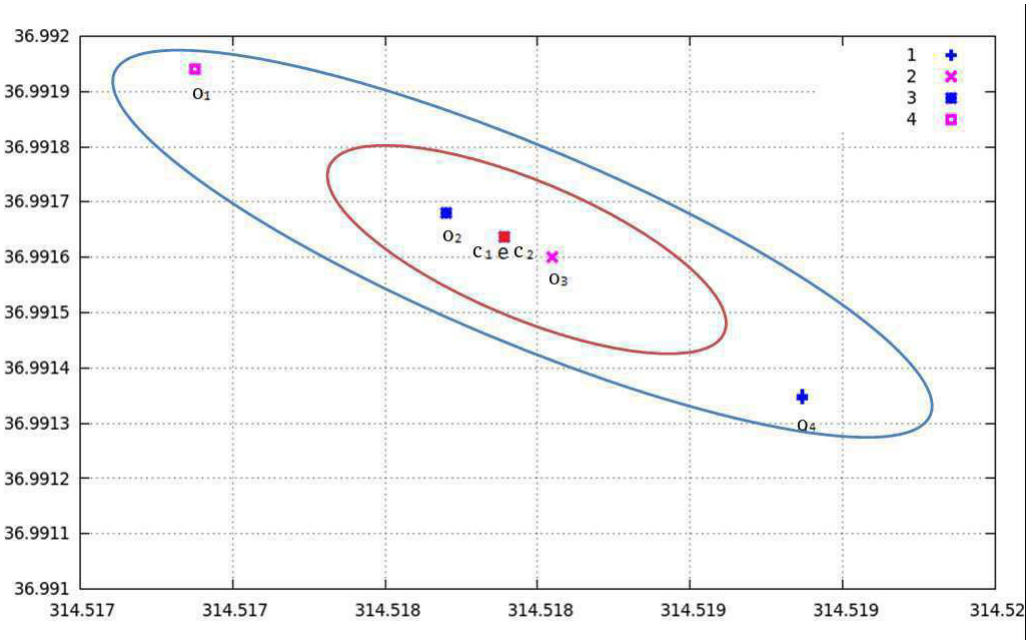


Figura 7.24: Clusters com com centroides na mesma posição formados depois da execução do NACluster

Esse é um tipo de problema que foi ignorado pelo NACluster e que pode ser tratado em trabalhos futuros. Cenários como os ilustrados nesta seção podem se repetir sempre que a distância entre dois objetos do mesmo catálogo for menor ou igual à 2ε.

#Catálogos	Tamanho	#Objetos	Precisão	Abrangência	Medida-F
3	31 GB	946 Milhões	0,99947	0,99947	0,99947
4	39 GB	1,1 Bilhão	0,99847	0,99849	0,99848
5	47 GB	1,4 Bilhão	0,99847	0,99849	0,99848
6	54 GB	1,6 Bilhão	0,99828	0,99831	0,99829

Tabela 7.6: *Métricas de qualidade do ParallelNACluster aplicado a diferentes conjuntos de catálogos baseados no 2MASS*

7.5 Experimentos com o ParallelNACluster

Nesta seção apresentamos os experimentos envolvendo o *ParallelNACluster*. Conforme visto no Capítulo 6, a versão paralela aceita o mesmo tipo de arquivo de entrada da centralizada. Portanto, usamos os mesmos conjuntos de catálogos usados nos experimentos da Seção 7.2.

O hardware utilizado nos testes com o ParallelNACluster é um *cluster*[2] formado por 5 máquinas comuns presentes no *DEXL Lab* (laboratório pertencente ao LNCC):

- **Master**: Processador core i7-4790, com 4 núcleos e 8 *threads* de 3,60Ghz. Além de memória RAM de 16G.
- **Slaves 1, 2 e 3**: Processador AMD Phenom(tm) II X2 B55 com 2 núcleos de 3,00 Ghz e 8GB de memória RAM.
- **Slave 4**: Processador core i5-3470, com 4 núcleos de 3,20Ghz e memória RAM de 8G.

7.5.1 Qualidade na execução de grandes catálogos

A Tabela 7.6 apresenta as medidas de qualidade dos casamentos feitos através do ParallelNACluster quando aplicado a grandes conjuntos simulados de catálogos gerados a partir de todos os objetos do *2MASS* com tamanhos variados de 31 GB a 54 GB e particionados em 500 partições. Cada linha corresponde à uma execução em um conjunto de catálogos de tamanho diferente. A primeira coluna, #*Catálogos*, apresenta a quantidade de catálogos. A segunda coluna mostra o tamanho do arquivo e a terceira coluna, #*Objetos*, representa a quantidade aproximada de objetos. As próximas colunas, *Precisão*, *Abrangência* e *Medida-F*, cujos nomes são autoexplicativos, apresentam seus respectivos valores.

[2]Cluster no sentido de ser formado por computadores ligados que trabalham em conjunto.

#Catálogos	#Clusters	#Clusters Corretos	#Clusters Errados	Precisão
3	646.361	613.845	32.516	0,9497
4	674.410	660.499	13.911	0,9794
5	692.301	659.987	32.314	0,9533
6	716.592	680.947	35.645	0,9503

Tabela 7.7: *Qualidade do casamento da fronteira na etapa Reduce dos mesmos conjuntos de catálogos da Tabela 7.6.*

Observamos que as medidas de qualidade permanecem por volta de 0,99, mesmo aumentando a quantidade de catálogos presentes nos conjuntos. Esses valores são semelhantes aos apresentados nos experimentos com o NACluster centralizado.

Também medimos a qualidade do casamento dos objetos influenciados pelas fronteiras, ou seja, aqueles identificados pelo *SCIBoundary* e casados na etapa *Reduce*. Como cada *dataset* possui 500 partições, identificamos 500 fronteiras. Nos conjuntos de catálogos testados, as fronteiras contêm aproximadamente $0,14\%$ do total de clusters.

A Tabela 7.7 apresenta a qualidade do casamento ocorrido com os objetos da fronteira. A primeira coluna contém a quantidade de catálogos do *dataset*. A segunda mostra a quantidade total de clusters da fronteira. A terceira apresenta o número de agrupamentos da fronteira classificados de forma correta segundo o *Golden Standard*, enquanto a quarta coluna mostra a quantidade de clusters errados. Esses valores foram usados para calcular a precisão, apresentada na coluna seguinte.

Na Seção 7.2.1 foi apresentada a correlação entre a qualidade do NACluster e o grau médio de ambiguidade de conjuntos de catálogos com diversos níveis de densidade. Realizamos experimentos com os mesmos *datasets* no ParallelNACluster para compararmos a qualidade da clusterização da versão paralela, da centralizada e do casamento ocorrido na fronteira.

Os resultados da comparação são apresentados na Figura 7.25. No eixo *x* temos o grau médio de ambiguidade e no eixo *y* a precisão do casamento. Os pontos no formato de losango correspondem à precisão do casamento ocorrido no ParallelNACluster. Os quadrados representam a qualidade da clusterização ocorrida no *SCIBoundary* e os triângulos dizem respeito à precisão do NACluster centralizado. Observamos que mesmo ao aumentarmos o grau médio de ambiguidade, a qualidade do ParallelNACluster, do NACluster e do SCIBoundary são semelhantes. Apenas a precisão no SCIBoundary fica, em média, com uma diferença em torno de 10^{-2} no valor da precisão se comparado aos demais, pois sua amostra é pequena, ou seja, a sua quantidade de *clusters* é muito menor que os demais.

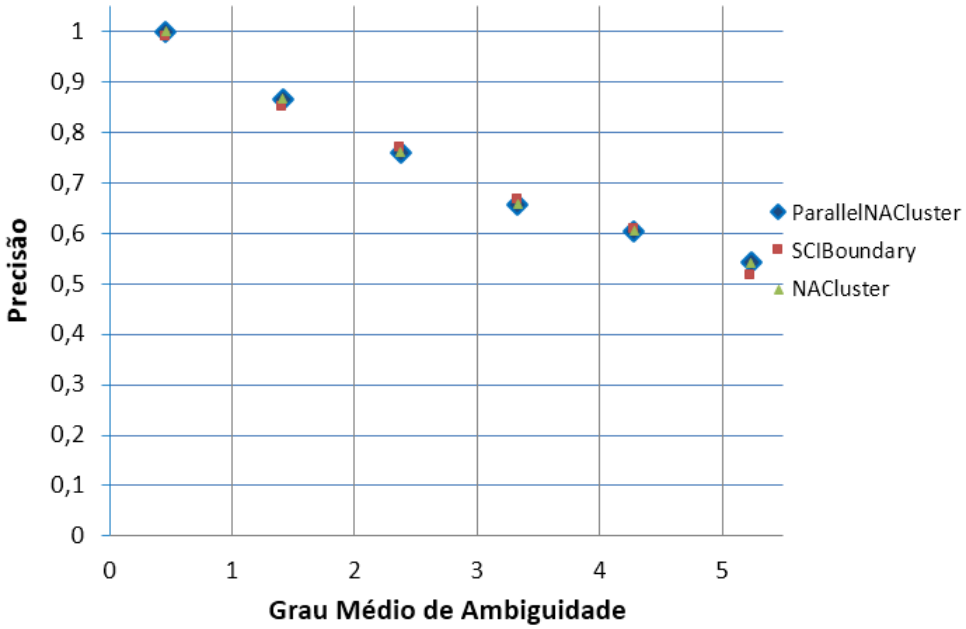


Figura 7.25: Grau médio de Ambiguidade x Precisão

7.5.2 Variação dos parâmetros do Spark x Tempo médio de execução

Nas demais Subseções apresentamos diversos gráficos mostrando a relação entre a variação dos parâmetros de submissão da aplicação spark ParallelNACluster e seu o tempo de execução. Para cada parâmetro variado foram feitas 10 execuções e a partir delas foram calculadas as médias do tempo e do desvio padrão. Ao todo fizemos aproximadamente 650 execuções da nossa aplicação spark. Detalhes sobre os parâmetros podem ser encontrados na Seção 2.5 do Capítulo 2.

Para melhor entendermos o que acontece nas execuções, é primordial conhecermos os estágios da execução do ParallelNACluster no Spark. Um estágio corresponde a um conjunto de tarefas que executam o mesmo código, cada um em um subconjunto diferente dos dados. Cada estágio contém uma sequência de transformações que pode ser concluída sem realizar o *shuffle* (embaralhamento na rede) dos dados completos.

Segundo (RYZA, 2015a), em operações como *Map* e *Filter* não ocorre *shuffle*, pois registros necessários na execução de uma única partição residem em uma única partição no RDD pai. Cada objeto só é dependente de um único objeto no pai.

Nas operações como o *ReduceByKey*, os dados necessários para calcular os registros em uma única partição podem residir em várias partições do RDD pai. Todas as tuplas com a mesma chave devem estar na mesma partição e serem processadas pela mesma tarefa. Para satisfazer estas operações, o Spark deve executar

um *shuffle*, que transfere dados em todo o cluster e resulta em um novo estágio com um novo conjunto de partições (RYZA, 2015a).

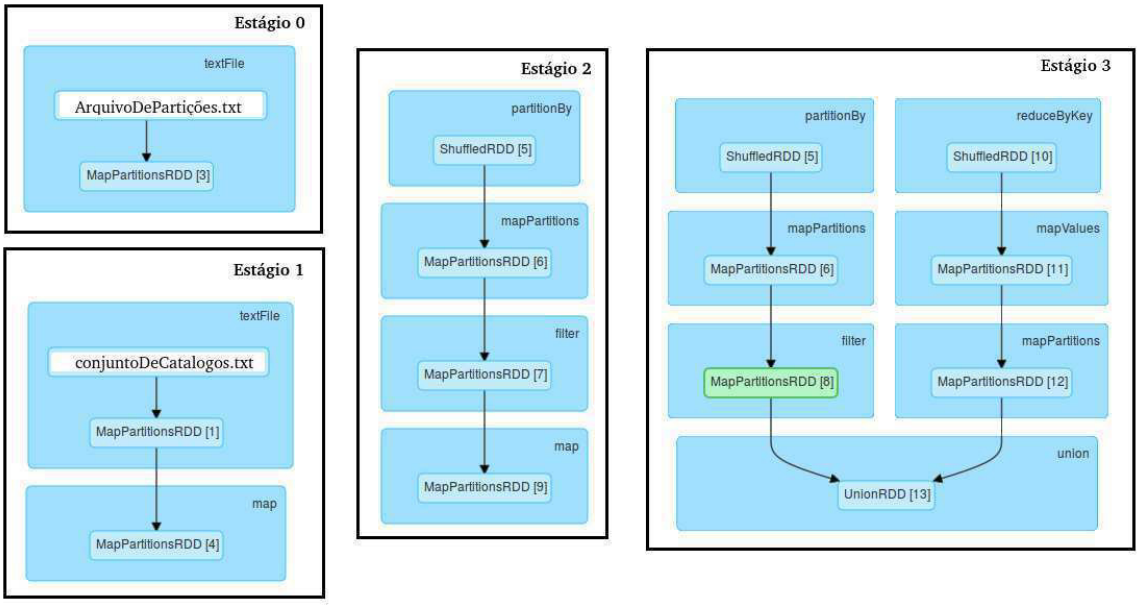


Figura 7.26: Estágios da execução do ParallelNACluster

A Figura 7.26 ilustra os estágios presentes na execução do ParallelNACluster do início ao fim da clusterização. A numeração ao lado dos RDDs mostra a sequência em ordem crescente em que eles foram criados. Ao todo temos 4 estágios, eles são:

- **Estágio 0 -** Carregamento do arquivo de partições gerado pelo FRANCE.
- **Estágio 1 -** Carregamento do arquivo correspondente ao conjunto de catálogos e, logo após, o mapeamento de cada objeto para a forma <chave,valor>, onde a chave corresponde à partição em que o objeto deve ser alocado e o valor é o conteúdo do objeto. O particionamento através do *partitionBy* decreta o fim deste estágio, pois ele precisa realizar um *shuffle* de todos os dados.
- **Estágio 2 -** Os dados particionados (*ShuffledRDD [5]*) são direcionados à transformação *mapPartitions*, a qual executa localmente o NACluster e o SCIBoundary e gera o *MapPartitionsRDD [6]*. Uma operação *filter* é feita com o objetivo de selecionar os clusters influenciados pela fronteira e um mapeamento é feito preparando os dados para o Reduce[3]. O *ReduceByKey* decreta o fim deste estágio, devido ao *shuffle* gerado por ele.

[3]Maiores detalhes da implementação são descritos no Capítulo 6

- **Estágio 3 -** O mesmo RDD, *MapPartitionsRDD [6]*, gerado pela transformação *mapPartitions* que executou o NACluster e o SCIBoundary no Estágio 2, é submetido a uma operação *filter* com o objetivo de identificar os clusters que não são influenciados pela fronteira e estes são armazenados no *MapPartitionsRDD [8]* e persistidos em disco para serem usados como entrada da operação *union* juntamente com o resultado da etapa Reduce, na qual o NACluster é aplicado sobre os clusters influenciados pela fronteira. O *UnionRDD [13]* contém a união da clusterização sem influência da fronteira com a clusterização influenciada pela fronteira já corrigida.

7.5.2.1 Número de Partições

A Figura 7.27 mostra o tempo médio de execução do ParallelNACluster ao variarmos o número de partições. No eixo *x* temos a quantidade de partições e no eixo *y* o tempo médio de execução em segundos. O desvio padrão é ilustrado pelas linhas verticais presentes em cada uma das barras. Usamos um conjunto de catálogos de 134 MB contendo 2,5 milhões de objetos e o submetemos à aplicação com os seguintes parâmetros:

- *–num-executors 4*
- *–executor-memory 5G*
- *–executor-cores 2*

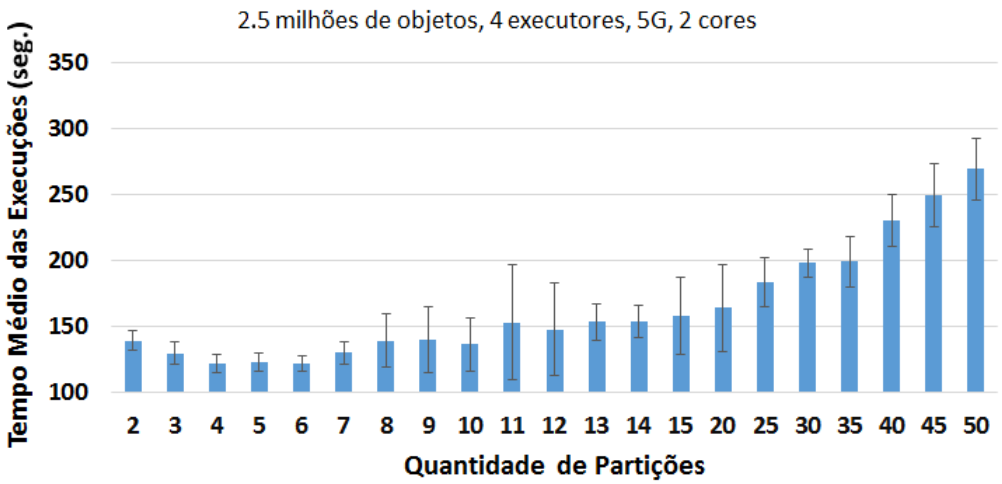


Figura 7.27: Quantidade de Partições x Tempo Médio de Execução

O número de partições foi variado de 2 a 50 nas execuções. Os melhores tempos médios ocorreram quando tínhamos de 4 a 6 partições. Quando o número

de partições era menor que a quantidade de instâncias, o tempo médio de execução também era baixo, dado que cada nó era responsável apenas por uma única partição, e a mesma era pequena e cabia em memória. No entanto, o tempo médio de execução era mais alto do que quando tínhamos de 4 a 6 partições, pois com 2 ou 3 partições, um ou dois nós do cluster, respectivamente, ficavam ociosos durante o estágio 2 (estágio em que ocorre o NACluster e o SCIBoundary). A partir de 4 partições, o paralelismo passou a ocorrer em todos os nós e, por isso, o tempo médio diminuiu.

De 7 partições em diante, o tempo tendeu a subir de acordo com o aumento no número de partições, pois o uso do disco aumenta com o crescimento do número de partições, e cada partição do RDD é persistida em disco toda vez que sua tarefa correspondente é concluída. Escolhemos persistir em disco pois a memória RAM do cluster utilizado para testes é pequena e ficaria cheia com facilidade. Caso usássemos um cluster com maior capacidade de memória RAM, poderíamos persistir os dados em memória, ao invés de disco, e assim, agilizar as execuções.

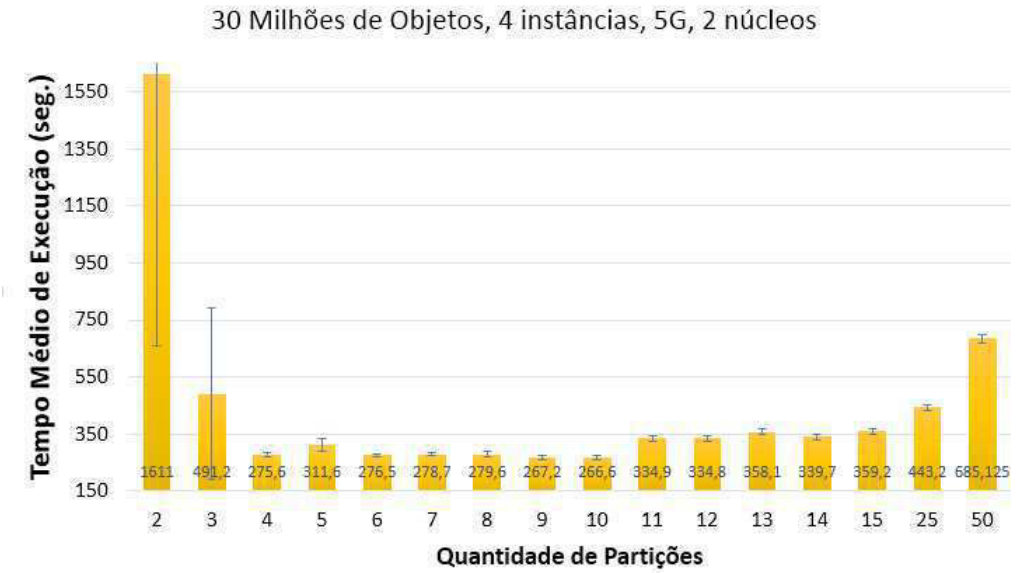


Figura 7.28: 2º Experimento com Quantidade de Partições x Tempo Médio de Execução

Fizemos um segundo experimento, semelhante ao primeiro, no qual fixamos os mesmos parâmetros, e usamos um conjunto de catálogos maior, com 952 MB de tamanho e 30 milhões de objetos. Variamos apenas o número de partições de 2 a 50. A Figura 7.28 apresenta o resultado deste experimento. Podemos perceber que ao usarmos apenas 2 partições, com 4 instâncias, 2 núcleos de CPU, e 5G de memória por instância, tivemos muitos erros do tipo *Lost Executors* durante a execução do estágio 3, e por isso, o tempo médio foi o mais alto de todos os testes.

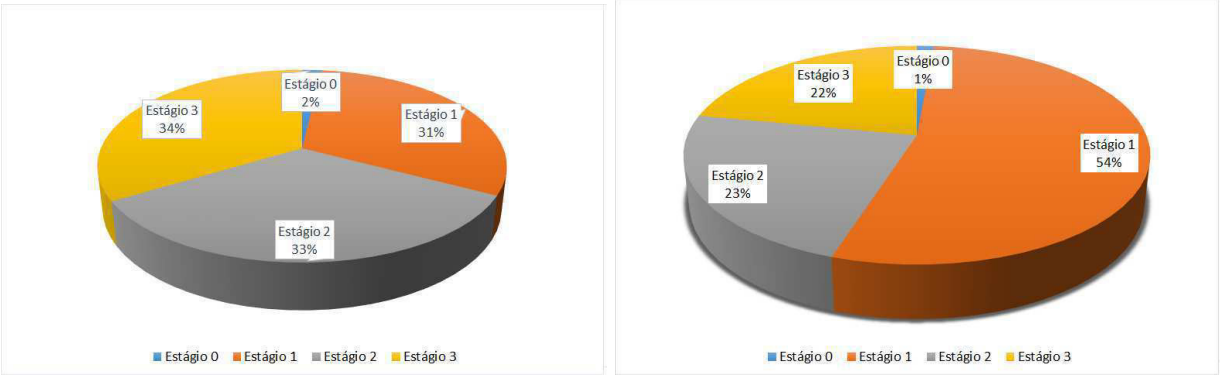
Ter as duas partições geradas pelo *FrancePartitioner*, fez com que todos os dados fossem divididos para serem alocados em apenas dois nós no estágio 2. Ao ir para o estágio 3, as duas partições são transformadas em 4, ou seja, duas grandes partições com os *clusters* não influenciados pela fronteira e duas pequenas partições com os clusters influenciados pela fronteira. Ao trabalhar com essas partições no estágio 3, a memória de uma das instâncias poderia ficar cheia, e o YARN executava um *kill* nesta instância, fazendo com que um nova instância fosse criada e os passos fossem repetidos, até que a execução total fosse feita com sucesso.

Ainda analisando o gráfico da Figura 7.28, ao usarmos 3 partições, o *dataset* é alocado em 3 nós (dos 4) após o particionamento, e o processamento do NACluster e do SCIBoundary não ocorre em todos os nós, deixando ociosa uma das instâncias, além de 3 núcleos (1 de cada nó utilizado). Tivemos os melhores resultados com 9 e 10 partições, pois foi a quantidade que melhor se aproveitou do paralelismo juntamente com o tamanho ideal das partições. A partir de 11 partições, o tempo de execução tendeu a subir e aumentou-se a quantidade de leitura e gravação em disco.

7.5.2.2 Proporção de tempo em cada Estágio do ParallelNACluster

A Figura 7.29 apresenta a proporção de tempo de execução de cada estágio do *ParallelNACluster* ao variarmos o número de partições do conjunto de catálogos de 30 milhões de objetos. Nos experimentos acima, encontramos o melhor tempo médio ao usarmos, para este *dataset*, 10 partições, 4 instâncias, 5G de memória e 2 núcleos. Na Figura 7.29a apresentamos a proporção de tempo em cada estágio para este cenário. Observamos que há um equilíbrio no tempo de cada um dos principais estágios (1, 2 e 3). Ao aumentarmos a quantidade de partições para 40, Figura 7.29b, percebemos que o estágio 1 ocupa a maior parte do tempo da execução da aplicação, ou seja, o *FrancePartitioner* ocupa 54% do tempo da execução do ParallelNACluster. Enquanto os estágios 2 e 3, que contém a implementação do NACluster/SCIBoundary e Reduce, respectivamente, continuam equilibrados, com 23% e 22% do tempo. Assim, percebemos que 40 é uma quantidade de partições muito grande para este *dataset* e este número impacta no tempo destinado ao estágio de particionamento.

Nesta subseção, concluímos que não devemos ter uma quantidade de partições menor que a quantidade de instâncias disponíveis para o processamento dos dados, bem como não podemos exagerar na quantidade de partições, pois pode aumentar consideravelmente o tempo de execução da aplicação. E, conforme mostrado em (VELLORE, 2015), não existe um número de partições ideal, ele deve ser encontrado a medida que for executando e obter um bom equilíbrio de simultaneidade e



(a) 30 Milhões de objetos, 10 partições (b) 30 Milhões de objetos, 40 partições

Figura 7.29: Proporção do tempo por estágio

tempo de execução das tarefas.

7.5.2.3 Quantidade de Instâncias

Nos próximos experimentos executamos o ParallelNACluster usando como entrada os mesmos conjuntos de catálogos da Seção 7.5.2.1 e fixando o número de partições com os melhores resultados encontrados, ou seja, 4 e 10 partições para os arquivos de 134 MB e 952 MB respectivamente.

A Figura 7.30 mostra a variação do número de instâncias. No eixo *x* temos a quantidade de instâncias e no eixo *y* o tempo médio de execução. O desvio padrão é ilustrado pelas linhas verticais presentes em cada uma das barras. Ao todo temos 4 nós e variamos as instâncias de 1 a 4.

Usamos no primeiro experimento o conjunto de catálogos de 134 MB contendo 2,5 milhões de objetos divididos em 4 partições e o submetemos à aplicação com os seguintes parâmetros:

- *–executor-memory 5G*
- *–executor-cores 2*

A cada experimento variamos o parâmetro *–num-executors* com valores de 1 a 4. Ao considerarmos uma instância de execução, toda a aplicação é executada em apenas um nó, por isso temos o maior tempo médio. Com duas instâncias o tempo de execução melhora, pois as quatro partições são processadas em paralelo, já que temos dois núcleos ativos em cada instância e cada núcleo fica responsável por uma partição. Ao considerarmos três instâncias de execução, as quatro partições são distribuídas em três nós e um deles aloca uma partição a mais em relação aos outros.

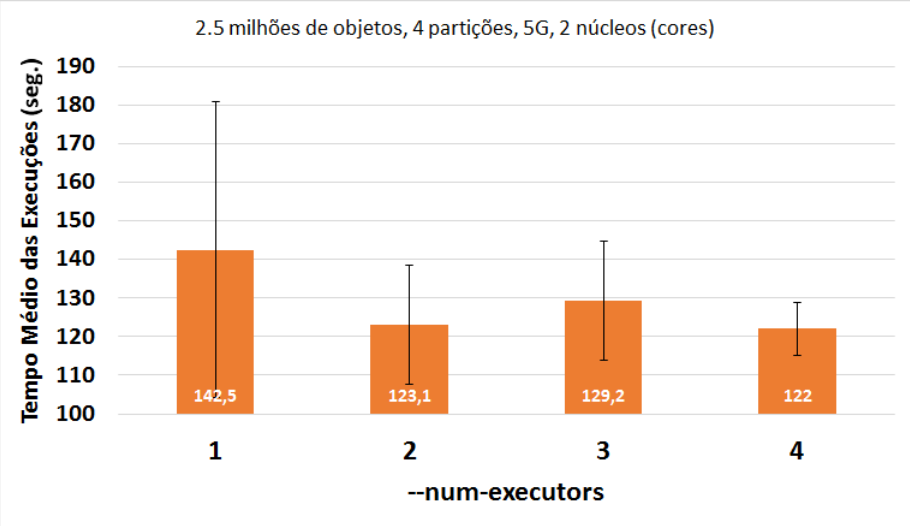


Figura 7.30: Quantidade de Instâncias (*–num-executors)* x Tempo Médio de Execução

Essas partições são consumidas pelos núcleos disponíveis, ou seja, na instância com duas partições, cada um dos dois *cores* fica responsável por uma delas, realizando o processamento em paralelo. Como cada uma das quatro partições é processada ao mesmo tempo, então existe um balanceamento. Ao considerarmos três instâncias existe uma variação no tempo, mas ela aparece dentro do desvio padrão do exemplo com duas e quatro instâncias. Assim, a partir do número de *cores* igual ao número de partições, temos uma execução balanceada. Não adianta ter *cores* em excesso.

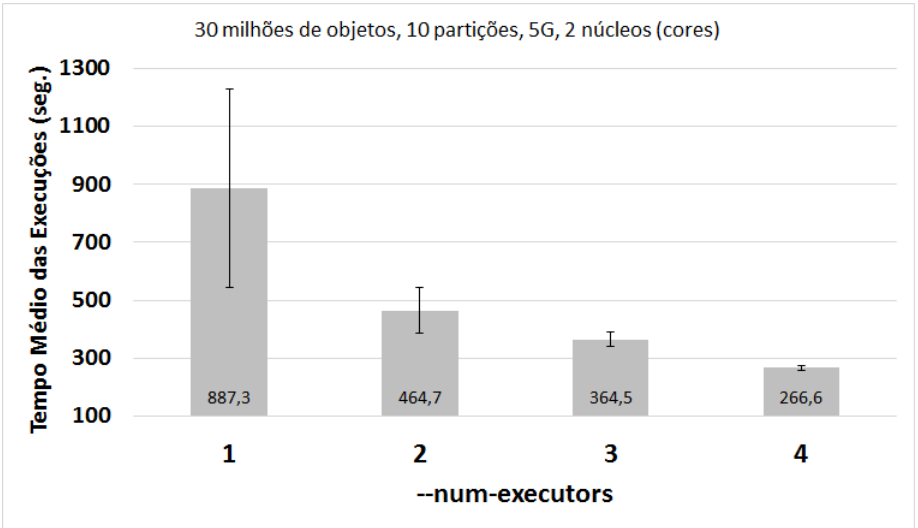


Figura 7.31: Quantidade de Instâncias (*–num-executors)* x Tempo Médio de Execução (2º Experimento)

Fizemos um segundo experimento, semelhante ao primeiro desta seção, no qual fixamos os mesmos parâmetros, e usamos um conjunto de catálogos maior, com 952 MB de tamanho e 30 milhões de objetos. Variamos apenas o número de

instâncias de 1 a 4. Os resultados são apresentados na Figura 7.31.

Ao termos apenas uma instância, todas as partições foram alocadas no mesmo nó e o recurso de distribuição no cluster não foi usado, por isso tivemos um tempo médio alto de 887,3 segundos além de um grande desvio padrão (341 segundos).

Quando executamos com duas instâncias, o tempo médio de execução melhorou e foi diminuído quase que pela metade, passou de 887,3 para 464,7 segundos. O motivo desta queda foi o processamento simultâneo das partições que antes estavam alocados em apenas um nó.

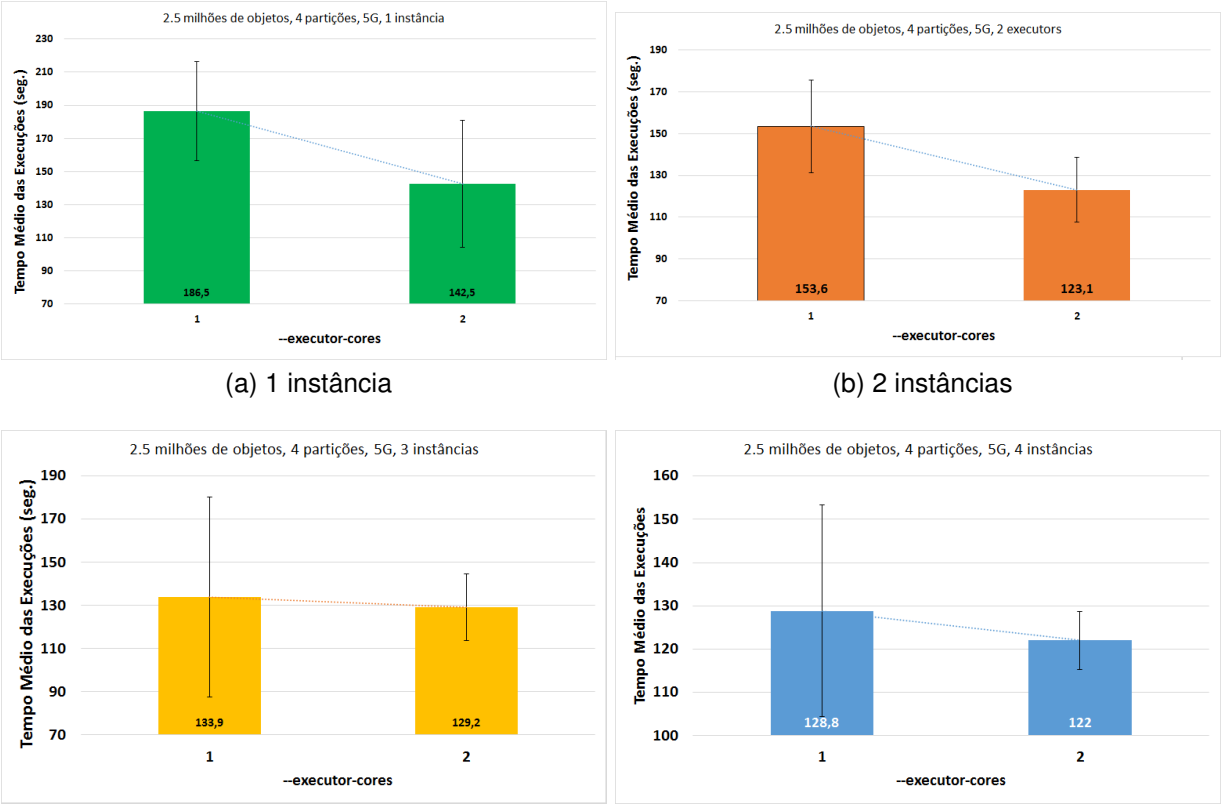
As execuções com três *executors* aumentaram o paralelismo e diminuíram o tempo médio em relação às execuções com duas instâncias, mas não superaram o melhor tempo da Seção 7.5.2.1, representado pela última coluna da Figura 7.31.

7.5.2.4 Quantidade de núcleos utilizada por instância

Usamos um conjunto de catálogos de 134 MB contendo 2,5 milhões de objetos divididos em 4 partições e o submetemos ao ParallelNACluster. A cada experimento fixamos as instâncias, com valores de 1 a 4, 5G de memória cada, e variamos o parâmetro *–executor-cores* com valores de 1 a 2, pois em nosso cluster a maioria das máquinas possuem apenas 2 núcleos.

Os gráficos da Figura 7.32 ilustram esse experimento. Em 7.32a mostramos o tempo médio de execução (em segundos) ao utilizarmos apenas uma instância e variarmos a quantidade de núcleos disponíveis neste único *executor*. Percebemos que ao aumentarmos de um para dois núcleos, o tempo melhora, pois as partições são processadas em paralelo pelas duas *CPUs*. Ao termos duas instâncias, o comportamento é parecido, mas o tempo de execução é melhor que o anterior nos dois casos, pois o recurso de paralelismo do cluster é utilizado e dois nós processam os dados ao mesmo tempo.

Na Figura 7.32c mostramos o desempenho ao variarmos a quantidade de núcleos e fixarmos três instâncias. Podemos perceber que a diferença do tempo médio da execução com um e dois núcleos é pequena, pois ocorre um desbalanceamento na alocação das partições, já que nos dois casos, um dos nós sempre vai receber duas partições e os outros dois receberão uma cada. O alto desvio padrão ao termos apenas um núcleo por instância é explicado pela diferença de *hardware* do cluster, pois como mencionado no início da Seção 7.5, existem três nós com processador AMD Phenom II X2 B55 de dois núcleos de 3,0 GHz e um nó com processador Intel Core



(a) 1 instância

(b) 2 instâncias

(c) 3 instâncias

(d) 4 instâncias

Figura 7.32: Quantidade de Núcleos (*–executor-cores)* x Tempo Médio de Execução

i5 de dois núcleos de 3,20 Ghz. A cada execução, três dos quatro nós são escolhidos aleatoriamente para alocar as instâncias. Quando o nó com processador Intel é um dos escolhidos, a execução é mais rápida do que quando os três nós possuem processador AMD. E ainda devido ao desbalanceamento na alocação das partições, o tempo médio chega a ser pior do que com duas instâncias e dois núcleos.

Na Figura 7.32d, como esperado, temos os melhores resultados, pois estamos usando as 4 instâncias e cada partição é alocada a um nó, tendo, assim, o maior grau de paralelismo.

7.5.2.5 Quantidade de Memória utilizada pelas Instâncias

Nos próximos experimentos executamos o ParallelNACluster usando como entrada os mesmos conjuntos de catálogos da seção anterior e fixando uma quantidade de 4 e 10 partições para os arquivos de 134 MB e 952 MB respectivamente.

No primeiro experimento submetemos à aplicação o conjunto de catálogos de 134 MB, contendo 2,5 milhões de objetos divididos em 4 partições, e fixamos os seguintes parâmetros a cada execução:

- *–num-executors 1*
- *–executor-cores 1*

O único parâmetro variado foi o *–executor-memory*. E como o arquivo de entrada é pequeno, avaliamos o desempenho de uma instância com apenas um processador ao trabalhar com folga de diferentes quantidades de memória.

A Figura 7.33 mostra a variação da quantidade de memória alocada a cada instância. No eixo *x* temos o valor do parâmetro *–executor-memory* e no eixo *y* o tempo médio de execução. O desvio padrão é ilustrado pelas linhas verticais presentes em cada uma das barras. Como as partições são pequenas e cabem todas em memória, a variação é minima entre os tempos, no entanto podemos observar que com a menor quantidade de memória a média do tempo de execução é maior.

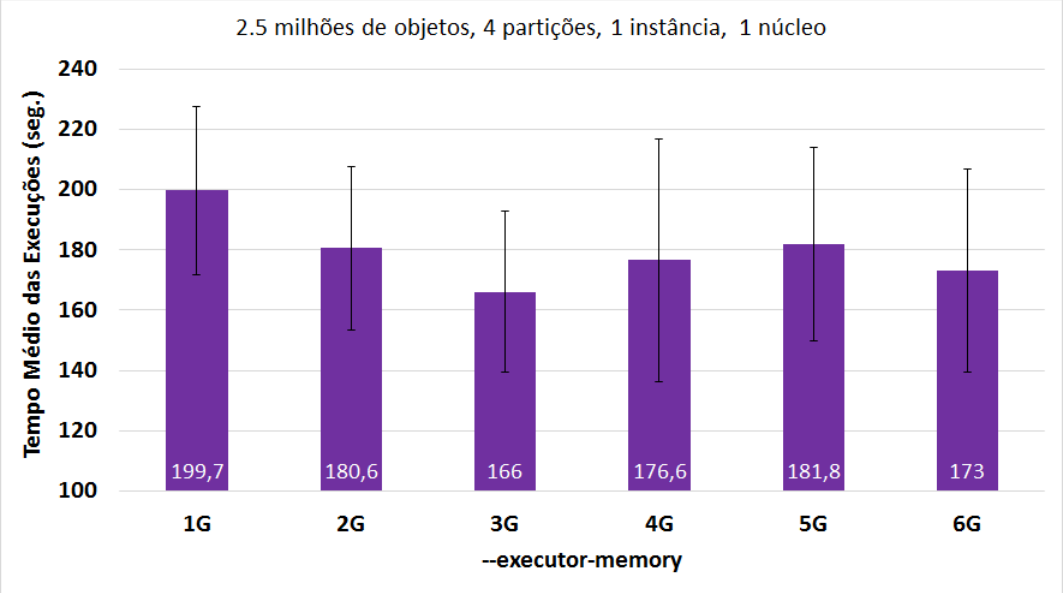


Figura 7.33: Quantidade (*–executor-memory)* x Tempo Médio de Execução

Fizemos um segundo experimento, semelhante ao primeiro desta seção, no qual fixamos os mesmos parâmetros, e usamos um conjunto de catálogos maior, com 952 MB de tamanho e 30 milhões de objetos. Variamos apenas a quantidade de memória de 1G a 6G por instância.

Apresentamos os resultados do segundo experimento na Figura 7.34. Os melhores tempos ocorrem quando escolhemos uma quantidade de memória intermediária, ou seja, 4G ou 5G. Como os nós têm 8G de memória RAM, escolher 6G para uma instância pode sobrecarregar o sistema operacional. Quando o *executor-memory* tem 3G ocorre o erro ”*cluster.YarnClusterScheduler: Lost executor*“ no estágio 3, e as etapas do estágio são refeitas em outra instância. O YARN é configurado para “matar”

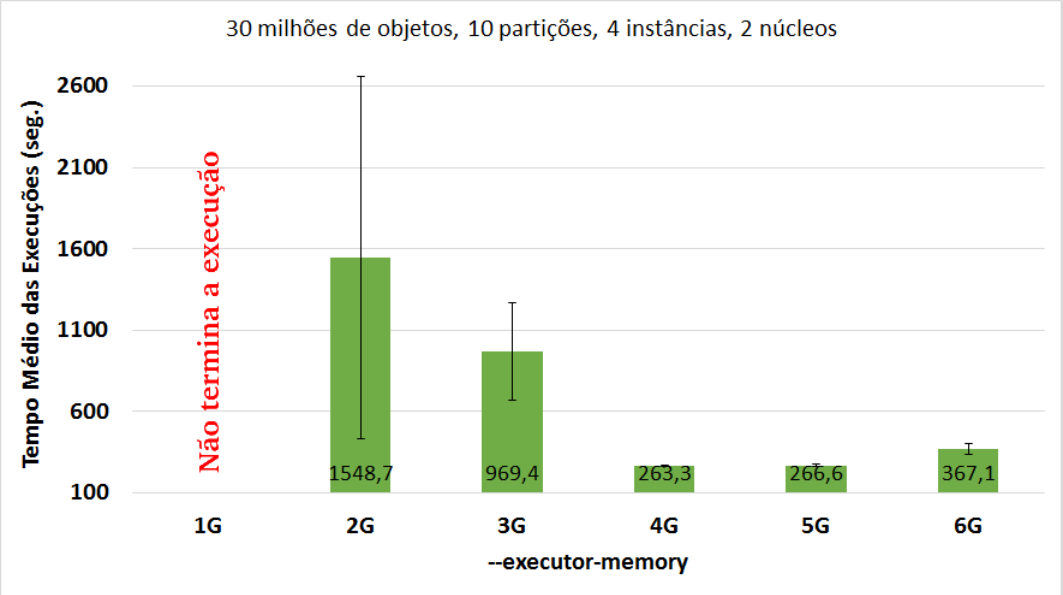


Figura 7.34: Quantidade (*–executor-memory)* x Tempo Médio de Execução (2º Experimento)

automaticamente a instância quando ela extrapola a memória destinada à ela. Apesar de um tempo de execução alto e com erros, o sistema consegue recuperar-se das falhas e executar por completo a aplicação.

Ao termos 2G alocado para cada instância, também temos erros no estágio 3 devido à pouca memória disponível, e a execução demora mais para terminar. Quando o *executor-memory* tem 1G os erros se tornam mais frequentes e começam no estágio 2. Assim, a aplicação fica em um ciclo sem fim de execução, apresentando erros e tentando se recuperar.

Na Seção 7.5 afirmamos que (KUMAR, 2016) recomenda não configurar o *–executor-memory* com mais de 40G de memória, pois pode levar a grandes pausas com a coleta de lixo (*Garbage Collector*). Como nosso *cluster* não tem toda essa capacidade de memória, não pudemos realizar este teste para verificar a veracidade desta informação.

7.5.2.6 Escalabilidade: Volume x Tempo de execução

Em todos os experimentos com o ParallelNACluster das seções acima nas quais variamos os parâmetros spark, utilizamos apenas conjuntos de catálogos com no máximo 30 milhões de objetos. A razão está em que os testes foram realizados em um cluster barato, e precisávamos realizar pelo menos 10 execuções usando cada configuração da aplicação para calcular o tempo médio, ou seja, como utilizamos 65 configurações diferentes, seriam no mínimo 650 execuções. Portanto, executar gran-

des conjuntos de catálogos 650 vezes poderia levar em torno de 2 anos para terminar de executar em nosso cluster, como veremos nos experimentos desta seção.

O objetivo dos próximos experimentos é avaliar a escalabilidade do ParallelNACluster, ou seja, se ele está preparado para o crescimento do volume de dados. Usamos quatro diferentes conjuntos de catálogos com as seguintes quantidades aproximadas de objetos:

- 947 milhões;
- 1,18 Bilhão;
- 1,41 Bilhão;
- 1,65 Bilhão;

E fixamos as seguintes configurações no Spark:

- 500 partições
- –num-executors 4
- –executor-cores 2
- –excutor-memory 5G

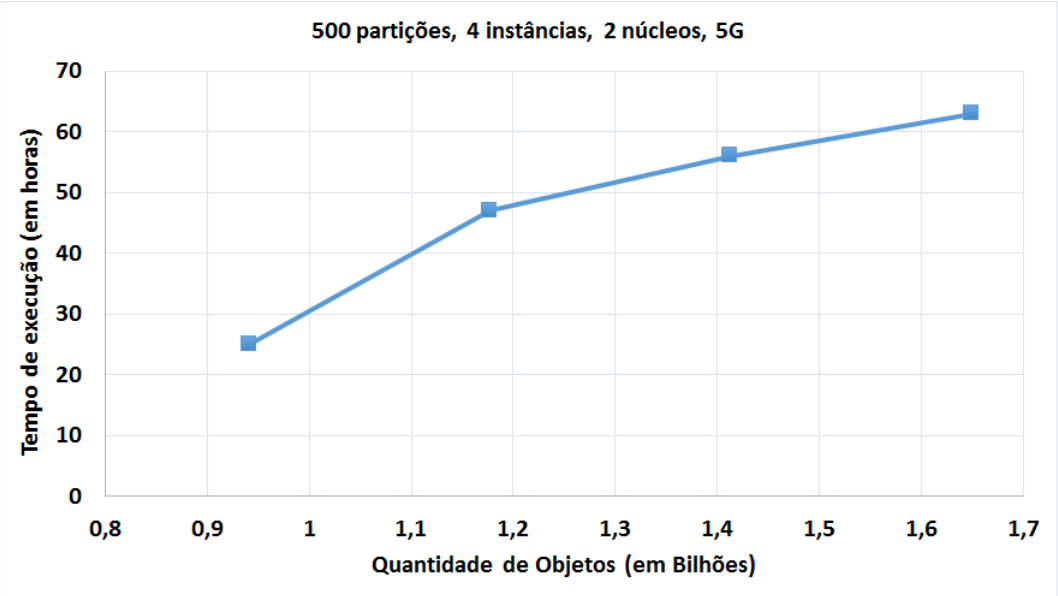


Figura 7.35: Volume x Tempo de uma Execução

Na Figura 7.35 apresentamos o tempo das execuções. No eixo *x* temos a quantidade de objetos e no eixo *y* o tempo de execução. Observamos que mesmo

aumentando o tamanho do conjunto de catálogos para grandes volumes de dados, na ordem de bilhões de objetos, a aplicação executa por completo. É importante ressaltar que neste caso não temos a média do tempo, pois a aplicação demora de um a dois dias para executar grandes volumes de dados em nosso pequeno cluster, e problemas como energia e rede, impossibilitaram realizar dez rodadas para cada conjunto de catálogos.

Concluímos que mesmo com grandes volumes de dados é possível encontrar um equilíbrio entre o número de partições, o volume de dados e a memória disponível para cada instância. Logo, podemos crescer o tamanho do conjunto de catálogos contanto que ajustemos os parâmetros do Spark para um valor adequado ao tamanho dos dados. Em nossos experimentos encontramos uma escalabilidade linear até 1,7 bilhão de objetos em um cluster barato formado por apenas 4 máquinas.

7.6 Experimentos com o AODP

No Capítulo 6 apresentamos o AODP, um *workflow* que organiza todas as etapas necessárias para a execução do ParallelNACluster. Ele realiza o pré-processamento do conjunto de catálogos, que inicialmente está em disco local, de forma que este conjunto possa ser executado pelo FRANCE, em ambiente centralizado, para identificar suas fronteiras e ser usado como entrada do ParallelNACluster em ambiente distribuído.

Os 650 experimentos feitos com o ParallelNACluster neste capítulo foram executados através do AODP, pois caso fizéssemos o pré-processamento dos arquivos de entrada manualmente, gastaríamos muito tempo e precisaríamos de muita atenção na manipulação destes arquivos. Qualquer erro comprometeria o resultado final do casamento, conforme explicado na Seção 6.2.2 do Capítulo 6.

Nesta seção estamos interessados em discutir a proporção do tempo que cada operador do AODP usa para executar. Alguns parâmetros influenciam diretamente nesta proporção. Eles são:

- **Tamanho do conjunto de catálogos:** influencia diretamente no tempo de execução dos operadores *FRANCE*, *CopyToHdfs* e *ParallelNACluster*. Quanto maior o conjunto de catálogos, maior o tempo de execução dos três operadores.
- **Número de partições:** influencia no tempo de execução dos operadores *FRANCE* e *ParallelNACluster*. Quanto maior o número de partições, mais tempo o *FRANCE* irá demorar para terminar sua execução. E, como visto na Seção 7.5.2.1, a quantidade de partições tem um grande impacto no tempo médio de execução do

ParallelNACluster. Quando o número de partições é menor que a quantidade de instâncias disponíveis para o processamento dos dados e quando essa quantidade é exagerada, o tempo de execução do operador tende a ser alto. Quanto ao *CopyToHdfs*, o número de partições gera uma mudança de poucos *bytes* no arquivo gerado pelo *FRANCE*, e a diferença de tempo na cópia deste arquivo para o *HDFS* é muito pequena.

- **Parâmetros usados na submissão da aplicação Spark (*–num-executors, –executor-memory, –executor-cores*):** como visto na Seção 7.5, eles influenciam diretamente no tempo de execução do operador *ParallelNACluster*.

Na Seção 7.5, a maioria dos experimentos foram feitos com dois conjuntos de catálogos. Um com 2,5 milhões de objetos e outro com 30 milhões. Na Figura 7.36 apresentamos a proporção da média do tempo gasto por cada operador do *AODP* na execução do primeiro conjunto de catálogos com as seguintes características:

- 134 MB de tamanho;
- 4 partições;
- *–num-executors 4*
- *–executor-memory* 5G
- *–executor-cores* 2

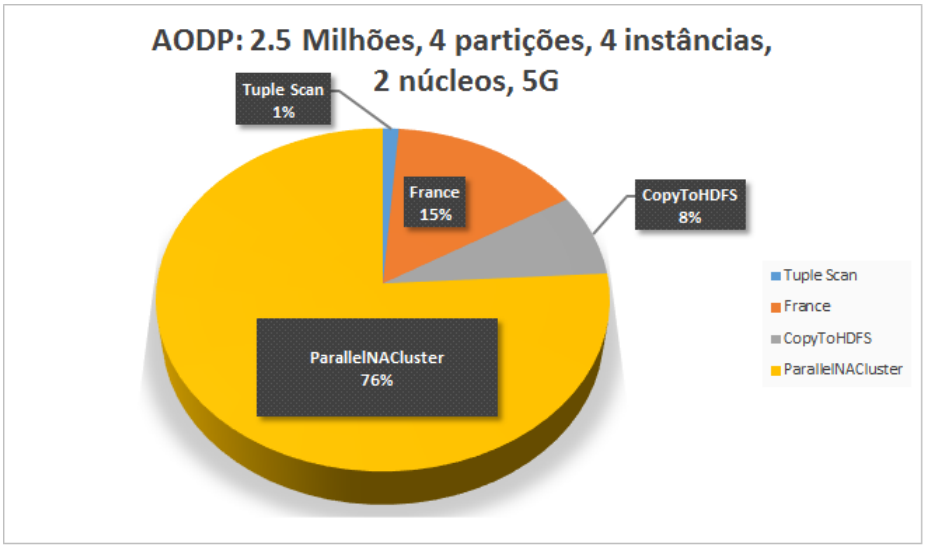


Figura 7.36: Proporção do tempo gasto por cada operador do AODP no primeiro experimento

Como o operador *TupleScan* é responsável apenas por obter o *dataset* do disco local e o deixar disponível para os demais operadores, ele gasta, em média, apenas 1% to tempo de execução do AODP para este conjunto de catálogos. E como o arquivo do conjunto de catálogos é pequeno, comparado aos de grandes volumes de dados, o FRANCE leva 15% do tempo e o CopyToHdfs 8%. A maior fatia do tempo gasto é na execução do ParallelNACluster. Essa fatia ainda poderia ser maior se outros parâmetros do Spark fossem escolhidos, como por exemplo, uma quantidade menor de instâncias e núcleos.

Na Figura 7.37 apresentamos a proporção da média do tempo gasto por cada operador do *AODP* na execução do segundo conjunto de catálogos com as seguintes características:

- 952 MB de tamanho;
- 10 partições;
- *–num-executors 4*
- *–executor-memory* 5G
- *–executor-cores* 2

Observamos que arquivos de tamanho diferente geram proporções desiguais de tempo. Neste segundo experimento, por termos um conjunto de catálogos aproximadamente sete vezes maior, a fatia de tempo correspondente ao operador *FRANCE* aumentou de 15% para 34% ao compararmos com o experimento anterior. A fração de tempo do operador CopyToHdfs aumentou de 8% para 14%. No entanto, a fatia de tempo dos operadores ParallelNACluster e TupleScan diminuíram. O ParallelNACluster diminuiu de 76% para 52% e o TupleScan de 1% para aproximadamente 0%.

Concluímos que fatores como o tamanho do conjunto de catálogos, o número de partições e os parâmetros da aplicação *spark* impactam diretamente no desempenho do AODP.

7.7 Considerações Finais

Neste capítulo apresentamos os experimentos com o NACluster, ParallelNACluster e AODP. Primeiramente discutimos como simulamos o catálogo de referência (*Golden Standard*) e os conjuntos de catálogos usados como entrada nas implementações e utilizados para avaliar a qualidade dos resultados.

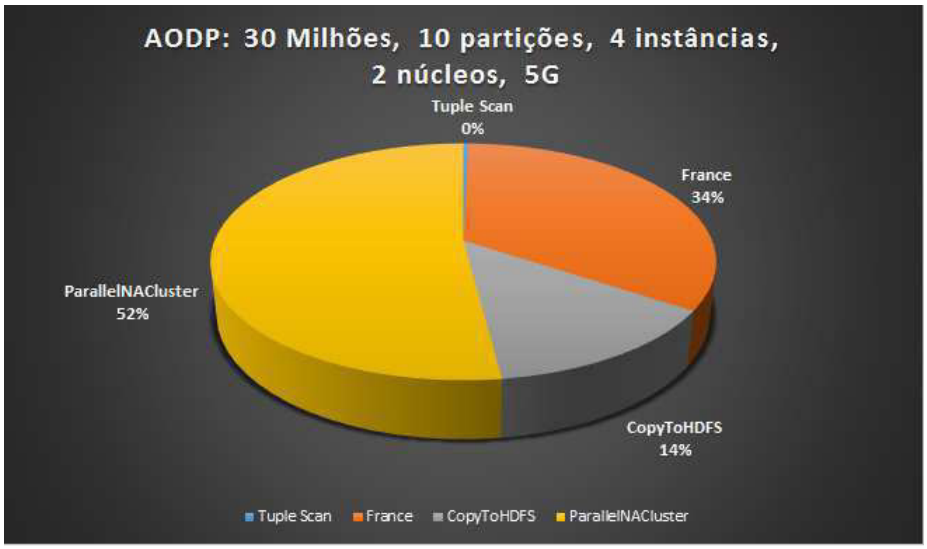


Figura 7.37: Proporção do tempo gasto por cada operador do AODP no segundo experimento

Em seguida, avaliamos o casamento do NACluster aplicado a diversos conjuntos de catálogos através da precisão, abrangência e medida-F. Nos *datasets* utilizados as medidas de qualidade permanecem próximas de 0,999. Esse alto valor ocorre pois a grande maioria dos objetos possuem apenas um centroide no raio de busca ε, ou seja, eles possuem apenas um cluster para alocá-lo, e o casamento tende a ser correto.

Propomos a métrica grau médio de ambiguidade para avaliarmos a ambiguidade do ponto de vista dos objetos, ou seja, o quanto cada objeto é ambíguo em relação aos clusters. Concluímos que quanto maior o grau médio de ambiguidade, menor é a qualidade do resultado do NACluster.

Também foram realizados experimentos para medir o tempo de execução do *NACluster* em relação ao aumento do volume do conjunto de catálogos. Concluímos que a relação *volume x tempo* tem tendência linear.

Em outro experimento, realizamos uma análise para verificar se os centroides iniciais influenciam na saída do NACluster. Os resultados mostram que a escolha dos centroides iniciais influencia na qualidade do resultado. E que o melhor resultado é encontrado quando inicia-se com o maior catálogo, assim como acontece por padrão no NACluster. No entanto, a qualidade do casamento é independente da escolha do catálogo utilizado na inicialização dos centroides, desde que ocorra uma segunda iteração do NACluster.

Na seção seguinte, fizemos uma comparação entre um algoritmo de *cross-matching* bastante utilizado na comunidade, o *Q3C Join*, e o NACluster. Concluímos

que, diferente da estratégia do algoritmo NACluster de dar uma solução para a ambiguidade dos casamentos, o *Q3C Join* tem uma estratégia de preservá-la, aceitando duplicidade nos casamentos. Essa característica faz com que o resultado do *Q3C* seja bastante dependente do raio utilizado. Caso seja utilizado um raio maior que a maioria das distâncias entre objetos "iguais" de diferentes catálogos ou eles sejam densos, a saída pode conter muito ruído. No *NACluster* o aumento de ε não piora a qualidade do casamento. É importante mencionar que, apesar do NACluster não preservar a ambiguidade no resultado, a saída da implementação mostra, além dos clusters, os outros objetos com potencial a serem alocados em cada cluster para análise manual por parte do usuário.

Também realizamos experimentos com o ParallelNACluster. Avaliamos a qualidade do seu casamento, bem como do casamento produzido pela fronteira. Concluímos que a qualidade dos seus casamentos são semelhantes aos do NACluster. Além disso, analisamos o impacto da mudança dos parâmetros do Apache Spark na submissão da aplicação distribuída e testamos a escalabilidade ao aumentarmos o volume de dados. Dentre as conclusões, vimos que precisamos encontrar um equilíbrio entre o número de partições, o volume dos dados e a quantidade de instâncias e núcleos para tirar o melhor desempenho do ParallelNACluster.

E, finalmente, fizemos testes com o AODP e discutimos a proporção de tempo que cada operador usa para executar e identificamos os parâmetros que influenciam diretamente nesta proporção. Concluímos que fatores como o tamanho do conjunto de catálogos, o número de partições e os parâmetros da aplicação *spark* impactam diretamente no desempenho do AODP.

8 CONCLUSÕES E TRABALHOS FUTUROS

Neste capítulo sintetizamos as conclusões e trabalhos futuros desta tese.

8.1 Conclusões

Dentre as principais contribuições desta tese temos o *NACluster*, um algoritmo de clusterização para realizar casamentos entre vários catálogos e solucionar o problema de resolução de entidades espaciais. O objetivo é agrupar os objetos dos catálogos em agrupamentos contendo potenciais casamentos. Os clusters formados são utilizados para determinar a identidade de uma entidade e não devem possuir mais de um objeto do mesmo catálogo. Assim, não se sabe antecipadamente quantos clusters serão produzidos ao final da execução. Um exemplo das diferenças de nosso trabalho em relação aos relacionados é o fato de usarmos técnicas de clusterização para realizar o casamento entre *n* catálogos espaciais. Outro exemplo é fato de termos proposto uma versão centralizada e uma distribuída com escalabilidade.

Nos experimentos, avaliamos a qualidade do casamento do NACluster aplicado a diversos conjuntos de catálogos através da precisão, abrangência e medida-F. Nos *datasets* utilizados as medidas permanecem próximas de 0,999. Esse alto valor ocorre pois a grande maioria dos objetos possuem apenas um centroide no raio de busca ε, ou seja, eles possuem apenas um cluster para alocá-lo, e o casamento tende a ser correto.

Para entendermos melhor esse resultado e encontrarmos alguma correlação entre a qualidade da clusterização do NACluster e a ambiguidade existente no agrupamento, propomos a métrica grau médio de ambiguidade para avaliarmos a ambiguidade do ponto de vista dos objetos, ou seja, o quanto cada objeto é ambíguo em relação aos clusters. Os experimentos nos fizerem concluir que a técnica tem um limite de aplicação em relação ao nível de ambiguidade e não é aplicável a qualquer cenário. Independente disso, o NACluster adiciona ao estado da arte o tratamento correto quando múltiplos catálogos precisam ser casados e não apresenta problemas de transitividade. Para os catálogos reais utilizados, o grau médio de ambiguidade fica entre 0 e 1 e a taxa de acerto é entre 99% e 100%. Vale ressaltar que dado um conjunto de catálogos, é possível separar os dados e aplicar o NACluster onde tem menor ambiguidade. E em trabalhos futuros, podemos adicionar na função de distância outros atributos e o NACluster tenderá a funcionar bem em regiões ambíguas.

Também foram realizados experimentos para medir o tempo de execução

do *NACluster* em relação ao aumento do volume do conjunto de catálogos. Concluímos que a relação *volume x tempo* tem tendência linear.

O NACluster sempre define seus centroides iniciais como sendo a posição dos objetos do maior catálogo, pois sua quantidade corresponde ao número mínimo de clusters do resultado final. Para saber se a escolha dos centroides iniciais influencia na saída do algoritmo, modificamos a implementação do NACluster para permitir escolher o catálogo no qual as posições dos seus objetos serão inicializados como centroides. Realizamos experimentos com o NACluster modificado variando todas as possibilidades de inicialização dos centroides. Os resultados mostraram que a escolha dos centroides iniciais influencia na qualidade do resultado. E que o melhor resultado é encontrado quando inicia-se com o maior catálogo, assim como acontece por padrão no NACluster. No entanto, a qualidade do casamento é independente da escolha do catálogo utilizado na inicialização dos centroides, desde que ocorra uma segunda iteração do NACluster. Mas ela não é necessária, pois inicializar os centroides pelos objetos do maior catálogo economiza esta iteração.

Também fizemos experimentos comparando um algoritmo de *cross-matching* bastante utilizado na comunidade, o *Q3C Join*, com o NACluster. O Q3C Join, como a maioria dos algoritmos de *cross-matching*, realiza o casamento de apenas um par de catálogos por vez, produzindo, como saída, um catálogo contendo os objetos que casam entre si. Com o objetivo de identificar a diferença de comportamento do NACluster e do *Q3C Join*, comparamos as saídas geradas por eles tendo como entrada dois catálogos. Concluímos que, diferente da estratégia do algoritmo NACluster de fornecer uma solução para a ambiguidade dos casamentos, o *Q3C Join* tem uma estratégia de preservá-la, aceitando duplicidade nos casamentos. Essa característica faz com que o resultado do *Q3C* seja bastante dependente do raio utilizado. Caso seja utilizado um raio maior que a maioria das distâncias entre objetos “iguais” de diferentes catálogos ou eles sejam densos, a saída pode conter muito ruído. No *NACluster* o aumento de ε não piora a qualidade do casamento. É importante mencionar que, apesar do NACluster não preservar a ambiguidade no resultado, a saída da implementação mostra, além dos clusters, os outros objetos com potencial a serem alocados em cada cluster para análise manual por parte do usuário.

O NACluster possui limitações físicas para processar grandes volumes de dados, pois precisa de computadores com bastante memória RAM para ser executado, algo que nos dias de hoje custa muito caro. Para baratear o processo e facilitar seu uso na comunidade que necessita processar grandes volumes de dados, propomos o ParallelNACluster, um algoritmo de clusterização que executa de forma distribuída e propõe-se a dar resultados semelhantes ao NACluster.

A solução paralela proposta requer alguns cuidados especiais. O primeiro deles, o particionamento, precisa preservar a vizinhança e manter o balanceamento das partições. Para atender este requisito, usamos o FRANCE (FRAgmeNtador de Catálogos Espaciais) para realizar o particionamento em uma dimensão. O segundo cuidado especial é com as fronteiras produzidas pelo particionamento. Segundo o algoritmo do NACluster, conhecer a vizinhança é necessário para realizar o casamento. No entanto, particionar os dados com o FRANCE, faz com que a vizinhança da fronteira de duas partições vizinhas seja separada. Portanto, conhecer as fronteiras e os objetos distantes até ε de centroides localizados do outro lado da fronteira torna-se essencial para que a execução da clusterização distribuída produza um resultado semelhante à da centralizada. Para esse tratamento especial das fronteiras propomos o *SCIBoundary*, um algoritmo recursivo que tem por objetivo encontrar os clusters e objetos influenciados pela fronteira, para que estes sejam mapeados para uma mesma partição e sejam executados pelo NACluster juntos em uma etapa *reduce*.

Para automatizar e acelerar a execução do pré-processamento dos dados, propomos o *AODP*, um workflow para particionamento dos dados em disco local e execução do casamento dos mesmos em ambiente distribuído através do *ParallelNACluster*.

Realizamos experimentos com a execução da implementação do *ParallelNACluster* feita para Apache Spark e avaliamos a qualidade do seu casamento através da precisão, abrangência e medida-f. Nos *datasets* utilizados, essas medidas de qualidade permaneceram próximas de 0,99, mesmo com o aumento do volume de dados e da quantidade de catálogos participantes do casamento. Esses valores de qualidade são semelhantes aos encontrados na execução dos mesmos *datasets* no *NACluster*. Também medimos a qualidade de casamentos de objetos influenciados pela fronteira produzidos através do *SCIBoundary* e concluímos que é semelhante ao do NACluster, com uma pequena diferença devido aos tamanhos das amostras (nos *datasets* testados (os objetos da fronteira correspondiam em torno de 0,14% do total).

Vimos que o desempenho da execução de uma aplicação em Apache Spark depende de vários fatores. Dentre eles: o tamanho do *dataset* de entrada; o número de partições ao qual o *dataset* é dividido; a quantidade de instâncias; a quantidade de núcleos de CPU; a quantidade de memória por instância. Logo, analisamos o impacto das mudanças nos parâmetros do Apache Spark na submissão do *ParallelNACluster* sobre o *YARN* e dentre as conclusões, concluímos que não devemos ter uma quantidade de partições menor que a de instâncias disponíveis para o processamento dos dados, bem como não podemos exagerar na quantidade de partições, pois pode aumentar consideravelmente o tempo de execução da aplicação. E, conforme mostrado

em (VELLORE, 2015), não existe um número de partições ideal, ele deve ser encontrado a medida que for executando e obter um bom equilíbrio de simultaneidade e tempo de execução das tarefas. Além disso, o tamanho da partição não deve ultrapassar a memória alocada para uma instância, pois caso isso ocorra a instância recebe um *kill* do *YARN*. Portanto, concluímos que é preciso encontrar um equilíbrio entre o número de partições, o volume dos dados e a quantidade de instâncias e núcleos para tirar o melhor desempenho do *ParallelNACluster*. E que mesmo com grandes volumes de dados podemos encontrar esse equilíbrio.

Quanto ao AODP discutimos a proporção de tempo que cada operador usa para executar e identificamos os parâmetros que influenciam diretamente nesta proporção. Concluímos que fatores como o tamanho do conjunto de catálogos, o número de partições e os parâmetros da aplicação *spark* impactam diretamente no desempenho do AODP.

Durante o desenvolvimento deste trabalho publicamos dois artigos. O primeiro deles apresenta o levantamento do estado da arte em casamentos de catálogos astronômicos e foi publicado no *VII Brazilian e-Science workshop* (BreSci), em 2013 (FREIRE et al., 2013). O segundo artigo, publicado no *10th IEEE International Conference on e-Science* em 2014 (FREIRE et al., 2014), apresenta o NACluster, alguns experimentos e uma comparação com entre o algoritmo proposto e o *Q3c_Join*. As demais contribuições desta tese estão sendo organizadas para serem submetidas à um periódico.

8.2 Trabalhos Futuros

Durante o desenvolvimento da tese, tentamos resolver o problema de resolução de entidades a partir de múltiplos catálogos espaciais como um todo e por isso algumas partes podem ser melhoradas e serão estudadas em trabalhos futuros. Além disso, algumas ideias interessantes surgiram, sem, no entanto, haver tempo hábil para investigá-las. Abaixo, listamos as nossas sugestões de trabalhos futuros:

- realizar experimentos com o ParallelNACluster em hardwares de alto desempenho;
- alterar a estratégia de particionamento visando reduzir o numero de cluster na fronteira;
- criar uma estratégia para tratar dois clusters que possuem o mesmo centroide;

- adicionar na função de distância outros atributos com o intuito de melhorar a qualidade do casamento nas regiões ambíguas;
- criar uma estratégia para calcular as probabilidades de casamento;
- desenvolver um método para tratar casamentos de catálogos com escalas diferentes e permitir múltiplos *matchings*;

O NACluster bem como os outros algoritmos da literatura, apresentados no Capítulo 4, realizam casamentos de catálogos utilizando somente a posição dos objetos em duas dimensões. Em 2019, com a chegada do LSST, essa abordagem não será suficiente.

O LSST (*Large Synoptic Survey Telescope*) (Ivezic et al., 2008) é o levantamento astronômico mais ambicioso atualmente planejado. Ele irá mapear todo o céu visível profundamente a cada semana com a sua câmera digital de três bilhões de pixels. O LSST iniciará as operações em 2019 e entrará em funcionamento completo a partir de 2022. Em seu primeiro mês de operação, obterá mais imagens do Universo do que todos os telescópios atualmente em operação somados. Sua câmera abrirá uma janela quase em tempo real para a monitoria de objetos que se movem ou que variam seu brilho. Sua excelente qualidade de imagem permitirá um mapeamento do cosmos em 3D com profundidade e nitidez inigualados. Com um mapeamento completo de todo o céu do hemisfério sul a cada 4 dias, o LSST fornecerá dados em tempo real para os astrônomos e para o público (Ivezic et al., 2008).

Como outro trabalho futuro relevante, sugerimos uma solução para a resolução de entidades no contexto do LSST, a partir da complementação do algoritmo NACluster para aceitar posições de objetos em três dimensões (incluindo a profundidade), bem como o aspecto temporal capturado pelo LSST, estendendo nossa estratégia para o novo contexto onde os objetos mudam de posição ao longo do tempo, totalizando assim, uma clusterização de objetos com 4 dimensões.

REFERÊNCIAS BIBLIOGRÁFICAS

AMÔRES, E. B. d.; ALEMAN, I. G. *Redshift e Lei de Hubble*. 2016. Disponível em: <http://www.telescopiosnaescola.pro.br/hubble.pdf>.

AYAT, N.; AKBARINIA, R.; AL. et. *Entity Resolution for Uncertain Data*. [S.l.]: University of Amsterdam, 2012.

BELKIN, N. J.; CROFT, W. B. Information filtering and information retrieval: two sides of the same coin? *Commun. ACM*, ACM, New York, NY, USA, v. 35, n. 12, p. 29–38, 1992. ISSN 0001-0782.

BERKHIN, P. A Survey of Clustering Data Mining Techniques. *Grouping Multidimensional Data*, p. 25–71, 2006.

BILENKO, M.; MOONEY, R.; AL. et. Adaptive name matching in information integration. *IEEE Intelligent Systems*, p. 2–9, Sept/Oct 2003.

BUDAVÁRI, T.; SZALAY, A. S. Probabilistic Cross-Identification of Astronomical Sources. *The Astrophysical Journal*, n. 1, p. 301, 2008.

BURCHAT, P. *TED Talks - Patricia Burchat: Shedding light on dark matter*. 2008. Acessado em Agosto de 2016. Disponível em: <https://www.ted.com/talks/patricia_burchat_leads_a_search_for_dark_energy>.

CDS, C. d. D. a. d. S. *VizieR Service*. 2016. Disponível em: <http://stupendous.rit.edu/tass/catalogs/ubvri.tass>.

COHEN, W. W.; KAUTZ, H.; MCALLESTER, D. Hardening soft information sources. In: *The Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2000)*. [S.l.: s.n.], 2000. p. 255–259.

DAED. *Astrofísica Geral 2013*. [S.l.], 2013. Acessado em Outubro de 2013. Disponível em: <http://www.on.br/ead_2013/>.

DAI, B.-R.; LIN, I.-C. Efficient Map/Reduce-Based DBSCAN Algorithm with Optimized Data Partition. In: *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*. Washington, DC, USA: IEEE Computer Society, 2012. p. 59–66. ISBN 978-0-7695-4755-8.

DU, P. et al. New cross-matching algorithm in large-scale catalogs with ThreadPool technique. *Science China: Physics, Mechanics and Astronomy*, v. 57, n. 3, p. 577–583, 2014. ISSN 16747348.

EL-SONBATY, Y.; ISMAIL, M.; FAROUK, M. An efficient density based clustering algorithm for large databases. In: *16th IEEE International Conference on Tools with Artificial Intelligence*. [S.l.]: IEEE Comput. Soc, 2004. p. 673–677. ISBN 0-7695-2236-X.

ELMAGARMID, A. K.; MEMBER, S. Duplicate Record Detection: A Survey. *Knowledge and Data Engineering, IEEE Transactions on*, v. 19, n. 1, p. 1–16, 2007.

ESTER, M. et al. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: SIMOUDIS, E.; HAN, J.; FAYYAD, U. M. (Ed.). *KDD*. [S.l.]: AAAI Press, 1996. p. 226–231. ISBN 1-57735-004-9.

FAN, D. et al. Efficient Catalog Matching with Dropout Detection. *Publications of the Astronomical Society of the Pacific*, v. 125, n. 924, p. 218–223, feb 2013. ISSN 00046280. Disponível em: <http://www.jstor.org/stable/info/10.1086/669707>.

FEKETE, G. *Hierarchical Triangular Mesh*. may 2007. `http://www.skyserver.org/htm/index.html`.

FELLEGI, I. P.; SUNTER, A. B. A Theory for Record Linkage. *Journal of the American Statistical Association*, v. 64, p. 1183–1210, 1969.

FINCH, C. T.; Zacharias, N.; Wycoff, G. L. UCAC3: Astrometric Reductions. v. 139, p. 2200–2207, jun 2010.

FIOC, M. Probabilistic positional association of catalogs of astrophysical sources: the Aspects code. 2012.

FONTANA, A.; NALDI, M. C. Estudo e comparação de métodos para estimação de números de grupos em problemas de agrupamento de dados. In: ICMC. [S.l.], 2009. ISSN 0103-2569.

FREIRE, V. P. et al. Algoritmos de cross-matching de dados astronômicos. In: *BreSci - VII Brazilian e-Science workshop. Proceedings of the XXXIII Congress of the Brazilian Computer Society (CSBC 2013)*. [S.l.]: SBC, 2013. p. 1870–1875. ISSN 2175-2761 / 9 772175 276009.

FREIRE, V. P. et al. NACluster: A Non-supervised Clustering Algorithm for Matching Multi Catalogues. In: *2014 IEEE 10th International Conference on e-Science*. [S.l.]: IEEE, 2014. p. 83–86. ISBN 978-1-4799-4288-6.

FRUCHTER, A. et al. *Equatorial coordinates Declination and Right Ascension*. 2000. Acessado em Agosto de 2016. Disponível em: <http://hubblesite.org/newscenter/archive/releases/2000/07>.

FU, B.; FINK, E.; AL. et. Fast Approximate Matching of Astronomical Objects. *2012 IEEE International Conference on Cluster Computing Workshops*, Ieee, set. 2012.

GAEDE, V.; GUNTHER, O. Multidimensional Access Methods. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 30, n. 2, p. 170–231, jun 1998. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/280277.280279>.

GASPAR, D.; PORTO, F. *A Multi-Dimensional Equi-Depth Partitioning Strategy for Astronomy Catalog Data*. 2014.

GORSKI, K. M. *HEALPix*. `http://healpix.jpl.nasa.gov/`. Em 09-04-2013.

GORSKI, K. M.; ARBALLO, J. K. *Healpix*. Nasa. Acessado em Outubro de 2013. Disponível em: <http://healpix.jpl.nasa.gov/>.

GÓRSKI, K. M.; HIVON, E.; WANDELT, B. D. Analysis Issues for Large CMB Data Sets. *Evolution of Large Scale Structure*, n. August, p. 6, dec 1998. Disponível em: <http://arxiv.org/abs/astro-ph/9812350>.

GRAY, J.; SZALAY, A. S.; AL. et. There Goes the Neighborhood: Relational Algebra for Spatial Data Search. *CoRR*, cs.DB/0408031, 2004.

HASSANZADEH, O. et al. Framework for Evaluating Clustering Algorithms in Duplicate Detection. In: *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB 2009)*. [S.l.: s.n.], 2009.

HASSANZADEH, O.; MILLER, R. J. Creating probabilistic databases from duplicated data. *The VLDB Journal*, v. 18, n. 5, p. 1141–1166, aug 2009. ISSN 1066-8888. Disponível em: <http://link.springer.com/10.1007/s00778-009-0161-2>.

HAVELIWALA, T. H.; GIONIS, A.; INDYK, P. Scalable Techniques for Clustering the Web. In: *WebDB (Informal Proceedings)*. [S.l.: s.n.], 2000. p. 129–134.

HERNáNDEZ, M. A.; STOLFO, S. J. Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Min. Knowl. Discov.*, v. 2, n. 1, p. 9–37, 1998.

Ivezic, Z. et al. LSST: from Science Drivers to Reference Design and Anticipated Data Products. *ArXiv e-prints*, 2008.

KANDEFER, M.; SHAPIRO, S. An f-measure for context-based information retrieval. In: *Commonsense 2009: the Ninth International Symposium on Logical Formalizations of Commonsense Reasoning*. [S.l.]: The Fields Institute, 2009.

KARAU, H. et al. *Learning Spark: Lightning-Fast Big Data Analytics*. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2015. ISBN 1449358624, 9781449358624.

KESTELYN, J. *Working with Apache Spark: Or, How I Learned to Stop Worrying and Love the Shuffle*. 2015. Acessado em Junho de 2016. Disponível em: <http://blog.cloudera.com/blog/2015/05/working-with-apache-spark-or-how-i-learned-to-stop-worrying-and-love-the-shuffle/>.

KOPOSOV, S.; BARTUNOV, O. Q3C , Quad Tree Cube – The new Sky-indexing Concept for Huge Astronomical Catalogues and its Realization for Main Astronomical Queries (Cone Search and Xmatch) in Open Source Database PostgreSQL. *The Astronomical Data Analysis Software and Systems (ADASS) conference*, v. 351, p. 735–738, 2006.

KOPOSOV, S.; BARTUNOV., O. *Q3C*. 2012. `http://code.google.com/p/q3c/`.

KRUEGER, P.; CHAWLA, R. The Stealth Distributed Scheduler. In: *11th Intl. Conf. Distributed Comput. Syst. (ICDCS)*. [S.l.: s.n.], 1991. p. 336–343.

KUMAR, S. *Achieving Optimal Performance with Apache Spark1.5*. 2016. Acessado em Junho de 2016. Disponível em: <http://datarpm.com/resources/achieving-optimal-performance-with-apache-spark1-5/>.

KUNSZT, P. Z.; Szalay, A. S.; Thakar, A. R. The Hierarchical Triangular Mesh. In: Banday, A. J.; Zaroubi, S.; Bartelmann, M. (Ed.). *Mining the Sky*. [S.l.: s.n.], 2001. p. 631.

KWON, Y. et al. Scalable clustering algorithm for N-body simulations in a shared-nothing cluster. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 6187 LNCS, p. 132–150, 2010. ISSN 03029743.

LANG, K. *Astrophysical formulae: a compendium for the physicist and astrophysicist*. [S.l.]: Springer-Verlag, 1974. (Astronomy and astrophysics library). ISBN 9783540099338.

LIM, E.-P.; SRIVASTAVA, J.; AL. et. Entity identification in database integration. *Proceedings of IEEE 9th International Conference on Data Engineering*, IEEE Comput. Soc. Press, p. 294–301, 1993.

MACQUEEN, J. B. Some Methods for Classification and Analysis of MultiVariate Observations. In: CAM, L. M. L.; NEYMAN, J. (Ed.). *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*. [S.l.]: University of California Press, 1967. v. 1, p. 281–297.

MAGNANI, M.; MONTESI, D. A Survey on Uncertainty Management in Data Integration. v. 2, n. 1, 2010.

MANNING, C. D.; RAGHAVAN, P.; SCHüTZE, H. *Introduction to Information Retrieval*. [S.l.]: Cambridge University Press, 2008. ISBN 0521865719.

MATTEUCCI, M. *A Tutorial on Clustering Algorithms*. Politecnico di Milano, 2013. Disponível em: <http://home.dei.polimi.it/matteucc/Clustering/tutorial_-html/index.html>.

MATTOS, G. d. O. *Aspectos de desempenho da computação paralela em clusters e grids para processamento de imagens*. Tese (Doutorado) — Universidade Federal de Pernambuco, Recife, 2008.

MENESTRINA, D.; BENJELLOUN, O.; GARCIA-MOLINA, H. *Generic Entity Resolution with Data Confidences*. [S.l.], 2005.

MENESTRINA, D.; WHANG, S. E.; GARCIA-MOLINA, H. Evaluating Entity Resolution Results. *Proc. VLDB Endow.*, VLDB Endowment, v. 3, n. 1-2, p. 208–219, sep 2010. ISSN 2150-8097.

MOLNÁR, A. J.; BENCZÚR, A. A.; SIDLÓ, C. I. Flexible and Efficient Distributed Resolution of Large Entities. In: ____. *Foundations of Information and Knowledge Systems: 7th International Symposium, FoIKS 2012, Kiel, Germany, March 5-9, 2012. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 244–263. ISBN 978-3-642-28472-4. Disponível em: <http://dx.doi.org/10.1007/978-3-642-28472-4_-14>.

NEWCOMBE, H. B. Record Linking: The Design of Efficient Systems for Linking Records into Individual and Family Histories. *American Journal of Human Genetics*, v. 19, n. 3, p. 335–359, May 1967.

NEWCOMBE, H. B. *Handbook of Record Linkage*. [S.l.]: Oxford University Press, 1988.

NEWCOMBE, H. B.; KENNEDY, J. M. Record linkage: making maximum use of the discriminating power of identifying information. *Commun. ACM*, ACM, New York, NY, USA, v. 5, n. 11, p. 563–566, nov. 1962. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/368996.369026>.

NEWCOMBE, H. B.; KENNEDY, J. M.; AL. et. Automatic Linkage of Vital Records. *Science*, American Association for the Advancement of Science, v. 130, n. 3381, p. 954–959, oct 1959. ISSN 1095-9203. Disponível em: <http://dx.doi.org/10.1126/science.130.3381.954>.

NG, R. T.; HAN, J. Efficient and Effective Clustering Methods for Spatial Data Mining. In: *Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994. (VLDB 94), p. 144–155. ISBN 1-55860-153-8.

OCHI, L. S.; DIAS, C. R.; SOARES, S. S. F. *Clusterização em Mineração de Dados*. [S.l.], 2004.

O'MULLANE, W. et al. Mining the Sky: Proceedings of the MPA/ESO/MPE Workshop Held at Garching, Germany, July 31 - August 4, 2000. In: ____. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. cap. Splitting the Sky - HTM and HEALPix, p. 638–648. ISBN 978-3-540-44665-1. Disponível em: <http://dx.doi.org/10.1007/10849171_84>.

ORTIZ, P. F. Why Indexing the Sky is Desirable. In: Payne, H. E.; Jedrzejewski, R. I.; Hook, R. N. (Ed.). *Astronomical Data Analysis Software and Systems XII*. [S.l.: s.n.], 2003. (Astronomical Society of the Pacific Conference Series, v. 295), p. 35.

PENCHIKALA, S. *Big Data com Apache Spark - Parte 1: Introdução*. 2015. Acessado em Junho de 2016. Disponível em: <https://www.infoq.com/br/articles/apache-spark-introduction>.

PINTO, P. M. T. L. N. *Uma avaliação de desempenho dos ambientes de programação paralela Hadoop e Spark*. Dissertação (Mestrado) — Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, 2015.

PORTO, F. et al. Qef-supporting complex query applications. In: IEEE. *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*. [S.l.], 2007. p. 846–851.

REDORBIT. *Redshift*. 2016. Disponível em: <http://www.redorbit.com/reference/redshift/>.

ROHDE, D.; GALLAGHER, M.; DRINKWATER, M. Astronomical catalogue matching as a mixture model problem. *AIP Conference Proceedings*, v. 1490, 2012.

ROHDE, D. J. et al. Applying machine learning to catalogue matching in astrophysics. *Monthly Notices of the Royal Astronomical Society*, v. 360, n. 1, p. 69–75, jun. 2005. ISSN 00358711.

RYZA, S. *How-to: Tune Your Apache Spark Jobs (Part 1)*. 2015. Acessado em Junho de 2016. Disponível em: <http://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-1/>.

RYZA, S. *How-to: Tune Your Apache Spark Jobs (Part 2)*. 2015. Acessado em Junho de 2016. Disponível em: <http://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/>.

SAMIRALOW, A. *Great Moments in Science and Technology - Edwin Powell Hubble and the Expanding Universe*. 2007. DVD Video.

SANTIAGO, B.; SALVIANO, A. *Astronomia Geodésica. Posicionamento pelas Estrelas*. [S.l.: s.n.], 2013. 1–166 p.

SARAWAGI, S.; BHAMIDIPATY, A. Interactive Deduplication using Active Learning. In: *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*. [S.l.: s.n.], 2002. p. 269–278.

SDSS. *Informações sobre astronomia*. [S.l.], 2013. Em 05-10-2013. Disponível em: <http://cas.sdss.org/dr7/pt/astro/>.

SKRUTSKIE, M.; CUTRI, R. M.; AL. et. The Two Micron All Sky Survey (2MASS). *AJ*, v. 131, n. 2, p. 1163–1183, fev. 2006.

Sutherland, W.; Saunders, W. On the likelihood ratio for source identification. *Monthly Notices of the Royal Astronomical Society*, v. 259, p. 413–420, dec 1992.

SZALAY, A. S. et al. Indexing the Sphere with the Hierarchical Triangular Mesh. *CoRR*, 2005.

TEPPING, B. J. A Model for Optimum Linkage of Records. *Journal of the American Statistical Association*, v. 63, n. 324, p. 1321–1332, December 1968.

THURMOND, R. *A History of Star Catalogues*. [S.l.], 2003. 1–55 p.

UNCU, Ö. et al. Gridbscan: Grid density-based spatial clustering of applications with noise. In: *SMC*. [S.l.]: IEEE, 2006. p. 2976–2981. ISBN 1-4244-0099-6.

VELLORE, T. *Performance Tuning Tips for Running Spark Applications*. 2015. Acessado em Junho de 2016. Disponível em: <http://techkites.blogspot.com.br/2015/04/performance-tuning-tips-for-running.html>.

VISWANATH, V. *Spark RDDs Simplified*. 2015. Acessado em Junho de 2016. Disponível em: <http://vishnuviswanath.com/spark_rdd.html>.

WANG, D. et al. Qserv: A distributed shared-nothing database for the lsst catalog. In: *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*. [S.l.: s.n.], 2011. p. 1–11.

WANG, Y. R.; MADNICK, S. E. The Inter-Database Instance Identification Problem in Integrating Autonomous Systems. In: *Proceedings of the Fifth IEEE International Conference on Data Engineering (ICDE 1989)*. [S.l.: s.n.], 1989. p. 46–55.

WAY, M. J. et al. *Advances in machine learning and data mining for astronomy*. Boca Raton, FL: CRC Press, 2012. (Chapman & Hall/CRC data mining and knowledge discovery series). ISBN 978-1-4398-4173-0. Disponível em: <http://opac.inria.fr/record=b1133742>.

WRIGHT, E. L. et al. The Wide-field Infrared Survey Explorer (WISE): Mission Description and Initial On-orbit Performance. v. 140, p. 1868–1881, dez. 2010.

ZAHARIA, M. et al. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2012. (NSDI'12).

ZÄSCHKE, T.; ZIMMERLI, C.; NORRIE, M. C. The PH-tree: a space-efficient storage structure and multi-dimensional index. In: *SIGMOD'14 , International Conference on Management of Data*. New York, NY, USA: [s.n.], 2014. p. 397–408. ISBN 978-1-4503-2376-5. Disponível em: <http://doi.acm.org/10.1145/2588555.2588564>.

ZHAO, Q. et al. A Paralleled Large-Scale Astronomical Cross-Matching Function. In: HUA, A.; CHANG, S.-L. (Ed.). *ICA3PP*. [S.l.]: Springer, 2009. (Lecture Notes in Computer Science, v. 5574), p. 604–614. ISBN 978-3-642-03094-9.

ZHU, M. *Recall, Precision and Average Precision*. [S.l.], ago. 2004. Disponível em: <http://www.stats.uwaterloo.ca/stats_navigation/techreports/04WorkingPapers/2004-09.pdf>.

APÊNDICE A – DEFINIÇÕES

Definição 1 (Resolução de Entidade) Dado um conjunto de registros $R = \{r_1, r_2, ...r_m\}$, onde cada r_j consiste de um conjunto de atributos. O objetivo da resolução de entidades é particionar os registros de acordo com as entidades as quais pertencem: Seja $E = \{e_1, e_2, ...e_n\}$ um conjunto de entidades, onde cada e_i consiste de um subconjunto de registros $e_i \subseteq R$ tal que a união das entidades cobre todos os registros, $\cup_{i=1}^{n} e_i = R$, e nenhum registro pertence a mais de uma entidade: $r \in e_i \wedge r \in e_j \Rightarrow i = j$ (MOLNÁR; BENCZÚR; SIDLÓ, 2012).

Definição 2 (Resolução de Entidade Espacial) Dado um conjunto $S = \{C_1, ...C_n\}$ com n catálogos espaciais C_i, $1 \leq i \leq n$, tal que $C_i = \{o_{1i}, o_{2i}, ..., o_{mi}\}$, onde o_{ji}, $1 \leq j \leq m$, é um objeto espacial do catálogo i; o resultado da resolução de entidade $ER(S)$ é o conjunto E, tal que $E = \{e_1, e_2, ..., e_k\}$, com $k \geq max\ |C_i|$, onde $|C_i|$ corresponde ao número de objetos em um catálogo C_i, e e_p, $1 \leq p \leq k$, é uma entidade e representa um objeto espacial. Adicionalmente, $e_p = \{o_{i1}, o_{i2}, ..o_{ir}\}$, tal que $r \leq n$ e para cada catálogo C_j existe no máximo um $o_{ij} \in e_p$, onde $1 \leq i \leq m$.

Definição 3 (Função de Distância) Uma função binária d correspondente à distância Euclidiana entre objetos de catálogos e entidades é definida como $d : C_i \times E \longrightarrow \mathbb{R}$.

Definição 4 (Função de Casamento) Uma função injetora M de mapeamento entre catálogos e entidades é definida como $M : C_i \longrightarrow E$, tal que, para cada $o_{ji} \in C_i$ e para todo $1 \leq i \leq n$, existe um $e_p \in E$, onde $d(o_{ji}, e_p) \leq \varepsilon$ e ε corresponde ao erro máximo de localização entre objetos de dois catálogos. Adicionalmente, $d(o_{ji}, e_p)$ é o valor mínimo dentre todas as entidades e_p.

Proposição 1 Determinar M tal que a Definição 4 seja verdade para um conjunto S e um dado valor de ε.

Invariante 1 Sejam $o_i \in C_i$ e $q_j \in C_j$ dois objetos espaciais. Seja ainda uma função M, conforme a Definição 4. Desta forma, $M(o_i) = M(q_j) = e_p$ se e somente se $C_i \neq C_j$.

Definição 5 (Função de Mapeamento entre Catálogos e Entidades) Uma função $Cand$ de mapeamento entre catálogos e entidades é definida como $Cand : C_i \longrightarrow E$, tal que, para cada $o_{ji} \in C_i$ e para todo $1 \leq i \leq n$, existe um ou mais $e_p \in E$, onde $d(o_{ji}, e_p) \leq \varepsilon$ e ε é a distancia máxima.

Definição 6 (Grau de Ambiguidade) Seja $|Cand(o_{ji})| = a$, o seu *grau de ambiguidade* é $G(o_{ji}) = a - 1$.

Definição 7 (Grau Médio de Ambiguidade) Seja $\sum_{i=1}^{n} |C_i|$ a quantidade total de objetos do conjunto de catálogos, o grau médio de ambiguidade é $\overline{\mathrm{G}} = \frac{\sum_{j=1}^{m} \sum_{i=1}^{n} G(o_{ji})}{\sum_{i=1}^{n} |C_i|}$, tal que m é a quantidade total de objetos do catálogo i.

Definição 8 (Partição de um Catálogo) Uma partição P_i de um catálogo C representa um subconjunto de C de tal forma todos os objetos em P_i encontram-se entre duas fronteiras espaciais f_i e f_{i+1}, de tal forma que a coordenada espacial associada a f_i tem um valor inferior à coordenada espacial associada à f_{i+1}.

Definição 9 (Balanceamento entre partições) Dadas duas partições P_i e P_j, dizemos que estão balanceadas se o número de objetos em P_i, $\mid P_i \mid$, é aproximadamente igual ao de P_j, $\mid P_j \mid$. Logo, $\mid P_i \mid + \delta = \mid P_j \mid$, tal que δ é um valor tolerável na diferença da quantidade de objetos das duas partições.

Definição 10 (Particionamento) Dado um objeto $o_{ji} < x, y, a_1, a_2, ..., a_t >$ tal que a_z, $1 \leq z \leq t$, é um atributo de o_{ji}, e x e y correspondem às coordenadas espaciais; um particionamento P é uma lista de valores $< v_1, v_2, ..., v_g >$ tal que $v_h \in x$ e $1 \leq h \leq g$. Desta forma, uma partição P_r, tal que $1 \leq r \leq g - 2$ apresenta objetos vizinhos em uma região do espaço delimitada por P_r e P_{r+1}.

Definição 11 (Fronteira) Dado um conjunto de partições P, uma fronteira f_i contém um conjunto de objetos O_{f_i} cujos valores da coordenada x de particionamento estão entre $v_h - \varepsilon \leq f_h \leq v_h + \varepsilon$, para todo $v_h \in P$.

Definição 12 (Objeto da Fronteira) Seja O_{f_i} uma fronteira, um objeto da fronteira é o_l tal que $o_l \in O_{f_i}$.

Definição 13 (Objeto influenciado pela fronteira) Seja uma entidade $e_p \subset O_{f_i}$ e um objeto $o_l \in O$ e $o_l \notin O_{f_i}$. o_l é influenciado pela fronteira $O_{f_i} \iff d(o_l, e_p) \leq \varepsilon$, onde d é a função de distância definida na Definição 3.

Definição 14 (Cluster influenciado pela Fronteira) Seja o_l um objeto influenciado pela fronteira O_{f_i}. Uma entidade e_j é um cluster influenciado pela fronteira O_{f_i} se $o_l \in e_j$.

Definição 15 (Grafo de Clusters (GC)) Um GC é um grafo $G = (V, E)$ de tal forma que V é um conjunto de vértices, e cada vértice $v \in V$ corresponde a um cluster, e as arestas $e \in E$, aos pares de clusters. Em um GC, uma aresta existe entre dois vértices v_i e v_j se existe um objeto $o_i \in v_i$ tal que a distancia$(o_i, v_j) < \varepsilon$, onde v_j é representado por seu centroide no cálculo da distância.