Um novo algoritmo de clustering para a organização tridimensional de dados de expressão gênica

Tiago José da Silva Lopes

Um novo algoritmo de clustering para a organização tridimensional de dados de expressão gênica

Tiago José da Silva Lopes

Orientador: *Prof. Dr. Guilherme P. Telles*

Monografia apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Fevereiro/2007

Resumo

Neste trabalho desenvolvemos um novo algoritmo para clustering para dados de expressão gênica. As abordagens tradicionais utilizam um conjunto de dados na forma de uma tabela de duas dimensões, onde as linhas são os genes e as colunas são as condições experimentais. Nós utilizamos uma estrutura de três dimensões, acrescentando fatias de tempo. Implementamos nosso algoritmo e testamos com conjuntos de dados sintéticos e dados reais, usando índices de validação para comparar os resultados obtidos pelo nosso algoritmo com os resultados produzidos pelo algoritmo TriCluster. Os resultados mostraram que o nosso algoritmo é bom para dados de expressão gênica em três dimensões e pode ser aplicado a dados de outros domínios.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo

1

Introdução

Neste capítulo descreveremos brevemente a motivação para utilizar técnicas de *clustering* em problemas de expressão gênica. Também apresentaremos o projeto desenvolvido neste estudo de mestrado, assim como sua relevância para a área de pesquisa. Além disso, no final do capítulo apresentaremos a organização dos demais assuntos deste trabalho.

1.1 Motivação

Com a crescente capacidade de armazenamento e transmissão de informações, a criação de conjuntos de dados cada vez maiores tornou-se uma prática comum. Dessa maneira, para extrair informações úteis, que ajudem a confirmar ou criar hipóteses, são necessários mecanismos eficientes de análise de dados. Dentre várias técnicas possíveis da estatística e da computação destacamos a classificação. Classificar significa, estabelecer segundo um determinado critério, grupos de objetos similares entre si e diferentes de objetos pertencentes a outros grupos. Nessa tarefa existem duas alternativas: a classificação supervisionada e a não-supervisionada. Algoritmos de classificação supervisionada podem ser utilizados quando possuímos um conjunto de dados para o qual a classificação correta dos objetos é conhecida. Assim, ao receber um novo objeto como entrada, o algoritmo o classifica como membro do grupo com o qual ele tiver maior similaridade. Já os algoritmos de classificação não-supervisionada são utilizados quando não há conhecimento prévio das classes existentes em um conjunto de dados. Por isso, quando um algoritmo dessa categoria recebe um conjunto de entrada, ele busca encontrar as classes em que os objetos se dividem levando em conta apenas o conhecimento disponível no próprio conjunto de dados. A classificação não-supervisionada é conhecida na literatura como *clustering* e consiste em encontrar grupos de objetos similares entre si (*clusters*) em um conjunto de dados.

Enquanto as técnicas de clustering e seus algoritmos estão intimamente ligados à computação, a biologia molecular faz uso de uma técnica chamada *microarray*. Sua função é medir o nível de expressão de um conjunto de genes em diferentes condições experimentais. Neste caso, nível de expressão está relacionado ao nível de atividade de um gene.

Conhecida por diferentes nomes, como chip de DNA ou microarranjos, a popularidade desta tecnologia tem crescido devido à queda nos preços de equipamentos e insumos, além da ampla difusão de protocolos experimentais.

A base dessa técnica é depositar em uma lâmina de vidro os fragmentos de diferentes genes de interesse, selecionados criteriosamente a fim de verificar mudanças em seu nível de expressão em diferentes condições experimentais. Além disso, é necessário preparar duas amostras contendo genes das condições de interesse. Vamos considerar por exemplo pessoas saudáveis e pessoas com câncer. Os genes das células são marcados com corante verde para pacientes saudáveis e vermelho para pacientes com câncer, e uma lâmina com milhares de genes impressos é colocada em contado com a mistura destas duas substâncias simultaneamente. Onde houver ligação forte entre os genes das diferentes amostras e os genes da lâmina, uma reação química ocorrerá e uma fluorescência poderá ser vista (através de um scanner especialmente desenvolvido para este propósito). Após esse processo teremos três possibilidades de cor para cada gene da lâmina (chamado de *spot*): verde, vermelho e amarelo. As duas primeiras cores indicam que o gene foi diferencialmente expresso em apenas uma das duas condições estudadas, e spots amarelos indicam que o gene está expresso nas duas condições experimentais. Além disso, spots com as cores branca ou preta indicam que não houve atividade do gene em nenhuma amostra. O maior interesse dos usuários dessa técnica é avaliar quais são os genes diferencialmente expressos (verdes ou vermelhos), e utilizar essas informações para inferir hipóteses a respeito dos processos celulares estudados. O próximo passo da técnica é quantificar o nível de fluorescência de cada spot e criar uma tabela onde as linhas representam os genes e as colunas representam seu nível de expressão nas diferentes codições experimentais. Nessa tabela, que normalmente contém milhares de linhas e diversas colunas, tornou-se comum utilizar técnicas de clustering para encontrar grupos de genes com um comportamento similar, em outras palavras, um padrão de expressão semelhante ao longo das diferentes condições experimentais. Com esses padrões é possível aos profissionais de biologia molecular e áreas afins entender e formular hipóteses a respeito de processos celulares.

Objetivo

Neste trabalho de mestrado foi desenvolvido um novo algoritmo de clustering de dados de microarray em três dimensões.

Como mencionamos anteriormente, todos os estudos criam tabelas para descrever os resultados da técnica de microarray onde as colunas podem ser:

- condições experimentais distintas, como pessoas saudáveis e pessoas portadoras de alguma doença ou
- avaliações em intervalos de tempo, como aos 5, 10, 15 e 20 minutos.

Neste estudo, diferentemente dos estudos tradicionais, utilizamos uma estrutura de três dimensões. Essa estrutura é uma matriz contendo nas linhas, os genes, nas colunas as diferentes condições experimentais e na terceira dimensão, diferentes fatias de tempo.

Com o algoritmo desenvolvido é possível encontrar *grupos de genes* que possuem *comportamento similar* nas diferentes *condições experimentais ao longo do tempo*. Com este conhecimento os pesquisadores terão acesso à informações complementares que podem ajudar a entender fenômenos anteriormente incompreensíveis devido a falta de integração entre os dados experimentais e temporais.

Para avaliar a eficácia do método proposto, o algoritmo de Zhao e Zaki [30], chamado *TriCluster* foi estudado e os resultados de ambos foram comparados através de diversos índices de qualidade.

Organização do Trabalho

Esta dissertação está organizada da seguinte maneira. No capítulo 1 temos esta introdução, deixando claro o conteúdo do trabalho. No capítulo 2 apresentamos uma revisão bibliográfica das mais importantes técnicas de clustering e sua relação à técnica de microarray. Neste segundo capítulo incluímos também a descrição do funcionamento do TriCluster, um algoritmo capaz de realizar agrupamento em três dimensões. No capítulo 3 apresentamos os detalhes do algoritmo desenvolvido neste trabalho. No capítulo 4 apresentamos os experimentos realizados, descrevendo sua motivação e planejamento. No capítulo 5 mostramos os resultados obtidos com a execução dos dois algoritmos e suas vantagens e desvantagens.

Capítulo

2

Agrupamento de dados

Neste capítulo forneceremos inicialmente uma descrição da técnica de *microarray*, incluindo sua função, importância e mecanismo de operação. Assim ficará clara a inspiração para utilizar as técnicas de clustering descritas neste capítulo em bases de dados geradas por essa técnica. Em seguida abordaremos os assuntos clustering, incluindo clustering em três dimensões, clustering de dados de microarray e validação de clustering.

2.1 Microarray

Microarray é uma técnica que permite avaliar o nível de expressão de milhares de genes simultaneamente. Isso é feito através da combinação de técnicas tradicionais da biologia molecular com avanços que permitem monitorar a expressão de genes sob determinadas condições. Assim, podemos realizar experimentos que submetem células ou tecidos a diferentes tipos de tratamento e obter um perfil global do nível de expressão de seus genes [2].

A maior parte dos experimentos é realizada através da comparação de uma condição controle a diversas situações experimentais. Por exemplo, podem ser utilizadas como controle as células de pacientes saudáveis e como condições de estudo as células de pacientes em diferentes estágios de uma doença. Isso permite a verificação de quais genes são mais ou menos ativos em diferentes estágios da enfermidade.

Podemos dividir a operação desta técnica em algumas etapas principais:

- na primeira etapa, são escolhidos genes que possuem relação com o estudo a ser desenvolvido. Por exemplo, para estudar o metabolismo de ferro no organismo são selecionados genes conhecidos por desempenharem algum papel nessa atividade. Além disso, são escolhidos outros genes candidatos a sofrerem alteração em seu nível de expressão quando submetidos a determinados tratamentos, e genes que com certeza não sofrerão alteração significativa sob estas condições (chamados de genes “housekeeping”). Este conjunto de genes selecionados é fixado por um robô

(*arrayer*) em uma lâmina de vidro ou membrana de nylon, formando uma matriz de pontos [12].

- Na segunda etapa, é extraído o RNA mensageiro (mRNA) das células submetidas a diferentes tratamentos, como pacientes que ingeriram drogas para alterar o metabolismo de ferro. Com esse material é feita uma reação envolvendo a enzima transcriptase reversa para gerar o DNA complementar (cDNA). Um fato importante nesta etapa é que se o número de cópias de mRNA de um determinado gene for grande, o número de cópias de seu cDNA também será e este princípio também vale para um número pequeno de cópias [12]. O material genético correspondente a cada condição (saudável/doente) é marcado com um corante diferente (verde ou vermelho). Estes corantes são chamados de Cy3 e Cy5, respectivamente.

  Logo após, a superfície preparada contendo genes impressos é colocada em contato com a mistura de materiais genéticos corados sob condições ideais de pH e temperatura. Ocorre então uma série de reações de hibridização entre o DNA imobilizado no array e as moléculas de cDNA coradas. É importante notar que isto é uma hibridização concorrente, pois estão presentes os materiais genéticos das duas condições experimentais avaliadas e por isso os genes com maior número de cópias têm mais chance de ligarem-se aos genes imobilizados na superfície.

- Após a hibridização, essa superfície é colocada em um scanner laser que detecta a intensidade luminosa de cada ponto (spot) da lâmina. Pontos de maior intensidade em uma determinada cor representam maior número de hibridizações e, conseqüentemente, maior número de cópias daquele gene presente na respectiva condição experimental (maior expressão). Essa etapa gera imagens (figura 2.1) cuja intensidade de cores será quantificada, dando origem a tabelas numéricas contendo nas linhas a identificação de cada gene impresso, nas colunas a identificação de cada condição estudada e em cada célula desta tabela, um valor representando a intensidade de cada ponto da superfície [11].

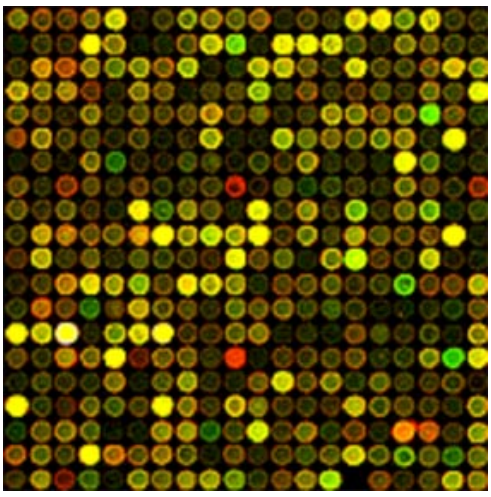

Figura 2.1: Imagem produzida pela técnica de microarray.

- Esses dados devem ser normalizados para eliminar as variações experimentais sistemáticas causadas por influências físicas e químicas inerente à técnica. Após a normalização espera-se que a única variação presente nos dados seja causada pela diferença entre os níveis de expressão dos genes estudados.

- Para detectar os genes diferencialmente expressos o trabalho de Tusher *et al.* [27] definiu uma técnica que tornou-se a mais utilizada em trabalhos que utilizam microarray. Este procedimento é bastante utilizado em uma etapa preliminar ao processo de clustering, pois além de reduzir bastante o número de objetos de um conjunto de dados, isso ainda garante apenas a análise de genes com diferenças entre as condições experimentais estudadas.

Pesquisas envolvendo microarray fazem uso de diferentes técnicas de clustering. Estas técnicas serão discutidas nas próximas seções.

2.2 Agrupamento de Dados

Vivemos em um mundo repleto de informações e que a todo momento nos apresenta novos objetos ou fenômenos que de alguma forma precisamos entender. Para isto, realizamos uma das mais primitivas e indispensáveis atividades que colaboraram no desenvolvimento dos seres humanos, a classificação. Para entender um novo objeto ou fenômeno, descrevemos suas principais características e o comparamos a outros objetos ou fenômenos conhecidos. E essa comparação é feita através de medidas de similaridade ou dissimilaridade, ambas geralmente referenciadas como *medidas de proximidade*.

O problema de clustering consiste em encontrar grupos de objetos similares entre si e diferentes dos objetos pertencentes a outros grupos. Na literatura e no restante deste trabalho, estes grupos de objetos são chamados de *clusters*.

A entrada para o problema pode ser de dois tipos: (1) uma matriz de atributos ou características M onde cada linha representa um objeto, cada coluna representa um atributo e cada célula m_{ij} tem o valor do atributo j para o objeto i ou (2) uma matriz quadrada S de similaridade, onde s_{ij} tem o valor de similaridade entre o objeto i e o objeto j. Alternativamente a matriz S pode ser uma matriz de dissimilaridade ou distância. Esses tipos de entrada podem ser convertidos entre si usando várias medidas e técnicas [10].

O foco deste projeto de mestrado são algoritmos de clustering. Nas próximas seções apresentaremos as etapas fundamentais de um estudo envolvendo técnicas de clustering, os mais influentes algoritmos e suas aplicações a dados de expressão gênica.

2.3 Etapas do Processo de Clustering

Podemos dividir o processo de clustering em 4 etapas, como mostra a figura 2.2. Como podemos ver nessa figura, a partir de qualquer etapa do processo é possível retornar a um dos passos anteriores para realizar modificações. Em experimentos envolvendo clustering, este processo de retornar à etapas anteriores e modificar algumas informações normalmente é feito de forma automática.

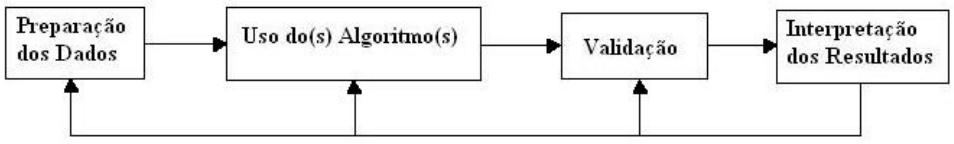

Figura 2.2: Principais etapas do processo de Clustering.

A preparação dos dados é a primeira e fundamental etapa do processo de clustering. Nela são escolhidos manualmente atributos ou características utilizados para melhor representar os objetos do conjunto de dados. Estes atributos são representados na forma de um vetor de características que podem assumir valores distintos para cada objeto. Esse processo pode ser aperfeiçoado através do uso de recursos como a seleção ou extração de características por algoritmos. A técnica de seleção de características consiste em escolher dentre diversos atributos quais são os melhores para representar os objetos a serem estudados, descartando as demais. A extração de características consiste em aplicar uma ou mais transformações nos dados para combinar ou destacar as características mais influentes do conjunto. Porém, segundo Parsons *et al.* [21] tanto a seleção quanto a extração de características têm o efeito colateral de causar a perda de informações potencialmente importantes.

Existem diferentes escalas e tipos de atributos para caracterizar um objeto. Esta variedade implica no uso de diferentes medidas de similaridade e diferentes algoritmos de clustering para tentar encontrar grupos em um conjunto de dados. Segundo Jain e Dubes [16], os atributos podem assumir as seguintes formas:

- **Binários:** atributos que podem ter apenas dois estados, como verdadeiro/falso, 0/1 ou sim/não.
- **Discretos:** atributos discretos apresentam apenas valores inteiros finitos como $(1,\ 2,\ 3,\ \ldots,\ n)$.
- **Contínuos:** os atributos contínuos representam valores reais, como $1.356,\ 25.20$ e -10.0445.

Podemos ainda representar dentro destas três categorias outros atributos, como:

- **Nominais:** atributos que possuem apenas valores diferentes, mas sem relação de ordem, como nomes, CEP e cores.
- **Ordinais:** valores que refletem uma ordem gradual, por exemplo, grau de escolaridade, hierarquia militar e níveis como, *Moderado*, *Médio* e *Avançado*.
- **Intervalo:** este tipo de atributo contém diferenças entre medidas, como a duração de um evento ou a diferença entre um evento e outro.

Após representarmos as informações é necessário escolher um algoritmo de clustering para tentar encontrar grupos de objetos no conjunto de dados. É importante mencionar que a escolha do algoritmo causa e sofre influência do passo anterior (representar os dados), pois todos os algoritmos têm exigências bem específicas a respeito do tipo de entrada, do tipo de informação a ser processada (atributos categóricos ou numéricos, por exemplo), além do fato de diferentes implementações precisarem de diferentes formatos para arquivos de entrada.

Para escolher um algoritmo de clustering é preciso levar em consideração a existência de várias alternativas, cada uma com suas peculiaridades e normalmente procurando suprir uma deficiência introduzida por outro algoritmo. O que precisa ficar claro na seleção dos algoritmos é que não existe um algoritmo ótimo para todos os casos e também não existe apenas uma resposta para o problema em questão [10]. Por isso, de acordo com Handl *et al.* [15], na prática é recomendável executar testes com diversos algoritmos e

avaliar comparativamente os resultados. Além disso, pode não haver apenas um algoritmo apropriado para o problema e sim uma combinação de soluções.

Dada a enorme quantidade de alternativas existentes, Berkhin [4] definiu algumas características a serem levadas em consideração para ajudar a escolher algoritmos de clustering:

- Tipo de entrada (matriz de similaridade/dissimilaridade ou matriz de atributos).
- Tipo de atributos que o algoritmo pode manipular.
- Escalabilidade para conjuntos de dados grandes.
- Habilidade para trabalhar com conjuntos de alta dimensionalidade.
- Habilidade para encontrar clusters com diferentes formas, se considerarmos cada objeto como um ponto no espaço numérico.
- Tolerância a ruído (*outliers*) nos dados.
- Complexidade de tempo e espaço.
- Dependência da ordem de entrada dos dados.
- Tipo de agrupamento criado ("Hard" vs. "Fuzzy", partição ou coleção de conjuntos, hierárquicos ou não-hierárquicos).
- Dificuldade em determinar os parâmetros de entrada do algoritmo.
- Forma de apresentação dos resultados.

Além disso, com a experiência adquirida durante o desenvolvimento deste trabalho, poderíamos recomendar a avaliação dos seguintes aspectos:

- *Tempo de execução para diferentes conjuntos de dados* - Como experimentos com clustering devem ser repetidos com diferentes parâmetros, execuções demoradas inviabilizam estudos que primam pela qualidade obtida através da alta granularidade nos experimentos.
- *Algoritmo com abordagem paralela* - Esta característica faz bastante diferença se forem estudados conjuntos de dados grandes e houver um cluster de máquinas ou máquinas de processamento paralelo disponíveis.
- *Algoritmo determininístico ou não* - Se o algoritmo for determinístico e independente da ordem de entrada, é confiável executá-lo apenas uma vez (a menos que o desempenho esteja sendo avaliado). Entretanto se o algoritmo não for determinístico e depender da ordem de entrada dos dados, é necessário executá-lo diversas vezes e obter o resultado médio das execuções. Com isso, o item relativo ao tempo de execução faz ainda mais diferença.

2.4 Algoritmos de Agrupamento

Existe uma grande discussão a respeito da classificação correta dos algoritmos de clustering descritos na literatura [28, 4, 17].

Porém, o único consenso é a divisão entre algoritmos particionantes e hierárquicos. Os algoritmos particionantes dividem o conjunto de dados em um certo número de partições (clusters) e os algoritmos hierárquicos constroem uma hierarquia de partições.

Nas próximas seções descreveremos os principais algoritmos das categorias hierárquicos e particionantes segundo a ordem proposta no trabalho de revisão de Xu e Wunsch II [28].

2.4.1 Algoritmos hierárquicos

Como mencionado anteriormente, estes algoritmos de clustering criam um conjunto de partições dos dados, organizadas de forma hierárquica e geralmente representadas por uma árvore binária ou dendrograma [10]. Eles podem ser divisivos ou aglomerativos. Os primeiros realizam uma divisão sucessiva de um único cluster com todos os objetos até chegar a clusters de apenas um elemento. Os segundos realizam a união sucessiva de clusters de apenas um objeto até restar apenas um cluster com todos os elementos do conjunto de dados.

O mais comum na prática é o uso de algoritmos aglomerativos, que partem de n clusters de apenas um elemento cada (onde n é o número de elementos do conjunto de dados) e realizam sucessivas uniões destes clusters até que uma hierárquia completa seja formada.

O algoritmo Hierarchical Clustering pode ser descrito pelos seguintes passos:

1. Começar com n clusters $C_1, \ldots, C_n$ de um único objeto e uma matriz simétrica M de tamanho $n \times n$, onde o elemento m_{ij} é a distância entre os clusters i e j.

2. Encontrar o par de clusters i e j tal que $D(C_i,\ C_j)$ é mínima, para uma função de distância D (normalmente a Euclideana).

3. Combinar os clusters C_i e C_j para formar um novo cluster.

4. Atualizar a matriz M calculando a distância do novo cluster para todos os outros.

5. Repetir itens 2, 3 e 4 até que todos os objetos estejam no mesmo cluster.

O critério de cálculo de distância entre clusters dos passos 2 e 4 possuem algumas alternativas:

- A chamada *single linkage* leva em consideração os objetos mais próximos que pertencem a dois clusters distintos para estabelecer o valor da distância entre esses clusters. Esse método é referenciado também como "vizinho mais próximo" (*nearest neighbor*). Um problema desse método é o efeito chamado *chaining*, que acaba produzindo clusters que são alongados e nem sempre satisfatórios.

- Outra alternativa é o método chamado *complete linkage*, que ao contrário do método anterior, considera a maior distância entre dois objetos que pertencem aos clusters para definir o valor da distância entre eles. Este método, também conhecido como vizinho mais distante (*farthest neighbor*), tem a característica de produzir clusters mais compactos que o método *single linkage*.

- Além disso, existe a opção de considerar os centróides dos clusters para calcular a distância entre eles. Esta alternativa chama-se *average linkage* e em diversos estudos foi observado que os agrupamentos utilizando este método apresentaram hierarquias mais úteis em relação aos agrupamentos criados usando os outros dois métodos [16].

Os algoritmos de clustering hierárquicos possuem algumas desvantagens. Dentre elas podemos mencionar o fato de que se um determinado objeto for associado a um cluster, ele não pode mais deixá-lo, pois o método hierárquico sofre do defeito de nunca poder consertar um erro cometido no passo anterior [10]. Outro problema diz respeito a sua falta de escalabilidade devido a sua complexidade $O(n^2)$ para a maioria de suas implementações conhecidas, o que limita sua aplicação a conjuntos de dados pequenos. Além disso, o método hierárquico é sensível a ruído (outliers) e tende a formar clusters hiper-esféricos, o que em alguns casos pode distorcer a estrutura presente nos dados [28].

2.4.2 Algoritmos particionantes

Ao contrário dos algoritmos hierárquicos, os algoritmos de clustering particionantes, ou baseados na otimização de uma função objetivo, dividem os dados em um conjunto de partições, sem nenhuma estrutura hierárquica. Isto é bom para os casos onde a criação de um dendrograma é computacionalmente proibitiva devido a um conjunto de dados muito grande [17].

As técnicas particionantes encontram clusters em um conjunto de dados através da otimização de uma função objetivo, como a minimização de *erros quadráticos*. Seria possível encontrar a separação ideal dos objetos do conjunto de dados através da tentativa de todas as possibilidades, porém este método força-bruta é inviável devido a seu custo computacional, dado que o número de partições de um conjunto cresce exponencialmente. O número de partições de um conjunto com n elementos é calculado pela recorrência $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$ [23].

A função de erros quadráticos de um agrupamento C de um conjunto de dados H de n elementos e contendo K clusters é:

$$e^2(C,\ H) = \sum_{j=1}^{K} \sum_{i=1}^{n_j} (x_i^j - c_j)^2$$

onde x_i^j é o valor do objeto i pertencente ao cluster j com centróide c_j e n_j é o número de elementos do cluster j.

O algoritmo K-means é a alternativa mais comum dessa categoria. Ele busca minimizar o valor da função de erros quadráticos descrita acima. Tendo K como parâmetro de entrada, seus principais passos são:

1. Escolher K objetos aleatoriamente no conjunto de dados para serem centróides.

2. Associar cada objeto, exceto os centróides, ao centróide mais próximo.

3. Recalcular os centróides baseando-se nas novas configurações dos clusters.

4. Se nenhuma mudança significativa na função de critério e^2 for notada, pare. Senão, volte ao passo 2.

O algoritmo K-means apresenta bons resultados na descoberta de clusters compactos e hiper-esféricos. Sua complexidade é da ordem de O(nKd), onde n é o número de objetos do conjunto de dados, K é o número de clusters a serem criados e d é o número de atributos do conjunto de dados. Como geralmente K e d são bem menores que n, este algoritmo é bastante apropriado para grandes volumes de dados.

Uma desvantagem do algoritmo K-means é a necessidade de definir o parâmetro K. Como a técnica de clustering normalmente é utilizada sem conhecimento prévio do conjuntos de dados, é difícil saber de antemão quantos clusters serão encontrados.

2.4.3 Clustering de dados de microarray

Dados de expressão gênica obtidos através da técnica de microarray são normalmente representados por uma tabela de *objetos* × *atributos*, onde cada célula dessa tabela possui um valor real relativo ao nível de expressão de um determinado gene.

A técnica de clustering tem sido aplicada aos dados de microarray para agrupar genes que possuem nível de expressão semelhante, e desta forma, sugerir que eles estão envolvidos em processos semelhantes nas células [31]. O primeiro trabalho envolvendo clustering de dados de microarray foi realizado por Eisen *et al.*, aplicando o clustering hierárquico a dados de expressão gênica de mais de 8000 genes humanos para categorizá-los de acordo com sua resposta temporal a soro em células de fibroblastos. Desde então, diversos trabalhos têm sido realizados envolvendo clustering de dados de microarray, tendo como objetivo facilitar a identificação de modelos de vias metabólicas e confirmar os modelos existentes [2]. A visualização dos resultados produzidos pelo processo de agrupamento aparece na figura 2.3, onde a intensidade das cores mostra o nível de expressão do gene em questão.

Em estudos de microarray tornou-se um padrão utilizar o *Hierarchical Clustering* ou o *K-means* [20]. Handl *et al.* [15] apontou a existência de alternativas cujos resultados demonstraram-se superiores aos dessas duas técnicas. No entanto esses métodos clássicos continuam sendo utilizados graças a sua simplicidade e a ampla disponibilidade de suas implementações.

2.4.4 Biclustering

As abordagens hierárquica e particionante podem ser utilizadas de duas maneiras distintas:

1. Encontrar grupos de objetos com comportamento similar ao longo das condições experimentais.

2. Encontrar grupos de condições experimentais onde os objetos têm um padrão similar de expressão.

Entretanto, ambas possuem um problema que é levar em conta todos os genes e todas as condições experimentais simultaneamente para encontrar clusters. Com a popularização da tecnologia de microarray e a disponibilidade de *chips* pré-fabricados contendo o mesmo conjunto de genes, está tornando-se cada vez mais comum a criação de conjuntos de dados com dezenas de condições experimentais e muitas vezes com resultados de grupos de pesquisa separados geograficamente. Por isso, diminui cada vez mais a chance de

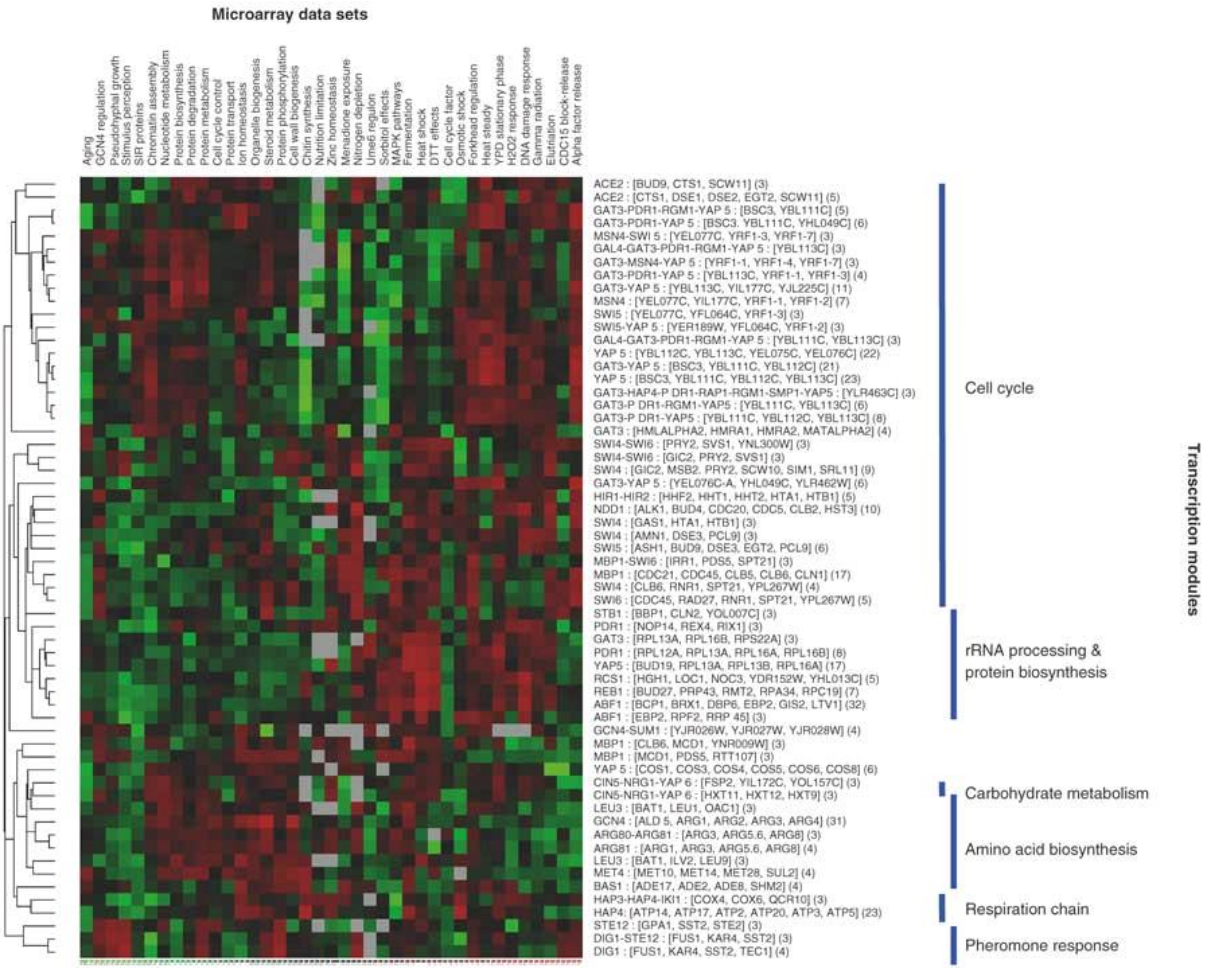

Figura 2.3: Visualização dos resultado obtido ao aplicar um algoritmo de clustering a dados de expressão gênica. Extraído de [25].

encontrar clusters cujos objetos possuem um comportamento similar ao longo de todas as condições estudadas.

Para lidar com esta questão, foi introduzido o conceito de *biclustering*, também conhecido na literatura por *coclustering*, *subspace clustering*, *projected clustering* e *bidimensional clustering*. Apesar de toda essa diferença na nomenclatura, todas dizem respeito a mesma formulação: encontrar clusters formados por sub-conjuntos de genes e sub-conjuntos de condições experimentais. No restante deste trabalho chamaremos estes sub-conjuntos de *sub-espaços*. Ou seja, os algoritmos membros dessa classe são capazes de formar grupos de objetos com comportamento similar apenas em alguns atributos.

Para encontrar os clusters verdadeiros nos dados (quando estes existirem), o ideal seria buscar em todos os sub-espaços possíveis e utilizar uma forma de validação para obter o sub-espaço com os melhores clusters [5]. Isto não é possível pois o problema de geração de sub-espaços é intratável. Esse problema diz respeito a realizar todas as combinações possíveis entre as diferentes dimensões e os diferentes objetos do conjunto de dados até encontrar o melhor agrupamento possível. Para isso uma heurística de busca eficiente é necessária, e cada um dos algoritmos existentes fornece uma alternativa diferente. Segundo Parsons *et al.* [21], a principal divisão entre os algoritmos existentes pode ser feita entre os que empregam a estratégia *bottom-up* e os que utilizam a estratégia *top-down*.

A primeira vertente, bastante semelhante aos algoritmos aglomerativos (Seção 2.4), iniciam sua operação a partir de atributos individuais (com apenas uma dimensão) e prosseguem unindo atributos até formar clusters de x dimensões, onde $1 \geq x \geq n$ e n é a quantidade de atributos do conjunto de dados. A outra vertente (*top-down*), segue um abordagem similar aos algoritmos divisivos (Seção 2.4), pois começam com clusters de n dimensões e os dividem até chegar a clusters de y dimensões, onde $1 \geq y \geq n$ e n é a quantidade de atributos do conjunto de dados.

2.4.5 Clustering em três dimensões

Nos estudos envolvendo a técnica de microarray para monitorar o nível de expressão gênica de um organismo, é comum o processo de agrupar genes com comportamento similar em condições experimentais. A maior parte desses agrupamentos é realizada em dois ambientes distintos, o primeiro consiste em avaliar o nível de expressão de um conjunto de genes ao longo de diferentes momentos do tempo como ilustrado na figura 2.4a. O segundo ambiente envolve o monitoramento do nível de expressão de um conjunto de genes em diferentes condições experimentais, como células tratadas com a droga A, outras tratadas com a droga B e assim sucessivamente, como na figura 2.4b.

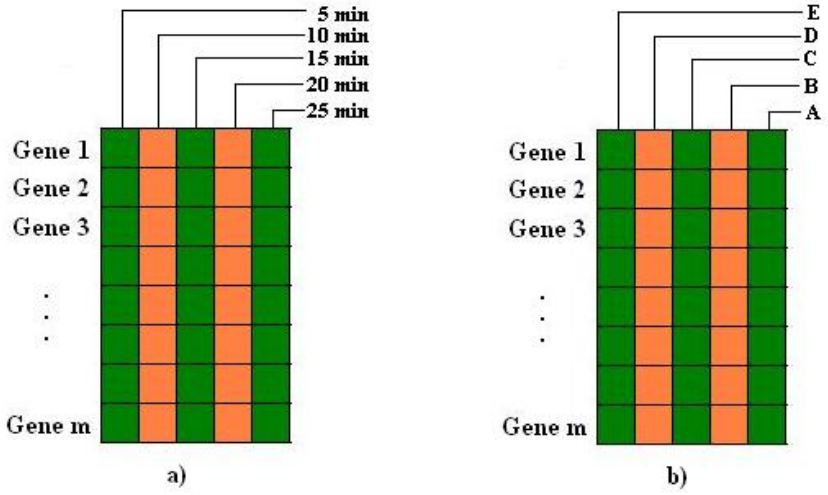

Figura 2.4: Diferentes configurações de experimentos relativos a expressão gênica.

Nesses dois casos a organização dos dados são matrizes de valores reais M_1 de dimesões $m \times l$ e M_2 de dimensões $m \times n$. Com a crescente disponibilidade de informações relativas a expressão gênica em repositórios públicos como o GEO (Gene Expression Omnibus - www.ncbi.nlm.nih.gov/geo) e o ArrayExpress (www.ebi.ac.uk/arrayexpress/), é possível criar conjuntos de dados contendo tanto informações relativas ao comportamento dos genes ao longo do tempo como também o nível de expressão desses genes em diferentes condições experimentais. Assim, de forma análoga à matriz de duas dimensões, podemos definir um novo conjunto de dados como uma matriz de valores reais M de dimensões $m \times n \times l$, onde cada célula m_{ijk} desta matriz representa o nível de expressão do gene i na condição experimental j no instante de tempo k. A figura 2.5 ilustra este novo conjunto de dados.

A utilidade de explorar esse novo conjunto de dados é fornecer a especialistas uma visão global do comportamento dos grupos de genes em certas condições experimentais ao longo do tempo. O resultado deste processo exploratório serve a dois propósitos:

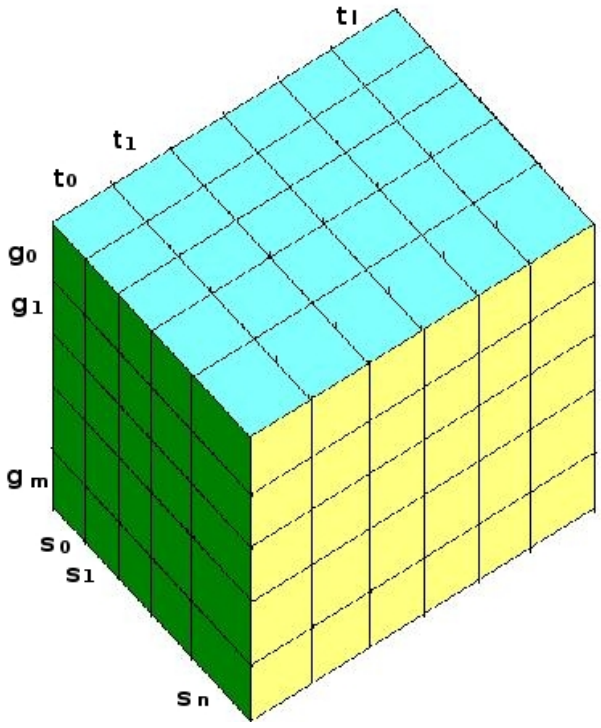

Figura 2.5: Conjunto de dados de expressão gênica em três dimensões.

- **Ajudar a entender fenômenos:** Para especialistas em biologia molecular, a existência de apenas um tipo de agrupamento (gene × tempo) ou (gene × condições), pode não ser suficiente para entender certos processos biológicos, nem o efeito de certos genes sobre os demais. Por isso, a disponibilidade de resultados complementares pode trazer componentes que elucidem os complexos mecanismos celulares, por exemplo, identificar genes co-regulados [6, 29].

- **Criar hipóteses:** Outra possibilidade para a exploração desse novo tipo de conjunto de dados é a criação de hipóteses para determinados eventos celulares. A existência de resultados computacionais coerentes, mas ainda não reportados na literatura, pode servir como ponto de partida para a execução de experimentos biológicos que expliquem fenômenos celulares. Neste caso, seria utilizar hipóteses computacionais para dar início a uma investigação biológica. Por exemplo, alguns estudos [24, 26] usam a tecnologia de microarrays para anotar a função de genes desconhecidos. Com essa nova abordagem, tal anotação estará baseada em um maior número de evidências e a função estabelecida para o gene pode ser mais confiável.

Um aspecto importante é a diferença fundamental entre explorar o conjunto de dados em três dimensões ao invés de unir as fatias de tempo lado a lado formando uma matriz M de dimensões $m \times nl$, como ilustrado na figura 2.6.

A diferença reside na coerência dos cluster criados. Se as fatias de tempo forem colocadas lado a lado, podemos criar clusters sem nenhuma relação entre as condições experimentais e os momentos do tempo. Já com os dados organizados na forma de "cubo", os genes dos clusters criados possuem um comportamento similar nas mesmas condições experimentais ao longo dos mesmos momentos do tempo.

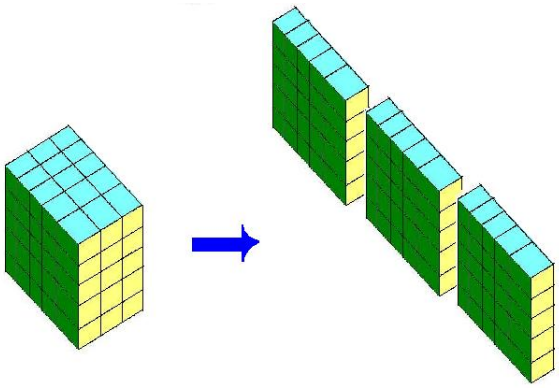

Figura 2.6: Processo de alinhar as diferentes fatias do tempo lado-a-lado.

Obter grupos significativos desta estrutura tridimensional foi o tema do estudo elaborado por Zhao e Zaki [30]. A seguir forneceremos uma explicação do funcionamento do algoritmo chamado TriCluster proposto por eles.

Sejam $G = \{g_0,\ g_1,\ \ldots,\ g_{n-1}\}$ um conjunto de n genes, $S = \{s_0,\ s_1,\ \ldots,\ s_{m-1}\}$ um conjunto de m tratamentos biológicos distintos e $T = \{t_0,\ t_1,\ \ldots,\ t_{l-1}\}$ um conjunto de l instantes de tempo. Assim, a estrutura formada é uma matriz de três dimensões $D = G \times S \times T$ de valores reais d_{ijk}. Um *tricluster* é uma sub-matriz C de D.

O algoritmo possui três passos principais e uma etapa opcional:

1. Para cada fatia de tempo, encontrar as células da matriz $M = G \times S$ cuja razão de seus valores não ultrapasse ϵ, que é um parâmetro de entrada. Esses objetos são utilizados para criar um grafo direcionado, como na figura 2.7, de forma que os vértices são as diferentes condições experimentais, e as arestas contém uma lista de genes cuja razão m_{ik}/m_{jk} está abaixo do parâmetro de entrada, de forma que $i, j \in [0,\ m-1]$ e $k \in [0,\ n-1]$. A complexidade de tempo desta etapa do algoritmo é da ordem de $O(|G||S|^2|T|)$.

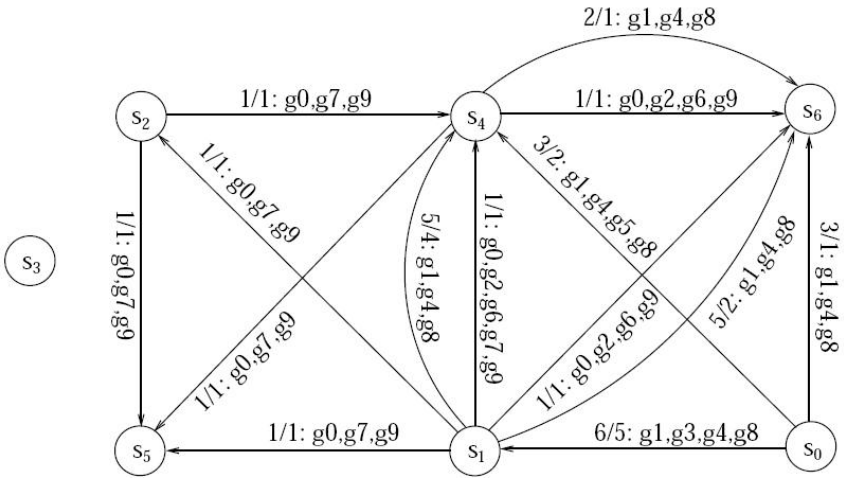

Figura 2.7: Grafo direcionado criado pelo algoritmo, onde os vértices são condições experimentais e as arestas são conjuntos de genes com comportamento similar nas condições experimentais adjacentes [30].

2. Para obter clusters em duas dimensões (biclusters), é feita uma busca em profundidade modificada no grafo, de forma que o algoritmo só continua a busca se o número

de genes presentes na lista da aresta que conecta dois vértices estiver acima de um parâmetro de entrada chamado min_g. Isso é feito individualmente para todas as camadas da terceira dimensão (fatias de tempo).

3. Depois de formados os biclusters em todas as camadas de tempo individualmente, os clusters de três dimensões (triclusters) são formados realizando a interseção entre os biclusters de todas as camadas de tempo. Se houver um número mínimo de condições experimentais min_s e um número mínimo de fatias de tempo min_t, um tricluster é criado, caso contrário é descartado.

   A segunda e terceira etapa do algoritmo possuem um mecanismo similar a enumerar os cliques maximais de um grafo e no pior caso podem ter complexidade exponencial. Por isso, estas duas etapas são as mais demoradas do algoritmo.

4. O quarto passo é opcional e consiste em unir ou apagar clusters que tenham uma sobreposição de genes, condições experimentais e fatias de tempo maiores que o valor de um parâmetro de entrada η. Esta etapa pode ser aplicada apenas a pares de clusters que se sobrepõem, assim, a complexidade da operação de determinar estes pares é da ordem de $O(|C|log(|C|)$, para todos os C clusters.

Este algoritmo pode encontrar cluster de diferentes formas, posicionados em qualquer lugar da matriz de dados e com sobreposição, o que é bastante apropriado para dados de expressão gênica.

Em um estágio avançado deste estudo tomamos conhecimento da existência de um outro algoritmo que encontra clusters em três dimensões [18]. Entretanto, o código fonte ou o arquivo binário foi solicitado diversas vezes mas não obtivemos resposta dos autores e devido a escassez de tempo, o algoritmo não foi implementado para testes.

2.5 Validação dos Resultados

Após a seleção e execução de um algoritmo de clustering é necessário validar os resultados obtidos. Segundo o estudo de Handl *et al.* [15], o processo de validação dos resultados de clustering em dados de expressão gênica freqüentemente é realizado através da inspeção visual em concordância com o conhecimento prévio da área, deixando de lado o uso de índices de qualidade.

Apesar da existência de muitos algoritmos de clustering, a maior parte não fornece informações a respeito da qualidade dos resultados obtidos e assim fica difícil saber quais são seus pontos fortes e fracos. Na dúvida os usuários continuam usando as mesmas alternativas de agrupamento, que nem sempre são as melhores mas estão disponíveis na maior parte dos pacotes de software. Assim, a validação dos resultados é necessária para quantificar a qualidade dos resultados obtidos tanto para um novo algoritmo quanto para um conjunto de dados recém-criado.

Existem várias técnicas de validação disponíveis na literatura, separadas em *validação externa* e *validação interna* [13]. A validação externa consiste em utilizar o algoritmo de clustering em conjuntos de dados cujas classes dos objetos são conhecidas. O uso dessas técnicas é útil para avaliar novos algoritmos de clustering no que diz respeito a sua capacidade de efetivamente revelar a real estrutura que os objetos formam no conjunto de dados. Quando as classes do conjunto de dados não forem conhecidas (novo conjunto de dados), as técnicas de validação interna devem ser utilizadas. Sem fazer uso de informação

adicional a respeito do conjunto de dados, essas técnicas avaliam a qualidade das estruturas encontradas levando em conta fatores como diferenças entre clusters e similaridade entre objetos do mesmo grupo.

Nas próximas seções, apresentamos as principais medidas de validação descritas por Handl *et al.* [15].

2.5.1 Medidas de validação externa

- *Medidas unárias*: Tradicionalmente as medidas de validação externas recebem como entrada o resultado de um determinado algoritmo de clustering e o comparam a um conjunto de dados cujas classes são conhecidas e cada objeto pertence a somente um cluster (*gold standard*). Assim, o resultado do processo de agrupamento pode ser comparado ao *gold standard* no que diz respeito à *pureza (purity)* e a *integridade (completeness)* dos clusters formados.

  – A **pureza** avalia a fração de objetos corretamente classificados levando em conta os clusters da saída do algoritmo e a classe mais apropriada do *gold standard*. Neste caso, definimos como mais apropriada a classe do *gold standard* que compartilha o maior número de objetos com o cluster formado. Esse critério de pureza pode possuir soluções triviais. O pior caso é a formação de clusters de apenas um elemento, que vai receber a pontuação máxima neste índice.

  – A medida de **integridade** consiste em avaliar qual fração dos objetos de uma determinada classe do *gold standard* foi classificada corretamente pelo algoritmo. Esta medida também pode possuir uma solução trivial, caso seja formado apenas um cluster contendo todos os elementos do conjunto de dados.

  Apesar de bastante semelhantes em uma primeira leitura, estas duas medidas são diferentes e possuem tendências opostas. Assim, para avaliar de forma satisfatória o resultado de um processo de clustering é necessário levar em conta tanto a medida de pureza quanto a medida de integridade, combinação feita por exemplo através da média harmônica (Medida-F).

- *Medidas binárias*: Estas medidas são utilizadas para avaliar a classificação correta de pares de objetos (com base no conjunto *gold standard*). Esta categoria de medidas difere das unárias pois apenas os pares de objetos recebem consideração positiva na comparação entre o resultado do algoritmo e o *gold standard*. É importante notar que considerar pares de objetos é mais restritivo que comparar apenas objetos individuais.

  Provavelmente a mais conhecida dessas medidasé o *Rand index* que determina a similaridade entre dois resultados de clustering baseado nas concordâncias e nas divergências do conteúdo de cada cluster. Existe também o Rand index ajustado, que introduz uma normalização em seu cálculo, de forma a deixar o valor do índice próximo a zero para classificações aleatórias. Outro índice foi proposto por Jaccard e leva em conta somente as concordâncias entre o resultado obtido e o *gold standard*.

2.5.2 Medidas de validação interna

- *Compactação*: Existem algumas medidas para avaliar a solidez ou homogeneidade dos clusters encontrados, por exemplo a soma do erro quadrático (otimizada pelo

algoritmo K-means) e a mínima variância intra-cluster. Exitem opções como a distância (média ou máxima) dos pares de objetos dentro dos clusters e similaridades máximas entre os elementos do mesmo grupo. Estas medidas asseguram que clusters de objetos similares recebam boa pontuação.

- *Conexão*: Estas medidas avaliam quanto o resultado é sensível a considerar densidades locais, agrupando os vizinhos mais próximos. Os principais representantes são: *k-neighborhood* [7] e *conectividade*.

- *Separação*: Estas avaliações são destinadas a quantificar o grau de separação entre os clusters descobertos. Estas distâncias podem ser por exemplo a média das distâncias entre os centróides dos clusters ou a menor distâncias entre os objetos dos clusters formados.

Apesar destes índices de validação externa e interna serem bastante conhecidos, eles inevitavelmente conduzem a perda de informação [15]. Assim, uma alternativa é o uso da fronteira de Pareto, que pode indicar vários resultados que combinam diversos índices de qualidade. Um resultado é considerado superior a outro se este for igual ou melhor em todos os índices utilizados e melhor em pelo menos um deles. Um estudo recente a este respeito pode ser encontrado em [14].

2.6 Interpretação dos Resultados

A interpretação dos resultados de clustering normalmente é feita com a colaboração do responsável pela execução dos experimentos e um profissional da área de biologia molecular. Este é um trabalho lento e delicado, e normalmente causa retornos a etapas preliminares da execução dos algoritmos. A primeira maneira de avaliar resultados de clustering para informações de microarray foi proposta por Eisen *et al.* [8] e consiste no uso de tabelas coloridas, onde as linhas representam os genes, as colunas representam as condições experimentais e a intensidade da cor nas células da tabela representam o nível de expressão do gene em questão, como na figura 2.3.

Entretanto, apesar de ser mais intuitiva que uma tabela numérica, esses gráficos ainda são bastante difíceis de avaliar, pois possuem um grande volume de informações agregadas.

Uma outra abordagem utilizada é o uso do Gene Ontology Project (/www.geneontology.org/). Este projeto traz informações a respeito da classificação de genes de acordo com sua função biológica. Desta maneira, ao obter um resultado de clustering é interessante verificar se os genes dos grupos formados realmente fazem parte de processos biológicos correlatos.

Capítulo

3

Algoritmo

Neste capítulo começaremos estabelecendo conceitos necessários para entender as etapas da solução que criamos neste estudo. Também apresentaremos uma descrição detalhada de cada etapa do algoritmo, incluindo seu mecanismo de operação e sua complexidade. No final do capítulo, criaremos um exemplo completo para elucidar e fixar o o funcionamento do nosso algoritmo.

3.1 Conceitos

Neste estudo desenvolvemos um algoritmo de clustering para encontrar grupos de genes em conjuntos de dados com três dimensões. Esse conjunto de dados pode ser descrito da seguinte forma. Considere três conjuntos: $G = \{g_1,\ g_2,\ \ldots,\ g_m\}$, $S = \{s_1,\ s_2,\ \ldots,\ s_n\}$ e $T = \{t_1,\ t_2,\ \ldots,\ t_l\}$ que representam, respectivamente, os genes, as condições experimentais e os instantes de tempo. Criamos com esses três conjuntos a entrada para o problema na forma $M = G \times S \times T$, onde cada célula m_{ijk} desta matriz representa o nível de expressão do gene i na condição experimental j no instante de tempo k.

Nosso algoritmo encontra clusters que combinam genes, condições experimentais e instantes do tempo. Estes clusters têm a forma $C = X \times Y \times Z$, onde $X \subseteq G$, $Y \subseteq S$ e $Z \subseteq T$ e $|X| \geq 2$, $|Y| \geq 2$ e $|Z| \geq 2$.

3.2 Algoritmo

As principais características do algoritmo são:

- **Realiza agrupamento por densidade:** Como demonstrado em estudos anteriores [9, 28], o uso de certos algoritmos de agrupamento como o K-means [20], combinado a certas medidas de distância favorece a descoberta de clusters com determinadas formas. Nesses estudos foi mencionado que o uso da distância Euclideana

favorece a descoberta de clusters hiper-esféricos, enquanto a distância Manhattan favorece a descoberta de clusters hiper-retangulares. Entretanto, quando o agrupamento é realizado através da densidade, isto é, quando os clusters são definidos como regiões contendo maior densidade de pontos que sua vizinhança, não existe favorecimento em relação a forma dos clusters. Esta característica é plenamente desejável, já que dados de expressão gênica não possuem forma.

- **Possui tolerância a ruído (*outliers*):** A presença de ruído em dados de microarray é amplamente conhecida e divulgada, representando uma forte crítica a confiabilidade desta técnica. Diferentes métodos estatísticos como o SAM (Significance Analysis of Microarray) [27] têm sido empregados na tentativa de separar ruído de informações relevantes. Entretanto, mesmo utilizando um método eficiente na fase de pré-processamento dos dados, ainda resta ruído nas informações. Nosso método é tolerante a ruído no que diz respeito a não incluir em nenhum cluster os objetos que possuem valores muito discrepantes em relação aos demais, ou que estão presentes em clusters de apenas um atributo devido a algum problema experimental.

- **Encontra clusters em subconjuntos de características:** Os algoritmos de clustering tradicionais buscam clusters em todas as dimensões (atributos) do conjunto de dados. O problema desta abordagem é que os algoritmos sofrem da chamada *maldição da dimensionalidade*. Este termo formulado por Richard Bellman [3], nos diz que conforme o número de dimensões de um conjunto aumenta, fica exponencialmente mais difícil otimizar alguma função neste espaço. Assim, se considerarmos um algoritmo de clustering como um problema de otimização cujo objetivo é encontrar os melhores grupos, a alta dimensionalidade de um conjunto de dados torna esta tarefa exponencialmente complexa. Além disso, conforme o número de dimensões cresce, torna-se menos provável que um grupo de genes tenha um comportamento similar ao longo de todas as dimensões. Neste estudo desenvolvemos um algoritmo capaz de considerar apenas subconjuntos dos atributos de objetos no momento de encontrar clusters. Assim, podem ser encontrados clusters que existem apenas em sub-espaços de características, sejam elas condições experimentais ou fatias de tempo.

- **Encontra clusters sobrepostos:** Para os profissionais da área de biologia molecular, é bastante conhecido o fato de um gene poder participar em diferentes processos celulares. Por isso nosso algoritmo é capaz de encontrar clusters com sobreposição, isto é, um mesmo objeto pode fazer parte de diferentes clusters. Entretanto, ao contrário de abordagens Fuzzy, o grau de pertinência de um objeto é o mesmo para todos os clusters dos quais ele faz parte.

Nosso algoritmo pode ser visto como uma extensão do algoritmo proposto por Agrawal *et al.* chamado CLIQUE [1]. As principais diferenças são a forma como as unidades densas são encontradas e a utilização de uma dimensão a mais no conjunto de entrada do nosso algoritmo. Enquanto o tamanho das unidades densas são fixas no CLIQUE, nossas unidades densas são dinâmicas, isto é, podem ser expandidas para cobrir uma região densa dos atributos (ver seção 3.2.1).

Para encontrar grupos de objetos com comportamento similar respeitando as exigências descritas acima, o algoritmo foi dividido em três etapas:

1. Encontrar unidades densas em cada atributo do conjunto de dados.

2. Unir as unidades densas em sub-espaços da mesma fatia de tempo.

3. Unir clusters de diferentes momentos do tempo.

Descreveremos cada etapa nas proximas seções, e no final do capítulo apresentaremos um exemplo completo para elucidar como as três etapas funcionam e integram-se para encontrar clusters no conjunto de dados.

3.2.1 Encontrar unidades densas

Vamos chamar de **atributo** os valores de expressão gênica em uma determinada condição experimental em um dado instante do tempo em ordem crescente. Ou seja, dado um j fixo no intervalo $1 \leq j \leq n$, e um k fixo no intervalo $1 \leq k \leq l$, o atributo A_{jk} representa um vetor de valores relativos ao nível de expressão de todos os genes na condição experimental j, no momento k. A representação visual de um atributo é mostrada na figura 3.1.

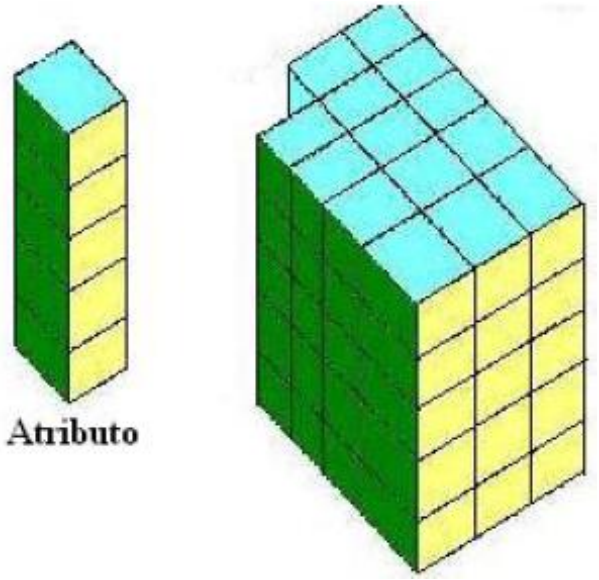

Figura 3.1: Representação de um atributo retirado do conjunto de dados.

O maior e o menor valores do atributo A_{jk} são denotados respectivamente por h_{jk} e l_{jk}. A **densidade de um atributo** é

$$d(A_{jk}) = \frac{m}{(h_{jk} - l_{jk})} \tag{3.1}$$

onde m é o número de genes do conjunto de dados. O raciocínio envolvido neste cálculo é a busca pelo valor da razão entre a quantidade de elementos existentes e o tamanho do intervalo que eles ocupam. Este é um princípio semelhante ao cálculo da densidade de substâncias, obtida pela divisão do peso pelo volume, por exemplo kg/m^3.

Vamos chamar de **unidade** um intervalo $U \subseteq A_{jk}$. Representamos respectivamente seu maior e menor valores por h_U e l_U. De forma análoga à equação 3.1, a densidade de uma unidade U é dada por

$$d(U) = \frac{|U|}{(h_U - l_U)}. \tag{3.2}$$

Uma unidade U é considerada **densa** se $d(U) \geq \delta d(A_{jk})$, onde $\delta \in \Re$ e $1 < \delta \leq \infty$.

Nesta primeira etapa do algoritmo, buscamos unidades densas em cada atributo A_{jk}. O motivo desta tarefa é encontrar grupos de genes com nível de expressão similar no mesmo atributo. Em outras palavras, são formados **clusters de apenas uma dimensão**. Ilustramos uma unidade densa em um atributo na figura 3.2a.

Para encontrar unidades densas em cada atributo usamos o seguinte procedimento que tem como parâmetros δ e min_g, onde min_g é a quantidade inicial de elementos de U e $min_g \geq 2$. Vamos usar um vetor ordenado de tamanho m onde cada posição possui o valor de expressão de um gene do atributo A_{jk}. Criamos uma unidade U_1 da primeira posição do vetor até min_g e verificamos se essa unidade é densa. Em caso afirmativo, aumentamos o tamanho dessa unidade densa em uma posição e realizamos a verificação novamente. Agora, essa unidade ocupa um intervalo da primeira posição do vetor até a posição $(min_g + 1)$, como mostrado na figura 3.2b. Estes dois passos:

1. Aumentar o tamanho da unidade;
2. Verificar se a nova unidade é densa;

são repetidos até que a unidade deixe de ser densa ou o vetor termine.

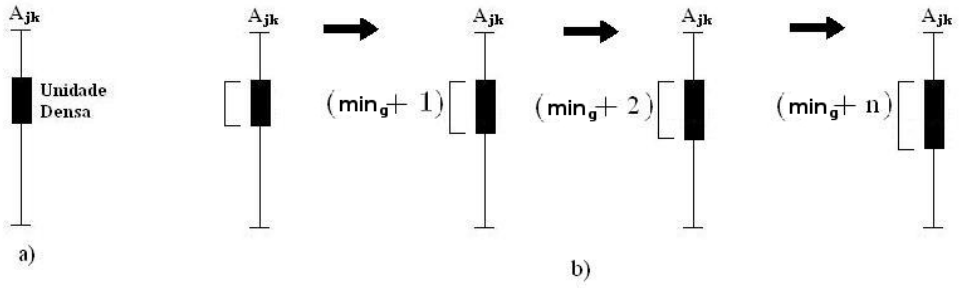

Figura 3.2: a) Unidade densa. b)Expansão de uma unidade densa.

Se a primeira unidade criada não for densa, prosseguimos criando uma nova unidade U_1 da posição 2 do vetor até a posição (min_g+1) e verificamos se esta unidade é densa. Em caso afirmativo, aumentamos seu tamanho pelo processo descrito acima; caso contrário, criamos uma nova unidade densa U_1 da posição 3 até a posição $(min_g + 2)$.

Uma situação possível é chegar próximo ao fim do atributo A_{jk} e restar um número de objetos inferior a min_g, insuficientes para criar uma unidade. Não poderiamos deixar para trás esses valores e perder informações. Por isso, levamos em conta a unidade densa mais próxima ao final do intervalo, denotada U_x, e verificamos se uma unidade maior, ocupando o intervalo da primeira posição de U_x até m é densa. Em caso afirmativo, a unidade densa U_x passa a ocupar este novo intervalo, caso contrário, decrementamos uma posição de m e verificamos se a nova unidade da primeira posição de U_x até a posição $m - 1$ é densa. Este procedimento é feito até a última posição à frente de U_x, como podemos ver na figura 3.3a.

Depois de encontrar todas as unidades densas em um atributo, verificamos para cada unidade densa se a união dela com a unidade densa adjacente constitui uma unidade densa maior.

A **união de duas unidades** densas adjacentes U_1 e U_2 tais que $h_{U_2} > l_{U_1}$ e separadas por um intervalo I formam uma unidade densa maior se

$$\frac{|U_1| + |U_2| + |I|}{h_{U_2} - l_{U_1}} \geq \delta d(A_{jk}). \tag{3.3}$$

Desta maneira, podemos perceber que duas unidades densas adjacentes podem ser tão densas que este "excesso" supera a falta de densidade no intervalo I que as separa, como mostra a figura 3.3b.

O algoritmo 1 traz formalmente as etapas de criação, expansão e testes da união de unidades densas para um atributo do conjunto de dados. Este mesmo procedimento deve ser realizado para todos os outros atributos do conjunto.

A entrada para o algoritmo 1 é um vetor contendo os valores de um atributo A_{jk} ordenado. Esses valores são extraídos da matriz de 3 dimensões. Além do vetor é necessário manter índices relacionado as posições do atributo aos índices dos genes, que não devem ser perdidas na ordenação. Ao término desse algoritmo, o vetor com os valores de A_{jk} e os índices dos genes serão usados para construir a entrada para o algoritmo 2. Esses detalhes são simples e foram omitidos.

No algoritmo, A_{jk} é um vetor que armazena os valores de um atributo da condição experimental j, fatia de tempo k. U é um registro que representa uma unidade e tem quatro campos: l que armazena o menor valor do atributo na unidade, h que armazena o maior valor do atributo na unidade, f que armazena a primeira posição da unidade em relação ao atributo e m que armazena o tamanho da unidade. Os campos h e l não são necessários mas foram incluídos para tornar o algoritmo mais claro. U_{jk} é um vetor de registros de unidades.

Após a execução dessa etapa, temos clusters unidimensionais representados por unidades densas. Os conjuntos dessas unidades possuem a forma $U_{jk} = \{u_1,\ u_2,\ \ldots,\ u_j\}$, $1 \leq j \leq m/min_g$, onde $u_i = (g_1,\ g_2,\ \ldots,\ g_p)$, para $min_g \leq p \leq m$ e $u_i \cap u_j = \emptyset$ para qualquer i e j; ou seja, não existe sobreposição entre unidades densas do mesmo atributo.

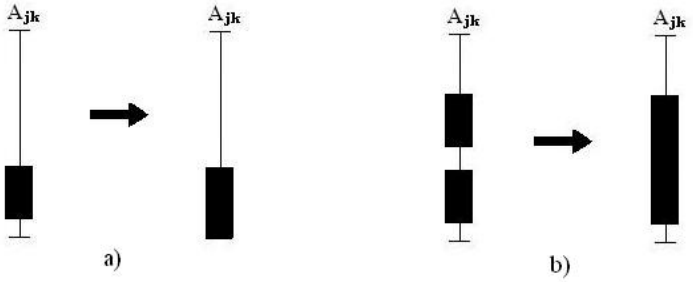

Figura 3.3: a) Processo de expansão de uma unidade densa levando em conta os objetos finais do atributo. b) União de duas unidades densas adjacentes para formar uma nova unidade densa.

Complexidade

A complexidade do algoritmo é $O(m)$ para um atributo. É fácil perceber que para encontrar as unidades densas cada elemento de A_{jk} participa no máximo de um número constante de cálculos de densidade. Considere por exemplo uma posição x de A_{jk}. Só existem duas possibilidades para a inclusão de x em alguma unidade. Ou x é o primeiro elemento de uma unidade, que pode densa ou não, ou x é acrescentado a alguma unidade que é densa. No primeiro caso, x pode ser usado no máximo m vezes no cálculo de densidade. No segundo, x será usado uma única vez no cálculo de densidade. O número

Entrada: Vetor A_{jk} ordenado com os valores de um atributo, δ: Fator de multiplicação da densidade do atributo, min_g: Tamanho mínimo de uma unidade densa.
Saída: U_{jk}: Conjuntos de unidades densas das condições experimentais j, fatias de tempo k.

$d_{A_{jk}} \longleftarrow m/(A_{jk}[m-1] - A_{jk}[0])$ *# Densidade do atributo*
$v \longleftarrow 0$ *# Número de unidades densas em* U_{jk}
$i \longleftarrow 0$
enquanto $i \leq m - min_g$ **faça**
  $U.f \longleftarrow i$
  $U.m \longleftarrow i + min_g - 1$
  $U.l \longleftarrow A_{jk}[i]$
  $U.h \longleftarrow A_{jk}[U.f + U.m - 1]$
  $d_U \longleftarrow min_g/(U.h - U.l)$
  se $d_U \geq \delta d_{A_{jk}}$ **então**
    # Expandir a unidade densa
    enquanto $d_U \geq \delta d_{A_{jk}}$ **e** $U.f + U.m - 1 < m$ **faça**
      $U.m \longleftarrow U.m + 1$
      $U.h \longleftarrow A_{jk}[U.f + U.m - 1]$
      $d_U \longleftarrow U.m/(U.h - U.l)$
    fim enquanto
    $U.m \longleftarrow U.m - 1$
    $U.h \longleftarrow A_{jk}[U.f + U.m - 1]$
    # Se houver mais que uma unidade densa, verificar se a união é densa
    se $v > 0$ **e** $(U.f + 1 + U.m - (U_{jk}[v-1].f)/(U.h - U_{jk}[v-1].l)$ **então**
      $U_{jk}[v-1].m \longleftarrow U.f + U.m - U_{jk}[v-1].f + 1$
      $U_{jk}[v-1].h \longleftarrow U.h$
    fim se
    senão
      $U_{jk}[v] \longleftarrow U$
      $v \longleftarrow v + 1$
    fim se
    $i \longleftarrow i + U.m + 1$
  fim se
  senão
    $i \longleftarrow i + 1$
  fim se
fim enquanto
 # Tentar expandir a última unidade.
se $v > 0$ **então**
  $U_{jk}[v-1].h \longleftarrow A_{jk}[m-1]$
  $U_{jk}[v-1].m \longleftarrow m - U_{jk}[v-1].f$
  enquanto $U_{jk}[v-1].m/(U_{jk}[v-1].h - U_{jk}[v-1].l) < d_{A_{jk}}$ **faça**
    $U_{jk}[v-1].m \longleftarrow U_{jk}[v-1].m - 1$
    $U_{jk}[v-1].h \longleftarrow A_{jk}[U_{jk}[v-1].m]$
  fim enquanto
fim se

Algoritmo 1: Algoritmo para encontrar unidades densas em atributos.

total é uma soma dessas duas situações. O algoritmo deve ser executado nl vezes e cada atributo deve ser ordenado, o que dá um custo total $O(nlm \log m)$.

3.2.2 Unir as unidades densas em sub-espaços do mesmo instante do tempo

Esta segunda etapa do algoritmo consiste em combinar as unidades densas de diferentes condições experimentais em cada fatia de tempo, para formar clusters de duas dimensões. Assim, teremos clusters com a forma $C = X \times Y$, onde $X \subseteq G$, $Y \subseteq S$.

O primeiro passo desta etapa é numerar as unidades densas de cada atributo de forma que cada uma tenha identificação própria e única. Em cada atributo a numeração vai de 1 em diante e os números são únicos apenas dentro de um atributo. Por exemplo, no atributo A pode existir apenas uma unidade densa u_4, mas em outro atributo B pode haver uma unidade denotada u_4.

A seguir, para cada fatia de tempo do conjunto de dados, construímos uma tabela T de tamanho $m \times n$ conforme a figura 3.4. Nessa tabela o valor de uma determinada célula é o número de identificação da unidade densa da qual o gene faz parte naquele atributo. Se o gene não fizer parte de nenhuma unidade densa então a célula fica com valor 0.

	A1k	A2k	A3k	A4k
g_1	1	0	2	5
g_2	2	3	3	1
g_3	4	0	2	1
g_4	1	1	3	0
g_5	1	0	0	2
g_6	4	0	1	1

Figura 3.4: Matriz contendo m linhas por n colunas na fatia de tempo t, onde cada célula traz o número da unidade densa à qual o gene pertence no atributo em questão.

Após criarmos a tabela, comparamos sua primeira linha, coluna a coluna, à todas as outras linhas. Aqui entra outro parâmetro do algoritmo, min_s, que estabelece o número mínimo de condições experimentais em que um cluster bidimensional deve existir. Se pelo menos min_s colunas são iguais então dois genes fazem parte das mesmas unidades densas em atributos diferentes do conjunto de dados. Ou seja, possuem um comportamento similar naquelas condições experimentais na mesma fatia de tempo. O mesmo procedimento é repetido para todas as linhas (genes) da tabela.

Quando duas linhas possuem o mínimo de colunas em comum, antes de criar um novo cluster verificamos se dentre os clusters existentes algum possui elementos nas mesmas condições experimentais. Em caso afirmativo o gene é adicionado ao cluster existente, caso contrário um cluster com dois genes é criado.

Nessa etapa pode ocorrer a criação de clusters sobrepostos, pois o mesmo gene pode ser membro de um cluster em algumas condições experimentais e membro de outro cluster em outras condições experimentais.

Esta etapa aparece formalizada no algoritmo 2. A entrada é um vetor U de vetores de registros representando unidades densas. Intituitivamente, a entrada contém genes em unidades densas em todos os atributos de uma fatia de tempo. Os campos do registro são um vetor chamado *genes*, que guarda índices dos genes na unidade densa, e um inteiro m

que é o tamanho do vetor *genes*. O vetor U pode ser facilmente construído a partir da saída do Algoritmo 1.

É importante descrever as seguintes estruturas utilizadas neste algoritmo:

- T é uma matriz de dimensões $m \times n$. Cada linha i da matriz é um vetor de condições experimentais com n posições e será denotada por $T[i]$. Cada posição de um vetor $T[i]$ contém o número de uma unidade densa a que o gene i pertence.
- C, C_{temp} são registros com dois vetores: genes e colunas. Esses vetores representam os clusters de duas dimensões. Vale lembrar que um cluster neste ponto é uma região de uma única fatia do tempo, isto é, tem duas dimensões. Assim, $C.genes$ armazena os índices dos genes do cluster C e $C.colunas$ armazena os índices de suas condições experimentais. B é um vetor de registros do mesmo tipo de C.
- A operação $T[i] \ominus T[j]$ é utilizada para representar a comparação, coluna a coluna, entre as linhas i e j e resulta em um vetor com as colunas comuns a essas duas linhas.

Complexidade

A inicialização de T e seu preenchimento custam $O(mn)$, pois existem no máximo $\frac{m}{2}$ unidades densas em um atributo. Para comparar cada cluster recém-criado com os clusters em B temos custo $O(n)$. Então o custo para comparar as linhas de T e construir B pode ser dado por

$$\sum_{i=1}^{m-1}\sum_{j=1}^{n-1}(O(n) + \sum_{k=1}^{ij/2} O(n)) \leq \sum_{i=0}^{m}\sum_{j=0}^{m}\sum_{k=0}^{m^2} = O(m^4 n)$$

O custo total é $O(m^4 n + m^2 n^2)$. Para l fatias de tempo temos $O(lm^4 n + lm^2 n^2)$.

3.2.3 Unir clusters de diferentes momentos do tempo

Na primeira etapa do algoritmo desenvolvido neste estudo encontramos *clusters unidimensionais* de genes sem sobreposição. Na segunda etapa do algoritmo encontramos *clusters bidimensionais* com sobreposição existentes em um subconjunto dos genes e das condições experimentais. Nesta terceira e última etapa descrevemos o procedimento para encontrar *clusters tridimensionais* com sobreposição existentes em subconjuntos de genes, condições experimentais e fatias do tempo.

Esta terceira etapa é bastante simples e consiste em selecionar cada cluster C_{ik} e obter a interseção com outros clusters C_{jp}, onde $1 \leq i, j \leq mn/2$ e $p > k$. Ou seja, selecionamos cada cluster de cada fatia de tempo e buscamos uní-lo à clusters de fatias de tempo posteriores. Não faz sentido realizar a interseção entre clusters da mesma fatia de tempo, pois os clusters maximais das fatias individuais já foram encontrados anteriormente (ver seção 3.2.2).

De forma semelhante à segunda parte do algoritmo, antes de criarmos um cluster tridimensional verificamos se existe um cluster C_{jp} tal que $C_{ik}.genes \cap C_{jp}.genes = C_{ik}.genes$ e $C_{ik}.colunas \cap C_{jp}.colunas = C_{ik}.colunas$. Quer dizer, verificamos se o cluster que está sendo criado já foi definido em outras fatias de tempo. Como nesta terceira etapa estamos

Entrada: U: Vetor de vetores de registros de unidades densas.
Saída: B: Clusters de duas dimensões para uma fatia de tempo.

 # Clusters temporários.
$C \longleftarrow \emptyset$
$C_{temp} \longleftarrow \emptyset$
$similaridade \longleftarrow 0$ *# Variável para armazenar o numero de colunas em comum.*

 # Inicializar e preencher a matriz T
$T \longleftarrow 0$
para $s \longleftarrow 1$ *até* n **faça**
  para $u \longleftarrow 0$ *até* $|U_s| - 1$ **faça**
    $i \longleftarrow 0$
    enquanto $i < U_s[u].m$ **faça**
      $T[U_s[u].genes[i]][s] \longleftarrow u + 1$
      $i \longleftarrow i + 1$
    fim enquanto
  fim para
fim para
 # Comparar linha a linha a matriz T
para $i \longleftarrow 0$ *até* $m - 1$ **faça**
  para $j \longleftarrow i + 1$ *até* $m - 1$ **faça**
    $similaridade \longleftarrow |T[i] \ominus T[j]|$
    se $similaridade \geq min_s$ **então**
      # Armazena as condições experimentais em C.
      $C.colunas \longleftarrow T[i] \ominus T[j]$
      # Armazena os genes em C.
      Acrescentar i à primeira posição livre do vetor $C.genes$
      Acrescentar j à primeira posição livre do vetor $C.genes$
      se $B \neq \emptyset$ **então**
        para todo $C_{temp} \in B$ **faça**
          # Se as condições experimentais forem as mesmas, faz a união.
          se $|C_{temp}.colunas \ominus C.colunas| = |C.colunas|$ **então**
            $C_{temp}.genes \longleftarrow C_{temp}.genes \cup C.genes$
          fim se
          senão
            Acrescentar C à primeira posição livre de B
          fim se
        fim para todo
      fim se
      senão
        Acrescentar C à primeira posição livre de B
      fim se
    fim se
  fim para
fim para

Algoritmo 2: Algoritmo para unir unidades densas em sub-conjuntos de atributos. É importante ressaltar que as operações de interseção são feitas em tempo linear.

lidando com clusters da forma $C = X \times Y \times Z$, onde $X \subseteq G$, $Y \subseteq S$ e $Z \subseteq T$, se o resultado da verificação de pré-existência for afirmativo, o conjunto Z do cluster existente recebe a adição de mais um elemento, correspondente a fatia de tempo k; caso contrário, um novo cluster tridimensional é criado.

Depois de processar todas as fatias de tempo e criar os clusters de três dimensões, realizamos uma filtragem para eliminar clusters existentes em poucas fatias de tempo. Para isso, introduzimos um parâmetro chamado min_t, que determina a quantidade mínima de fatias do tempo que um cluster deve possuir para ser considerado válido.

A formalização desta terceira parte pode ser vista no algoritmo 3. A entrada é um vetor de clusters em duas dimensões do tipo produzido pelo algoritmo 2. A saída é um vetor de registros de clusters.

Complexidade

De forma similar a segunda etapa do algoritmo, antes de criar um novo cluster verificamos se ainda não existe um cluster com os mesmos genes e condições experimentais que o cluster candidato. O processo completo de verificação para todas as codições experimentais e todas as fatias de tempo possui custo

$$\begin{aligned}
\sum_{i=1}^{l}\sum_{j=1}^{l}\sum_{a=1}^{mn/2}\sum_{b=1}^{mn/2}\left(O(m+n)+\sum_{c=1}^{\frac{l^2m^2n^2}{4}}(m+n)\right) &\leq \sum_{i=1}^{l}\sum_{j=1}^{l}\sum_{a=1}^{mn}\sum_{b=1}^{mn}\sum_{c=1}^{l^2m^2n^2}(m+h) \\
&\leq (((((m+n)l^2m^2n^2)mn)mn)l)l \\
&\leq l^4m^4n^4(m+n) \\
&= O(l^4m^5n^4 + l^4m^4n^5)
\end{aligned}$$

3.2.4 *Exemplo*

Para ilustrar o funcionamento e a integração de todas as partes do algoritmo proposto, utilizamos um exemplo envolvendo um conjunto de dados elaborado para este propósito. As tabelas 3.1 mostram um conjunto de dados com o nível de expressão de 10 genes, 6 condições experimentais e 4 fatias do tempo.

Os parâmetros do algoritmo recebem os seguintes valores: $\delta = 133.0$, $min_g = 4$, $min_s = 3$ e $min_t = 2$. Dessa maneira, buscamos clusters que tenham densidade no mínimo 133 vezes maior que a densidade do atributo, no mínimo 4 genes, 3 condições experimentais e 2 fatias de tempo.

Começando pela fatia de tempo t_0 e utilizando o algoritmo 1 vamos encontrar *unidades densas* em cada atributo. Podemos ver na tabela 3.2 as densidades de cada atributo da fatia t_0 e na tabela 3.3 todas as unidades densas para estes atributos.

Após a descoberta das unidades densas em cada atributo da fatia t_0, o algoritmo 2 é responsável por unir essas unidades densas de uma dimensão para formar *clusters bidimensionais* com sobreposição. Nesse caso, ainda considerando apenas a fatia de tempo t_0 obtemos os clusters bidimensionais da tabela 3.4.

Essas duas etapas do algoritmo (encontrar clusters uni e bidimensionais) devem ser feitas para todas as fatias de tempo do conjunto de dados. Vamos considerar que estas duas etapas também foram realizadas nas fatias de tempo t_1, t_2 e t_3 e o resultado são os clusters bidimensionais mostrados na tabela 3.5.

Entrada: B: Vetor de clusters bidimensionais.
Saída: C: Conjunto de clusters de três dimensões.

 # Clusters temporários.
$C_a \longleftarrow \emptyset$
$C_b \longleftarrow \emptyset$
$C_z \longleftarrow \emptyset$
$C_{temp} \longleftarrow \emptyset$
$C \longleftarrow \emptyset$ *# Conjunto de clusters de três dimensões.*

para $i \longleftarrow 0$ *até* $l - 1$ **faça**
  para $j \longleftarrow i$ *até* $l - 1$ **faça**
    $C_{temp}.genes \longleftarrow C[i].genes \cap C[j].genes$
    $C_{temp}.colunas \longleftarrow C[i].colunas \cap C[j].colunas$
    Acrescentar i à primeira posição livre do vetor $C_{temp}.tempo$
    Acrescentar j à primeira posição livre do vetor $C_{temp}.tempo$
    # Testa a validade de duas dimensões do cluster criado.
    se $|C_{temp}.genes| \geq min_g$ **e** $|C_{temp}.colunas| \geq min_s$ **então**
      se $C \neq \emptyset$ **então**
        para todo $C_z \in C$ **faça**
          se $|C_{temp}.genes \cap C_z.genes| = |C_z.genes|$ **e**
          $C_{temp}.colunas \cap_z .colunas| = |C_z.colunas|$ **então**
            Acrescentar j à primeira posição do vetor $C_z.tempo$
          fim se
        fim para todo
        senão
          Acrescentar C_{temp} à primeira posição livre de C
        fim se
      fim se
      senão
        Acrescentar C_{temp} à primeira posição livre de C
      fim se
    fim se
  fim para
fim para
para todo $C_z \in C$ **faça**
  se $|C_z| < min_t$ **então**
    Remover C_z *de* C
  fim se
fim para todo

Algoritmo 3: Algoritmo para unir clusters de diferentes fatias do tempo.

t_0	s_1	s_2	s_3	s_4	s_5	s_6
g_1	-0.21	-0.72	1.33	1.22	-0.23	1.2
g_2	-0.23	0.6	0.23	1.21	-0.22	-0.99
g_3	-0.19	0.44	0.89	1.23	-0.21	-0.19
g_4	-0.2	0.89	-0.55	1.21	-0.22	0.94
g_5	-0.22	1.21	-0.54	-0.99	-0.21	0.93
g_6	1.2	1.22	-0.55	0.99	1.2	0.94
g_7	0.74	1.2	-0.56	-0.65	0.88	0.95
g_8	0.76	1.19	-0.55	-0.66	0.87	0.93
g_9	0.73	1.22	1.2	-0.64	0.88	0.95
g_{10}	0.75	1.21	0.55	-0.65	0.89	1.23

t_1	s_1	s_2	s_3	s_4	s_5	s_6
g_1	1.12	-0.61	2.66	2.55	1.1	2.53
g_2	1.1	1.93	1.56	2.54	1.11	-0.34
g_3	1.14	1.77	2.22	2.56	1.12	1.14
g_4	1.13	2.22	0.78	2.54	1.11	2.27
g_5	1.11	2.54	0.79	-0.34	1.12	2.26
g_6	-2.53	2.55	0.78	2.32	-2.53	2.27
g_7	2.07	2.53	0.77	0.68	2.21	2.28
g_8	2.09	2.52	0.78	0.67	2.2	2.26
g_9	2.06	2.55	2.53	0.69	2.21	2.28
g_{10}	2.08	2.54	-1.88	0.68	2.22	2.56

t_2	s_1	s_2	s_3	s_4	s_5	s_6
g_1	1.33	-0.18	-0.72	-1.17	1.2	1.2
g_2	0.23	1.14	0.6	0.15	-0.99	-0.99
g_3	0.89	0.98	0.44	-0.01	-0.19	-0.19
g_4	-0.55	1.43	0.89	0.44	0.94	0.94
g_5	-0.54	1.75	1.21	0.76	-0.99	-0.99
g_6	-0.55	1.76	1.22	0.77	0.99	0.99
g_7	-0.56	1.74	1.2	0.75	-0.56	-0.56
g_8	0.72	1.73	1.19	0.74	-0.55	-0.55
g_9	1.2	1.76	1.22	0.77	1.2	1.2
g_{10}	0.55	1.75	1.21	0.76	0.55	0.55

t_3	s_1	s_2	s_3	s_4	s_5	s_6
g_1	3.68	-2.17	-1.63	-1.18	3.55	3.55
g_2	2.58	3.49	2.95	2.5	-1.36	1.36
g_3	3.24	3.33	2.79	2.34	2.16	2.16
g_4	1.8	3.78	3.24	2.79	3.29	3.29
g_5	1.81	4.1	3.56	3.11	1.36	-1.36
g_6	1.8	4.11	3.57	3.12	3.34	3.34
g_7	1.79	4.09	3.55	3.1	1.79	1.79
g_8	3.07	4.08	3.54	3.09	1.8	1.8
g_9	-3.55	4.11	3.57	3.12	3.55	3.55
g_{10}	2.9	4.1	3.56	3.11	2.9	2.9

Tabela 3.1: Valores de expressão gênica relativos as diferentes fatias de tempo.

Atributos de t_0	**Densidade**
s_1	6.99
s_2	5.15
s_3	5.29
s_4	4.5
s_5	6.99
s_6	4.5

Tabela 3.2: Valores de densidade dos atributos da fatia t_0.

Unidade Densa	**Genes**	**Atributo**	**Densidade**
u_1	$\{g_1, g_2, g_3, g_4, g_5\}$	s_1	166.67
u_2	$\{g_7, g_8, g_9, g_{10}\}$	s_1	400.0
u_3	$\{g_7, g_8, g_9, g_{10}\}$	s_2	133.33
u_4	$\{g_4, g_5, g_6, g_7, g_8, g_{10}\}$	s_3	200.0
u_5	$\{g_7, g_8, g_9, g_{10}\}$	s_4	250.0
u_5	$\{g_1, g_2, g_3, g_4\}$	s_4	200.0
u_6	$\{g_1, g_2, g_3, g_4\}$	s_5	200.0
u_7	$\{g_4, g_5, g_6, g_7, g_8, g_9, g_{10}\}$	s_6	200.0

Tabela 3.3: Unidades densas em cada atributo da fatia t_0 e seus valores de densidade.

Cluster	**Genes**	**Atributos**
C_1^{2d}	$\{g_1, g_2, g_3, g_4\}$	$\{s_1, s_4, s_5\}$
C_2^{2d}	$\{g_7, g_8, g_9, g_{10}\}$	$\{s_1, s_2, s_4, s_5\}$
C_3^{2d}	$\{g_5, g_6, g_7, g_8\}$	$\{s_2, s_3, s_6\}$

Tabela 3.4: Clusters bidimensionais encontrados na fatia de tempo t_0.

Fatia	**Cluster**	**Genes**	**Atributos**
t_0	C_1	$\{g_1, g_2, g_3, g_4\}$	$\{s_1, s_4, s_5\}$
	C_2	$\{g_7, g_8, g_9, g_{10}\}$	$\{s_1, s_2, s_4, s_5\}$
	C_3	$\{g_5, g_6, g_7, g_8\}$	$\{s_2, s_3, s_6\}$
t_1	C_1	$\{g_1, g_2, g_3, g_4\}$	$\{s_1, s_4, s_5\}$
	C_2	$\{g_7, g_8, g_9, g_{10}\}$	$\{s_1, s_2, s_4, s_5, s_6\}$
	C_3	$\{g_5, g_6, g_7, g_8\}$	$\{s_2, s_3, s_6\}$
t_2	C_1	$\{g_5, g_6, g_7, g_8, g_9, g_{10}\}$	$\{s_2, s_3, s_4\}$
t_3	C_1	$\{g_5, g_6, g_7, g_8, g_9, g_{10}\}$	$\{s_2, s_3, s_4\}$

Tabela 3.5: Clusters bidimensionais encontrados nas demais fatias de tempo.

Cluster	Genes	Atributos	Fatias
C_1	$\{g_1, g_2, g_3, g_4\}$	$\{s_1, s_4, s_5\}$	$\{t_0, t_1\}$
C_2	$\{g_7, g_8, g_9, g_{10}\}$	$\{s_1, s_2, s_4, s_5\}$	$\{t_0, t_1\}$
C_3	$\{g_5, g_6, g_7, g_8\}$	$\{s_2, s_3, s_6\}$	$\{t_0, t_1\}$
C_4	$\{g_5, g_6, g_7, g_8, g_9, g_{10}\}$	$\{s_2, s_3, s_4\}$	$\{t_2, t_3\}$

Tabela 3.6: Clusters tridimensionais encontrados no conjunto de dados do exemplo.

O último passo realizado pelo algoritmo 3 consiste em combinar os clusters de diferentes fatias do tempo para encontrar clusters existentes em três dimensões com genes que tenham comportamento similar tanto nas condições experimentais quanto ao longo do tempo. Devemos encontrar a interseção entre os cluster de uma fatia do tempo e os clusters das demais. Após este procedimento, encontramos os clusters tridimensionais presentes na tabela 3.6. Isso conclui a execução do algoritmo.

Capítulo

4

Experimentos

Neste capítulo descrevemos a implementação do nosso algoritmo, os experimentos comparativos realizados para demonstrar as vantagens e desvantagens da nossa solução em relação ao algoritmo TriCLuster de Zhao e Zaki [30]. Descrevemos também os mecanismos de validação utilizados para atestar a qualidade dos resultados obtidos e o ambiente experimental utilizado para realizar este estudo.

4.1 Implementação

A estratégia que escolhemos para realizar a implementação e testes da solução proposta neste estudo foi dividir a tarefa em três partes:

- Algoritmo de clustering;
- Algoritmos de validação;
- Programas para executar os experimentos.

Implementamos o algoritmo principal e os mecanismos de validação na linguagem C++, enquanto os programas responsáveis por realizar os experimentos foram implementados em Bash script.

4.1.1 Algoritmo principal

Escolhemos a abordagem orientada a objetos para implementar o algoritmo, por isso as classes foram cuidadosamente projetadas antes de iniciar a implementação. Elaboramos no software Umbrello 1.5.5 (http://uml.sourceforge.net/) o diagrama de classes relativo a modelagem UML (Unified Modelling Language) da figura 4.1.

Podemos ver na figura 4.1 a existência de 5 classes, sendo que 4 delas são as classes principais e uma delas convencionamos nomear *classe de apoio*. A seguir descrevemos cada uma delas e seus relacionamentos.

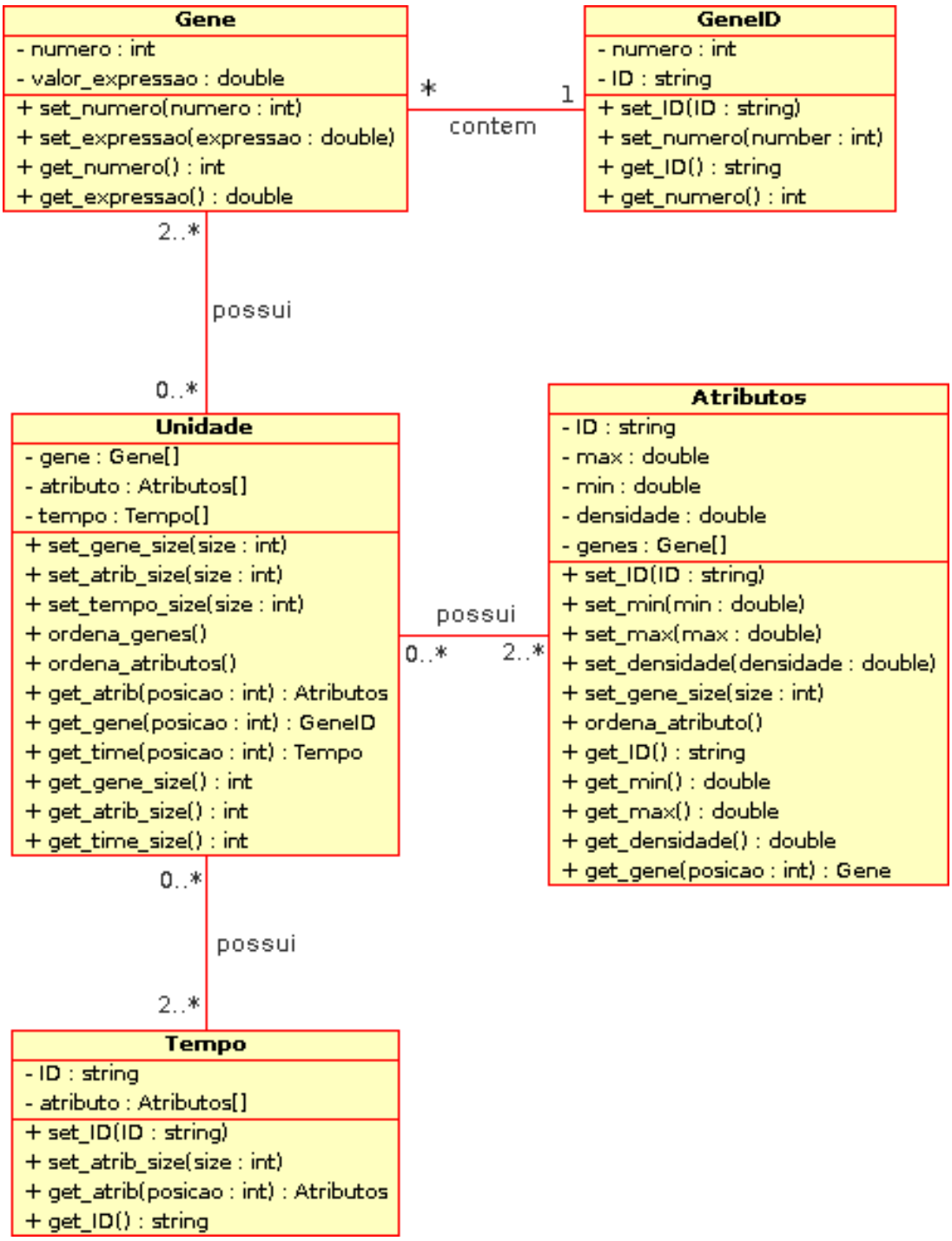

Figura 4.1: Diagrama de Classes do algoritmo desenvolvido.

- **Gene:** Esta classe define objetos que representam genes do conjunto de dados. Cada objeto contém um número de identificação e um valor real relativo a seu nível de expressão. Utilizamos a classe de apoio **GeneID** para armazenar a identificação textual do gene. Estas duas classes relacionam-se através de seu número de identificação. A razão para possuir uma classe de apoio é economizar memória.

- **Atributo:** Representamos as condições experimentais do conjunto de dados por objetos desta classe. Dessa forma, armazenamos informações necessárias à execução do algoritmo, como o menor e o maior valores de expressão dos genes naquela condição experimental, além de um vetor com referências (ponteiros) para objetos da classe **Gene**. Além disso, mantemos esse vetor ordenado de forma crescente pelo método *ordena_atributo*, devido à primeira etapa do algoritmo, que consiste em encontrar unidades densas (ver seção 3.2.1).

- **Tempo:** Representamos nesta classe as diferentes fatias de tempo do estudo. Seus objetos são bastante simples e consistem apenas de uma identificação textual e um vetor de referências para os objetos do tipo **Atributo**.

- **Unidade:** Armazenamos nos objetos desta classe os clusters encontrados no conjunto de dados. Ela foi projetada para armazenar todos os clusters criados ao longo da execução do algoritmo, desde clusters unidimensionais encontrados na primeira etapa com a forma $C = G \times S$, onde $|S| = 1$, passando por clusters bidimensionais encontrados na segunda etapa do tipo $C = G \times S$, onde $|S| \geq 2$ e por fim clusters tridimensionais com a forma $C = G \times S \times T$, onde $|S| \geq 2$ e $|T| \geq 2$. Lembrando que nos três tipos de cluster, $|G| \geq 2$.

  Alcançamos o objetivo de armazenar três tipos de clusters sem criar três classes distintas utilizando objetos que contém sempre três vetores de referência. O primeiro para objetos da classe **Gene**, outro para objetos do tipo **Atributo** e o último vetor, com referências para objetos da classe **Tempo**. Porém, apesar dos objetos possuirem esses três vetores, clusters uni e bidimensionais possuem o vetor de referências para *Tempo* vazios.

Utilizando as classes definidas acima, implementamos o algoritmo usando a biblioteca STL (Standard Template Library). Essa biblioteca fornecida pela SGI (Silicon Graphics Incorporated) fornece um conjunto de classes e métodos altamente otimizados para operações envolvendo estruturas de dados. Além disso, por ser uma biblioteca criada utilizando o conceito de programação genérica [22], suas classes são parametrizadas e vêm na forma de *templates*. Esta funcionalidade significa que podemos criar, por exemplo, vetores ou árvores para nossos próprios tipos de dados, como *Gene* ou *Atributo*.

4.1.2 Algoritmos de validação

Para ter certeza que os resultados obtidos na execução dos algoritmos fazem sentido, é preciso utilizar índices de qualidade. A validação dos resultados de clustering ainda é um problema aberto, pois não existe um índice de qualidade que não favoreça algum método de clustering.

Definimos a etapa de validação dos resultados a partir de dois estudos conduzidos por Handl *et al.* [15] e Halkidi [13]. Ambos tratam do mesmo tema sob diferentes pontos

de vista: enquanto o primeiro utiliza exemplos de biologia computacional, mais especificamente expressão gênica, o segundo é mais genérico e não utiliza nenhuma aplicação específica dos métodos de clustering.

Segundo Handl *et al.*, existem duas principais vertentes relativas à validação de resultados de clustering: a primeira diz respeito a validar um novo algoritmo e a segunda, validar resultados obtidos após executar um algoritmo conhecido em um novo conjunto de dados. Como desenvolvemos um novo algoritmo de clustering, nosso intuito foi comparar os resultados da nossa solução à alternativa existente, chamada TriCluster.

Para esta finalidade, seguindo as recomendações de Handl *et al.*, desenvolvemos conjuntos de dados com classes conhecidas e utilizamos três medidas de *validação externa*. Estas medidas comparam a classificação correta dos objetos à classificação obtida pelos dois algoritmos. Além disso, também utilizamos um conjunto de dados reais, com informações relativas a um estudo de expressão gênica conduzido por Spellman *et al.* [25]. Para validar os resultados dos algoritmos em dados reais (cuja classificação correta dos objetos não é conhecida), utilizamos duas medidas de *validação interna*.

A seguir descreveremos o processo de criação dos conjuntos de dados e as características das medidas de validação utilizadas.

Conjuntos de dados

Algoritmos de clustering favorecem a descoberta de algum tipo de cluster. Por exemplo, o algoritmo DBSCAN [9] tem facilidade em encontrar clusters com objetos conectados (muito próximos), como os da figura 4.2a. Entretanto, este algoritmo não é eficaz como o K-means [20] em diferenciar clusters com separação espacial, como os da figura 4.2b.

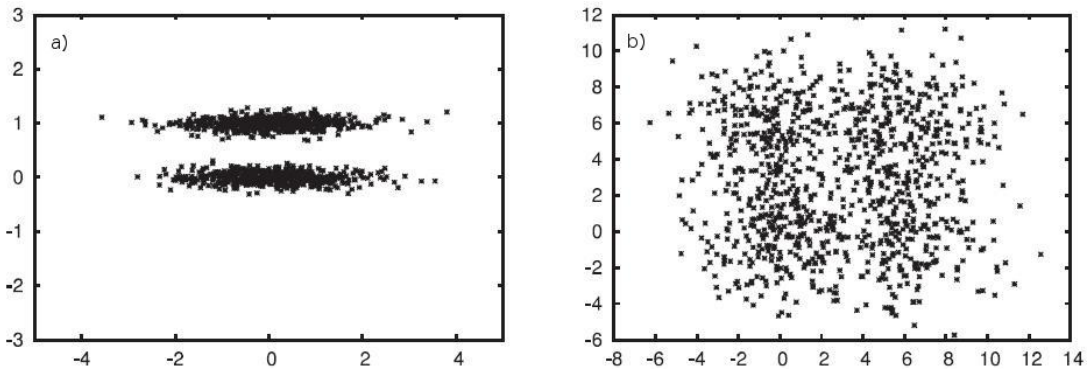

Figura 4.2: Diferentes tipos de clusters em duas dimensões. Na figura a), podemos ver dois clusters, cujos objetos próximos criam clusters alongados. Na figura b) existem 4 clusters cujos objetos apesar da separação espacial, são equi-distantes dos respectivos centróides. Imagem obtida de Handl *et al.*[15].

Para testar a qualidade do algoritmo desenvolvido neste estudo, criamos conjuntos de dados com dois tipos de clusters que não favorecem nenhum algoritmo conhecido. Desta forma acreditamos ser difícil tanto para o nosso algoritmo quanto para qualquer outro determinar a estrutura correta desses conjuntos de dados.

Utilizamos os geradores de conjuntos de dados criado por Handl e Knowles [14], disponíveis em http://dbkgroup.org/handl/. Originalmente estes geradores criavam apenas conjuntos do tipo $G \times S$, e para nosso problema precisávamos de conjuntos de dados do tipo $G \times S \times T$, por isso modificamos os algoritmos dos geradores para criar conjuntos de dados tridimensionais.

Descrevemos a seguir as caracteríticas matemáticas dos dois tipos de clusters presentes nos conjuntos de dados criados.

Clusters Gaussianos

Este primeiro tipo de cluster é gerado utilizando a distribuição normal multivariada. Segundo Handl e Knowles [15], as matrizes de covariância precisam ser simétricas e positivas. Este objetivo é alcançado construindo as matrizes com diagonais dominantes e contendo elementos positivos.

Um esquema passo a passo para a criação desses clusters é:

1. A média deve esta uniformemente distribuída no intervalo $[-10.0,\ 10.0]$

2. Os elementos não pertencentes à diagonal da matriz de covariância são números aleatórios no intervalo $[-1.0,\ 1.0]$, com uma distribuição seguindo $\pm y$, $y = x^2$, onde x é um desvio uniforme aleatório no intervalo $[0.0,\ 1.0]$, e o sinal de y é definido de forma aleatória.

3. Os elementos na diagonal da matriz de covariância são gerados como a soma de todos os elementos de fora da diagonal e de mais um valor aleatório no intervalo $[0.0,\ 20 \cdot \sqrt{D}]$, com uma distribuição seguindo $\pm y$, $y = x^2$, onde x é um desvio uniforme aleatório em $[0.0,\ 1.0]$, o sinal de y é definido de forma aleatória e D é a dimensionalidade do cluster desejado.

Além disso, o uso da distribuição $y = x^2$ serve para criar clusters alongados no conjunto de dados. Na figura 4.3 podemos ver uma ilustração de clusters Gaussianos em duas dimensões.

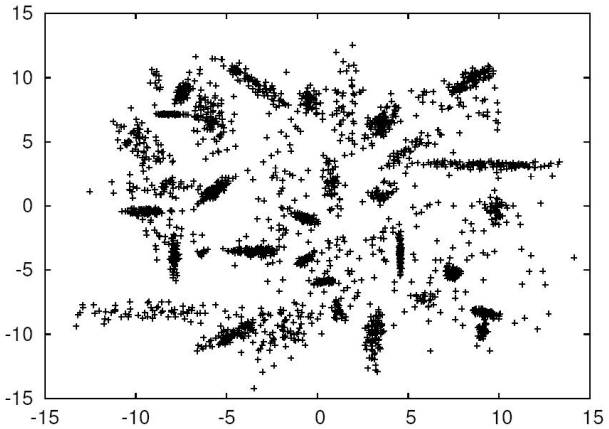

Figura 4.3: Ilustração de clusters Gaussianos em duas dimensões. Figura obtida de http://dbkgroup.org/handl/.

Clusters elipsóides

A criação deste segundo tipo de clusters é um pouco mais complexa, pois como será descrito mais adiante, utilizamos algoritmos genéticos para assegurar a criação de clusters coerentes. Este gerador cria clusters elipsóides com seu eixo principal possuindo uma orientação arbitrária, e seus limites são definidos por quatro parâmetros:

1. A origem, que também é o primeiro foco.

2. A distância interfocal, distribuída uniformemente no intervalo $[1.0, 3.0]$.

3. A orientação dos eixos principais, uniformemente distruibuídos entre a orientação dos demais eixos criados.

4. A maior distância Euclideana entre os dois focos, também uniformemente distruibuída no intervalo $[1.05, 1.15]$, equivalente a excentricidade variando no intervalo $[0.870, 0.952]$.

Em cada cluster, seus objetos são gerados segundo uma distribuição Gaussiana, estabelecendo uma distância entre um objeto criado e outro selecionado aleatoriamente do eixo principal. Esse objeto criado pode ser rejeitado se a distância estiver acima de um limite pré-estabelecido.

Depois que os objetos de todos os clusters são gerados, com a origem em 0.0 para todas as dimensões do conjunto de dados, um algoritmo genético é utilizado para re-arranjar os clusters, de forma que eles fiquem compactos como na figura 4.4. Nesse algoritmo genético, o objetivo é buscar um arranjo dos clusters modificando suas origens, de forma que possuam a menor sobreposição possível.

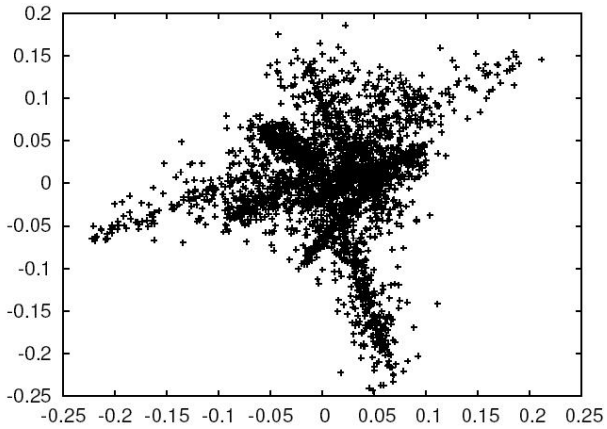

Figura 4.4: Ilustração de clusters Elipsóides em duas dimensões. Figura obtida de http://dbkgroup.org/handl/.

Utilizando os tipos de clusters descritos acima, criamos os 6 conjuntos de dados da tabela 4.1. Todos os conjuntos possuem 5 condições experimentais por 5 fatias de tempo.

O conjunto com dados reais foi obtido de um estudo conduzido por Spellman *et al.* [25] a respeito dos genes que influenciam os ciclos celulares da levedura *Saccharomyces cerevisae*. Esse conjunto de dados com 7679 genes, 13 condições experimentais e 4 fatias de tempo está disponível em http://genome-www.stanford.edu/cellcycle/.

Índices de qualidade

Uma etapa que demandou grande esforço neste projeto foi a implementação de programas para avaliar a qualidade dos resultados obtidos tanto pelo TriCluster quanto pelo algoritmo aqui desenvolvido.

Clusters Elipsóides		
Identificação	**Nro. Objetos**	**Nro. Clusters**
Eli_5c	7793	5
Eli_10c	14293	10
Eli_20c	4943	20
Clusters Gaussianos		
Identificação	**Nro. Objetos**	**Nro. Clusters**
Gau_5c	1488	5
Gau_10c	2753	10
Gau_20c	5993	20

Tabela 4.1: Conjuntos de dados sintéticos cujos classes de objetos são conhecidas.

4.1.3 Validação externa

Definimos um ambiente e um processo de validação externa para os dois algoritmos de clustering. A figura 4.5 ilustra o processo e suas partes são descritas abaixo:

- **Conjuntos de dados:** Estes são conjuntos de dados sintéticos descritos anteriormente, lembrando que as classes destes conjuntos de dados são conhecidas.
- **Clustering:** Esta etapa consiste em executar um dos dois algoritmos de clustering utilizando um dos conjuntos de dados como entrada e com uma certa configuração de parâmetros.
- **Classificação:** Depois da execução de um dos algoritmos, é obtido um resultado contendo a classificação dos objetos.
- **Índices Parciais:** Para compor os valores indicativos de qualidade dos resultados obtidos, utilizamos a classificação correta dos objetos (*Gold Standard*) para calcular índices parciais. Estes valores devem ser normalizados utilizando índices de classificações aleatórias. Este processo é necessário para eliminar as influências (bias) que as medidas de validação externa sofrem em relação ao número e tamanho dos clusters obtidos [13]. Amenizamos estas influências calculando o índice ajustado I_a pela equação:

$$I_a = (I - I_e)/(I_m - I_e) \tag{4.1}$$

  onde I_e é o valor do índice obtido para uma classificação aleatória, I_m é o maior valor que o índice pode assumir, e I é o valor do índice obtido pela classificação fornecida pelo algoritmo [15].

- **Índices Finais:** Estes são os valores das medidas de validação externa que utilizamos para comparar a qualidade dos resultados dos dois algoritmos estudados.

O gerador de números aleatórios utilizado foi o MT19937 [19], disponível na biblioteca GSL (Gnu Scientific Library). A razão para escolher este gerador de números aleatórios é que ele é rápido, possui um período de números primos de $2^{19937} - 1$ equi-distribuído em 623 dimensões. O gerador padrão do Linux não foi utilizado pois seus desenvolvedores

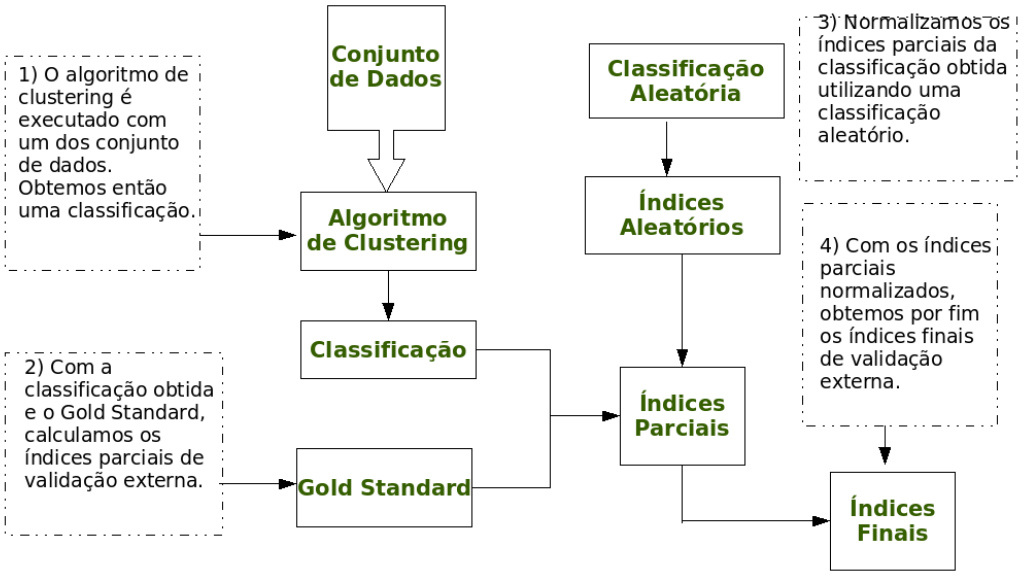

Figura 4.5: Etapas do processo de validação externa dos algoritmos de clustering.

alertam através das *man pages* que este não deve ser usado para aplicações científicas sérias.

Com o gerador de números aleatórios escolhido, foram criadas variações da classificação correta de cada conjunto de dados sintético utilizado neste estudo. Estas variações compreendem mudanças tanto no número de clusters, quanto na organização do conjunto de dados. Fizemos 100 modificações em cada conjuntos de dados, de forma que o número de clusters das modificações variou no intervalo $x = [1, 50]$, e para cada passo discreto deste intervalo, nós redistribuimos duas vezes os objetos dos clusters originais em x clusters.

Os índices utilizados para a validação externa foram:

- Coeficiente de Jaccard
- Pureza (Purity)
- Integridade (Completeness)

O coeficiente de Jaccard leva em conta o *gold standard* e a classificação obtida pelo algoritmo de clustering. Este índice considera todas as combinações formando pares de objetos (x_1, x_2), e classifica cada par como:

- **a**, se x_1 e x_2 pertencem ao mesmo cluster na classificação do algoritmo e no *gold standard*,
- **b**, se x_1 e x_2 pertencem ao mesmo cluster na classificação do algoritmo e a clusters diferentes no *gold standard*,
- **c**, se x_1 e x_2 pertencem a clusters diferentes na classificação do algoritmo e ao mesmo cluster no *gold standard*,
- **d**, se x_1 e x_2 pertencem a clusters diferentes na classificação do algoritmo e a clusters diferentes no *gold standard*.

O coeficiente de Jaccard é calculado por:

$$J_c = \frac{a}{a+b+c} \tag{4.2}$$

Desta forma, o índice tem valores no intervalo [0.0, 1.0], e quanto mais próximo a 1.0, melhor a qualidade do resultado obtido.

Este índice foi escolhido pois não penaliza as classificações indicadas pela letra **d**, ao contrário do Rand Index e suas variações. Esta característica é desejável pois como estamos criando clusters com sobreposição, pode ser que um mesmo objeto pertença a diferentes clusters e não deve ser considerado uma classificação incorreta.

Os outros dois índices, pureza e integridade podem ser facilmente confundidos, por isso utilizaremos o exemplo da figura 4.6 para apoiar a explicação.

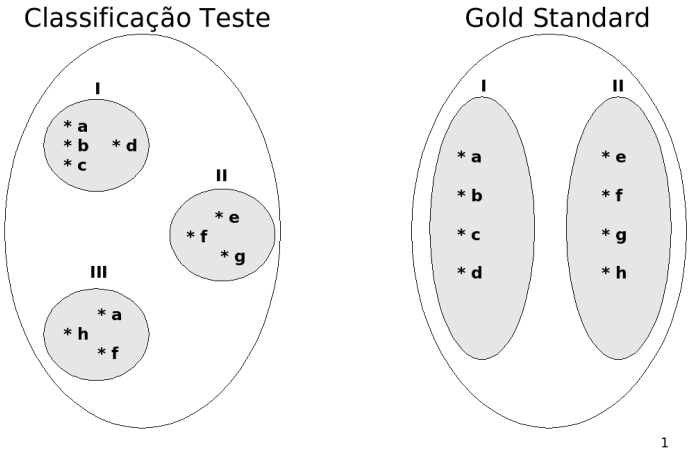

Figura 4.6: Após executarmos um algoritmo de clustering em um conjunto de dados cujas classes são conhecidas, temos a *Classificação Teste* (a esquerda) e o *gold standard*.

- **Pureza:** Selecionando cada cluster da classificação teste, determinamos qual fração de objetos está presente no mesmo cluster do *gold standard*. Assim, podemos ver que do cluster I da classificação teste, todos estão no cluster I do gold Standard (100%); do cluster II, todos os objetos estão no cluster II do gold standard (100%). No terceiro e último cluster da classificação teste, temos dois elementos fazendo parte do mesmo cluster (f e h), e apenas um (a) pertencente a outro cluster. Por isso, contamos sempre a semelhança com o cluster predominante do *gold standard*, neste caso o cluster II. O cálculo da pureza é dado por:

$$P = \frac{\text{fração cluster I} + \text{fração cluster II} + \text{fração cluster III}}{\text{número de clusters}} = \frac{100.0 + 100.0 + 75.0}{3}$$

$$= \frac{275.0}{3.0} = 91.6$$

- **Integridade:** Agora partindo do *gold standard*, verificamos que dos objetos do cluster I, todos estão no mesmo grupo na classificação teste (100%). Dentre os

objetos do cluster II do *gold standard*, apenas 3 dos 4 objetos estão no mesmo cluster (II) da classificação teste (75%), então, o cálculo da integridade é dado por:

$$C = \frac{\text{fração cluster I} + \text{fração cluster II}}{\text{número de clusters}} = \frac{100.0 + 75.0}{2.0} = 87.5$$

Estes dois índices possuem valores no intervalo [0.0, 1.0] e assim como o coeficiente de Jaccard, quanto mais próximos a 1.0, melhor o resultado. Assim, ao dividí-los por 100 temos, respectivamente para os índices de pureza e integridade, 0.91 e 0.87.

Vale notar que estes dois índices possuem tendências opostas, isto é, enquanto o valor da *pureza* vai ser máximo ao criar clusters compostos por um único objeto (singleton), a integridade vai ser máxima ao criar um cluster que inclua todos os elementos do conjunto de dados. Por isso, combinamos os dois índices através da média harmônica, também conhecida por medida-F, definida pela equação:

$$F = \frac{2 * pureza * integridade}{pureza + integridade} \tag{4.3}$$

No exemplo da figura 4.6 a medida-F é dada por:

$$F = \frac{2 * 0.91 * 0.87}{0.91 + 0.87} = 0.88$$

Assim, para avaliar a qualidade dos resultados dos algoritmos em conjuntos de dados cujas classes já são conhecidas, utilizamos o coeficiente de Jaccard e a média harmônica dos índices de pureza e integridade.

4.1.4 Validação interna

O uso de medidas de validação interna é necessário pois não sabemos a classificação correta dos objetos pertencentes ao conjunto de dados reais, e segundo o estudo de Halkidi [13], o ideal é encontrar clusters compactos, com uma separação razoável entre si e cujos vizinhos próximos tenham sido colocados no mesmo grupo.

Utilizamos neste estudo duas medidas de validação interna, propostas por Handl e Knowles [14], que avaliam a compactação (*compacteness*) e a conectividade (*conectedness*) dos objetos do cluster. Essas medidas foram escolhidas dentre as diversas opções, pois refletem duas características desejáveis na tarefa de descobrir grupos de objetos relacionados: primeiro, bonifica clusters compactos, cujos objetos estão próximos no espaço numérico e em segundo lugar, valoriza o fato de objetos vizinhos (*nearest neighbors*) pertencerem ao mesmo cluster. Desta maneira, segundo a autora do estudo, ao contrário de outras medidas, esses índices não favorecem a descoberta de clusters com nenhuma forma em particular. Além disso, estas medidas não penalizam a existência de clusters sobrepostos.

Para avaliar o nível de compactação dos clusters (*compacteness*), medimos o *desvio total* dos clusters resultantes da execução de cada algoritmo. Esse índice é obtido somando a distância de todos os objetos do conjunto de dados aos respectivos centróides de seus clusters:

$$Dev(C) = \sum_{C_k \in C} \sum_{i \in C_k} \delta(i, \mu_k) \tag{4.4}$$

onde C é o conjunto de todos os clusters encontrados, μ_k é o centróide do cluster k e $\delta(i,j)$ é a distância Euclediana entre os objetos x_i e x_j. Desta forma, se o valor do índice é pequeno isto indica a descoberta de clusters compactos. Neste estudo, o centróide dos clusters foi encontrado através do cálculo da média dos valores dos objetos em cada uma de suas dimensões.

O índice utilizado para quantificar a conectividade dos clusters é chamado *Connectivity* e serve para avaliar o quanto objetos vizinhos no espaço numérico foram colocados pelo algoritmo no mesmo cluster. Podemos calcular seu valor por:

$$Conn(C) = \sum_{i=1}^{N}(\sum_{j=1}^{L} d(i,j)) \tag{4.5}$$

onde

$$d(i,j) = \begin{cases} 0 & \text{se } i \text{ e } j \text{ pertencem ao mesmo cluster} \\ \frac{1}{j} & \text{caso contrário} \end{cases}$$

Neste caso, L é um parâmetro que determina o número de vizinhos proximos a serem considerados ao cálcular este índice, que será tanto melhor quanto menor for seu valor.

É importante ressaltar que os detalhes da escolha de parâmetros para o cálculo dos índices de validação interna e externa serão apresentados no capítulo 5, assim como os resultados obtidos com estas escolhas.

4.1.5 Ambiente experimental

Segundo o estudo de Halkidi [13], para obter os melhores resultados em um estudo envolvendo clustering, é necessário executar o mesmo algoritmo diversas vezes, variando seus parâmetros e armazenando os resultados obtidos.

Neste estudo, realizamos este procedimento tanto para nosso algoritmo quanto para o TriCluster. Para esta tarefa, utilizamos as seguintes máquinas do ICMC-USP:

- Cluster LABIC (Laboratório de Inteligência Computacional). Este cluster conta com 11 máquinas de processadores Intel(R) Xeon(TM) CPU $2.40GHz$, 1 GB de memória RAM, sistema operacional Linux (Debian) e kernel versão 2.4.26. Este cluster também conta com o Open Mosix (http://openmosix.sourceforge.net/). A imagem única do sistema criada pelo Open Mosix faz com que alguns processos sejam migrados automaticamente para outros nós do cluster quando houver sobrecarga de uma de suas partes.

- Cluster Campus II. Este cluster conta com 16 nós Pentium 4 de $3.4Ghz$ $64bits$, $3GB$ de memória RAM, sistema operacinal Linux, kernel versão $2.6.13-15$. O mecanismo de operação deste cluster foi o Open MPI (Message Passing Interface) versão 1.1.2 disponível em www.open-mpi.org.

- Maquinas do STI (Suporte Técnico em Informática). Tivemos a disposição duas máquinas com 4 processadores AMD Opteron(tm), $7.4GB$ de memória RAM, sistema operacional Gentoo Linux com kernel versão $2.6.9 - 14$. O mecanismo de distribuição de carga destas máquinas foi a biblioteca Pthreads (POSIX Threads), disponível no próprio sistema operacional.

Para resolver a dificuldade de lidar com sistemas tão heterogêneos, escrevemos um script cuja função é criar diversos processos, executando um dos dois algoritmos com uma

pequena variação em seus parâmetros. Assim, os processos são migrados entre os nós (e processadores), executados e seus resultados gravados em um arquivo, em uma partição compartilhada NFS (Network File System).

Capítulo

5

Resultados e Discussão

Conforme descrito no capítulo 4, realizamos experimentos comparativos entre o algoritmo desenvolvido neste estudo e o TriCluster. Esses experimentos foram feitos utilizando diferentes conjuntos de dados e um grande número de variações nos parâmetros dos algoritmos. Neste capítulo forneceremos os detalhes desses experimentos e os resultados obtidos. Através da análise dessas informações encontramos as características das duas soluções, incluindo a dedução de suas vantagens e desvantagens.

5.1 Execução dos Algoritmos

Dividimos a apresentação de resultados em duas partes. A primeira consiste em uma avaliação individual dos algoritmos e a segunda consiste em uma comparação entre os resultados obtidos por ambos em conjuntos de dados sintéticos e reais

Para avaliar separadamente o nosso algoritmo desenvolvido e o TriCluster, fizemos experimentos com os conjuntos de dados descritos na seção 4.1.2. Executamos os experimentos de validação externa buscando encontrar clusters em subconjuntos de atributos e subconjuntos de fatias de tempo. Adotamos essa prática para verificar se os algoritmos são capazes de encontrar primeiramente clusters pequenos e gradativamente aumentá-los até seu tamanho máximo. Os dois algoritmos possuem os parâmetros min_g, min_s e min_t que estabelecem respectivamente os números mínimos de genes, condições experimentais e fatias de tempo para os clusters. Entretanto, como é difícil saber de antemão o tamanho mínimo destas características, o ideal seria começar com valores pequenos e confiar no poder de expansão de clusters oferecido pelos algoritmos.

Foram realizados experimentos variando gradativamente os parâmetros dos dois algoritmos. No caso do nosso algoritmo, além dos parâmetros min_g, min_s e min_t descritos anteriormente, fizemos mudanças no parâmetro δ, que representa o fator de multiplicação da densidade mínima, como explicado na seção 3.2.1.

No caso do TriCluster, o parâmetro cujo valor foi modificado ao longo das execuções é chamado de ϵ e diz respeito ao valor da razão mínima entre os genes do mesmo cluster [30]. Um ponto importante é a característica do TriCluster de unir (*Merge*) ou apagar (*Delete*)

clusters muito semelhantes, como foi explicado na seção 2.4.5. Para recordar, os autores do TriCluster consideraram clusters com um certo grau de sobreposição muito parecidos e forneceram a possibilidade de unir ou apagar esses clusters. Essa característica foi compreendida, mas optou-se por não implementá-la na solução desenvolvida neste trabalho de mestrado pelos seguintes motivos:

- *Apagar clusters:* Apagando clusters muito similares deixaríamos para trás as informações que diferenciam esses clusters. Essa diferença deve ser preservada, pois técnicas de mineração de dados mais eficazes estão sendo desenvolvidas e em algum momento estas informações (que seriam consideradas desnecessárias) podem fazer diferença no processo de criação ou validação de hipóteses biológicas.

- *Unir clusters:* Quando o algoritmo de clustering foi desenvolvido, ele foi programado para encontrar clusters segundo um critério. Desta forma, se o algoritmo encontrou 2 clusters e não apenas 1, é porque aqueles clusters devem possuir separadamente características exclusivas.

As tabelas 5.1, 5.2 e 5.3 mostram os experimentos executados para o nosso algoritmo e para o TriCluster. No nosso algoritmo e no TriCluster o passo do incremento no parâmetro min_g foi 1, e no parâmetro δ foi 0.1. Para os parâmetros min_s e min_t, foram feitas apenas combinações horizontais. Apenas experimentos como 3×3 ou 5×5 foram executados e nunca experimentos do tipo 2×4. Desta maneira, considerando todas as mudanças nos parâmetros e considerando os seis conjuntos de dados temos um total de 74.520 execuções do nosso algoritmo e 165.174 o TriCluster.

A diferença entre os experimentos envolvendo o TriCluster e o nosso algoritmo, se deu em função da enorme dificuldade em encontrar o intervalo de parâmetros válidos para o TriCluster. Para as execuções do TriCluster foi impossível determinar apenas um intervalo de variação de seus parâmetros, pois uma grande parte das execuções com um certo conjunto de parâmetros resultava em tempos de execuções excessivamente longos (acima de 10 horas), ou em uso extensivo de memória, consumindo todos os recursos das máquinas. Essas situações foram consideradas fora do escopo deste estudo.

Seria cientificamente mais correto modificar o código fonte do TriCluster para resolver estes problemas, utilizando ferramentas como o GDB (*Gnu Debugger)* e o *GProof*. Entretanto os autores Zhao e Zaki [30] não forneceram o código fonte, e apenas depois de várias tentativas de contato eles forneceram apenas o arquivo executável compilado para Linux na arquitetura de 32 bits.

Outra alternativa seria reimplementar o algoritmo, pois ele é determinístico e acreditamos ser possível obter os mesmos resultados presentes no trabalho original. Entretanto, dada a limitação de tempo no desenvolvimento desta dissertação de mestrado, a solução foi utilizar apenas os maiores intervalos de valores de parâmetros onde existiam resultados válidos.

5.2 Avaliação Individual das Soluções

5.2.1 Resultados do nosso algoritmo

Após a execução do algoritmo segundo o plano descrito na tabela 5.1, foram calculados o coeficiente de Jaccard e dois outros índices, a *pureza (purity)* e a *integridade (comple-*

Conjunto de Dados	min_s	min_t	min_g	δ
Elipsóide com 5 clusters (Eli_5c)	2	2	$[5, 50]$	$[1.1, 5.6]$
	3	3		
	4	4		
	5	5		
Elipsóide com 10 clusters (Eli_10c)	2	2	$[5, 50]$	$[1.1, 5.6]$
	3	3		
	4	4		
	5	5		
Elipsóide com 20 clusters (Eli_20c)	2	2	$[5, 50]$	$[1.1, 5.6]$
	3	3		
	4	4		
	5	5		
Gaussianos com 5 clusters (Gau_5c)	2	2	$[5, 50]$	$[1.1, 5.6]$
	3	3		
	4	4		
	5	5		
Gaussianos com 10 clusters (Gau_10c)	2	2	$[5, 50]$	$[1.1, 5.6]$
	3	3		
	4	4		
	5	5		
Gaussianos com 20 clusters (Gau_20c)	2	2	$[5, 50]$	$[1.1, 5.6]$
	3	3		
	4	4		
	5	5		

Tabela 5.1: Valores de parâmetros utilizados nos experimentos envolvendo nosso algoritmo.

C. Dados	min_s	min_t	min_g	ϵ	**Passo** ϵ	**Total**
Eli_5c	2	2	[5, 25]	[0.000009, 0.00005]	0.000001	820
	2	2	[26, 100]	[0.0003, 0.0009]	0.0001	444
	3	3	[10, 30]	[0.0001, 0.0005]	0.00001	800
	3	3	[31, 60]	[0.0005, 0.0008]	0.00001	870
	3	3	[61, 100]	[0.0007, 0.002]	0.0001	507
	4	4	[10, 50]	[0.001, 0.002]	0.00005	800
	4	4	[51, 100]	[0.004, 0.006]	0.0001	980
	5	5	[10, 50]	[0.002, 0.009]	0.0001	2800
	5	5	[51, 100]	[0.002, 0.005]	0.00001	14700
Eli_10c	2	2	[10, 50]	[0.0001, 0.0002]	0.000005	800
	2	2	[51, 100]	[0.0002, 0.0004]	0.00001	980
	3	3	[10, 50]	[0.00006, 0.0004]	0.00001	1360
	3	3	[51, 100]	[0.0004, 0.0009]	0.0001	245
	4	4	[10, 50]	[0.0004, 0.004]	0.0001	1764
	4	4	[51, 100]	[0.004, 0.007]	0.0002	735
	5	5	[10, 50]	[0.002, 0.009]	0.0001	2800
	5	5	[51, 100]	[0.002, 0.005]	0.00001	14700
Eli_20c	2	2	[10, 50]	[0.00009, 0.0005]	0.00001	1640
	2	2	[51, 100]	[0.0005, 0.002]	0.0001	735
	3	3	[10, 50]	[0.0005, 0.003]	0.0001	1000
	3	3	[51, 100]	[0.003, 0.006]	0.0001	1470
	4	4	[10, 50]	[0.001, 0.006]	0.0001	2000
	4	4	[51, 100]	[0.007, 0.02]	0.0001	6370
	5	5	[10, 50]	[0.002, 0.009]	0.0001	2800
	5	5	[51, 100]	[0.002, 0.005]	0.00001	14700

Tabela 5.2: Valores de parâmetros utilizados nos experimentos envolvendo o algoritmo TriCluster.

C. Dados	min_s	min_t	min_g	ϵ	**Passo ϵ**	**Total**
Gau_5c	2	2	$[10, 50]$	$[0.0001,\ 0.005]$	0.0001	1960
	2	2	$[51, 100]$	$[0.005,\ 0.009]$	0.0001	1960
	3	3	$[10, 30]$	$[0.002, 0.005]$	0.0001	600
	3	3	$[31, 60]$	$[0.009, 0.02]$	0.001	319
	3	3	$[61, 100]$	$[0.02, 0.05]$	0.001	1470
	4	4	$[10, 50]$	$[0.0055, 0.01]$	0.0001	1800
	4	4	$[51, 100]$	$[0.01, 0.09]$	0.001	3920
	5	5	$[10,\ 50]$	$[0.002,\ 0.009]$	0.0001	2800
	5	5	$[51,\ 100]$	$[0.002,\ 0.005]$	0.00001	14700
Gau_10c	2	2	$[10, 200]$	$[0.0001, 0.001]$	0.00001	3420
	2	2	$[10, 200]$	$[0.001, 0.002]$	0.0001	380
	3	3	$[10, 30]$	$[0.002, 0.005]$	0.0001	600
	3	3	$[31, 60]$	$[0.009, 0.02]$	0.001	319
	3	3	$[61, 100]$	$[0.02, 0.05]$	0.001	1117
	4	4	$[10, 50]$	$[0.0025, 0.02]$	0.0001	7000
	4	4	$[51, 100]$	$[0.02, 0.07]$	0.001	2450
	5	5	$[10,\ 50]$	$[0.002,\ 0.009]$	0.0001	2800
	5	5	$[51,\ 100]$	$[0.002,\ 0.005]$	0.00001	14700
Gau_20c	2	2	$[10, 50]$	$[0.00009,\ 0.0005]$	0.00001	1640
	2	2	$[51, 100]$	$[0.0005,\ 0.002]$	0.0001	735
	3	3	$[10, 50]$	$[0.0001,\ 0.0006]$	0.00002	1000
	3	3	$[51, 100]$	$[0.0007,\ 0.002]$	0.00005	1274
	4	4	$[10, 50]$	$[0.0004, 0.004]$	0.0001	1440
	4	4	$[51, 100]$	$[0.004, 0.009]$	0.0001	2450
	5	5	$[10,\ 50]$	$[0.002,\ 0.009]$	0.0001	2800
	5	5	$[51,\ 100]$	$[0.002,\ 0.005]$	0.00001	14700

Tabela 5.3: Valores de parâmetros utilizados nos experimentos envolvendo o algoritmo TriCluster.

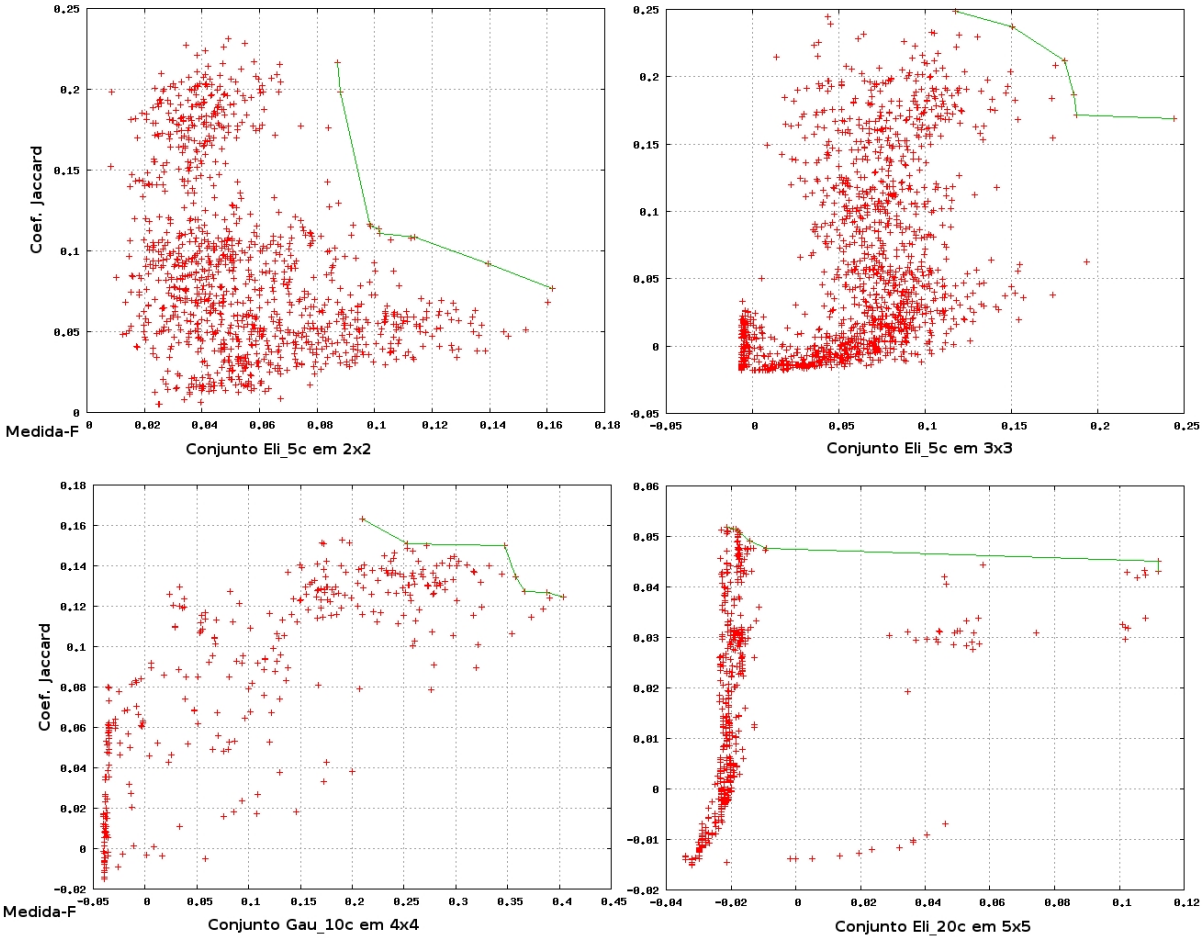

Figura 5.1: Execuções do algoritmo com um determinado conjunto de parâmetros em alguns conjuntos de dados nos 4 sub-espaços considerados. A linha é a fronteira de Pareto, contendo as melhores soluções. Note que algumas execuções possuem índices de qualidade negativos, cujo significado será explicado posteriormente. Vale notar que não colocamos os gráficos de todos os conjuntos de dados em todos os sub-espaços por restrições de espaço.

teness). Lembrando que que os valores destes dois últimos foram combinados através da média harmônica *(Medida-F)* (seção 4.1.2).

Como possuíamos um contingente enorme de valores, optou-se por encontrar medidas que pudessem condensar estes resultados para facilitar sua interpretação e entender o comportamento do algoritmo em relação aos diferentes conjuntos de dados e variações de sub-espaços. Utilizando os dois índices de validação externa de cada solução encontrada pelo algoritmo, nós criamos uma fronteira de Pareto para cada conjunto de dados em cada sub-espaço. Para fazer parte desta fronteira, cada solução com seus respectivos índices de qualidade deve possuir valores melhores ou iguais em todos os índices e pelo menos em um deles, o valor deve ser apenas superior. Podemos ver na figura 5.1 um gráfico de soluções encontradas nas execuções do nosso algoritmo e a fronteira de Pareto encontrada, contendo as melhores soluções.

Calculamos para cada conjunto de dados e para cada sub-espaço a média e o desvio padrão tanto do coeficiente de Jaccard quanto da Medida-F de duas formas, primeiro de todas as soluções encontradas e depois apenas das soluções presentes na fronteira. Podemos ver nas figuras 5.2 e 5.3, a diferença entre a média dos índices entre todas as

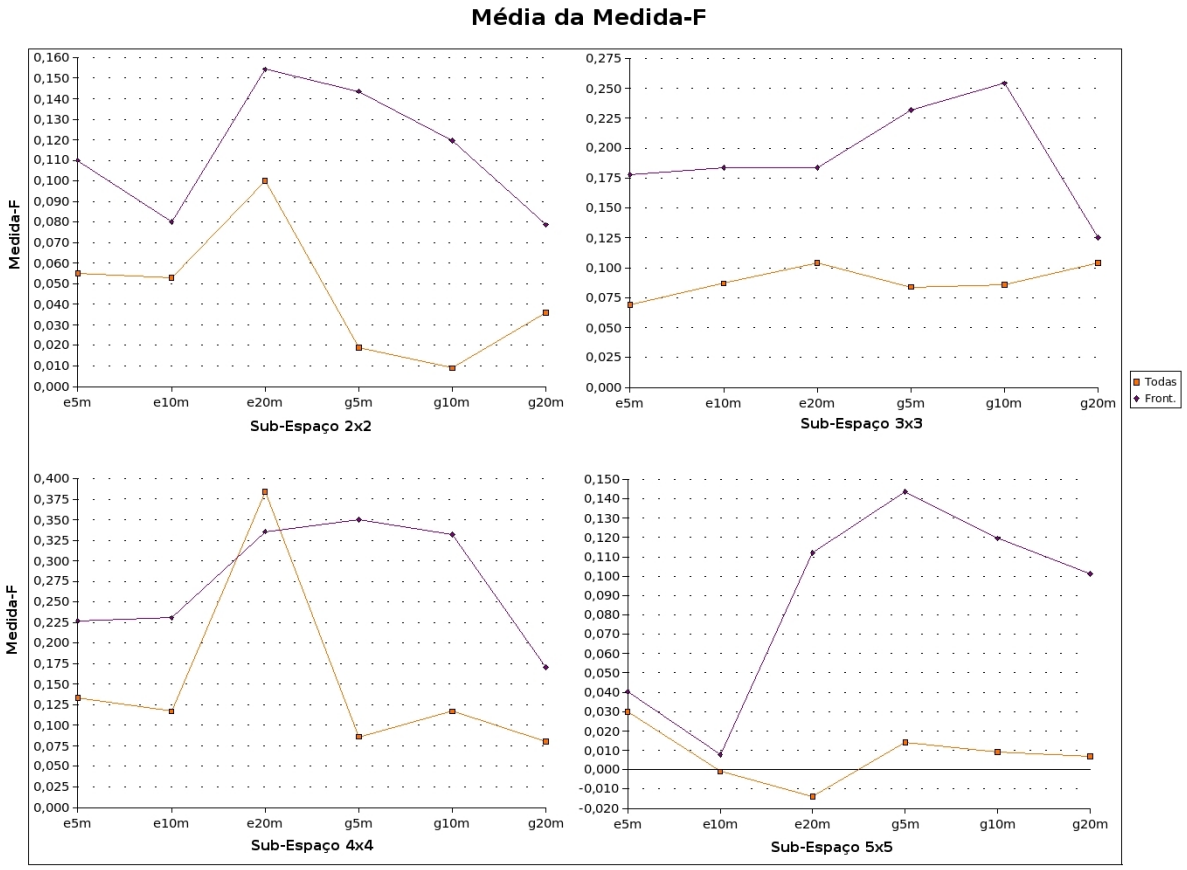

Figura 5.2: Comparação das médias da Medida-F de todas as soluções e apenas as soluções presentes na fronteira de Pareto, considerando todos os conjuntos de dados e todos os sub-espaços.

soluções e as soluções da fronteira.

Podemos ver na tabela 5.4 os valores da média e do desvio padrão para os diferentes conjuntos de dados e os diferentes sub-espaços considerados, calculados levando em consideração todos as soluções obtidas. Na tabela 5.5 podemos ver a média e o desvio padrão dos mesmos índices, mas levando em conta somente as soluções presentes nas fronteiras de Pareto.

Levando em conta a diferença entre os valores relativos aos índices de qualidade de todas as soluções e apenas as soluções da fronteira de Pareto, decidimos utilizar os resultados da fronteira para estudar o comportamento do algoritmo nos diferentes conjuntos de dados. Essa decisão foi tomada para evitar que a heterogeneidade e a grande quantidade de soluções inválidas (explicadas mais a frente) afetassem a análise dos pontos fortes e fracos do nosso algoritmo. Foram construídos os gráficos da figura 5.4 e 5.5 com os índices de qualidade destes resultados.

Destes gráficos podemos inferir o seguinte a respeito do algoritmo proposto:

- Verificando os gráficos relativos ao coeficiente de Jaccard observamos que para conjuntos de dados Gaussianos, conforme o número de clusters aumenta nos conjuntos de dados é mais difícil classificar os pares corretamente, exceto para um sub-espaço de tamanho 2×2. Além disso, o aumento no tamaho do sub-espaço considerado faz com que a classificação de pares seja mais eficaz para os conjuntos com clusters Gaussianos, até o sub-espaço máximo 4×4.

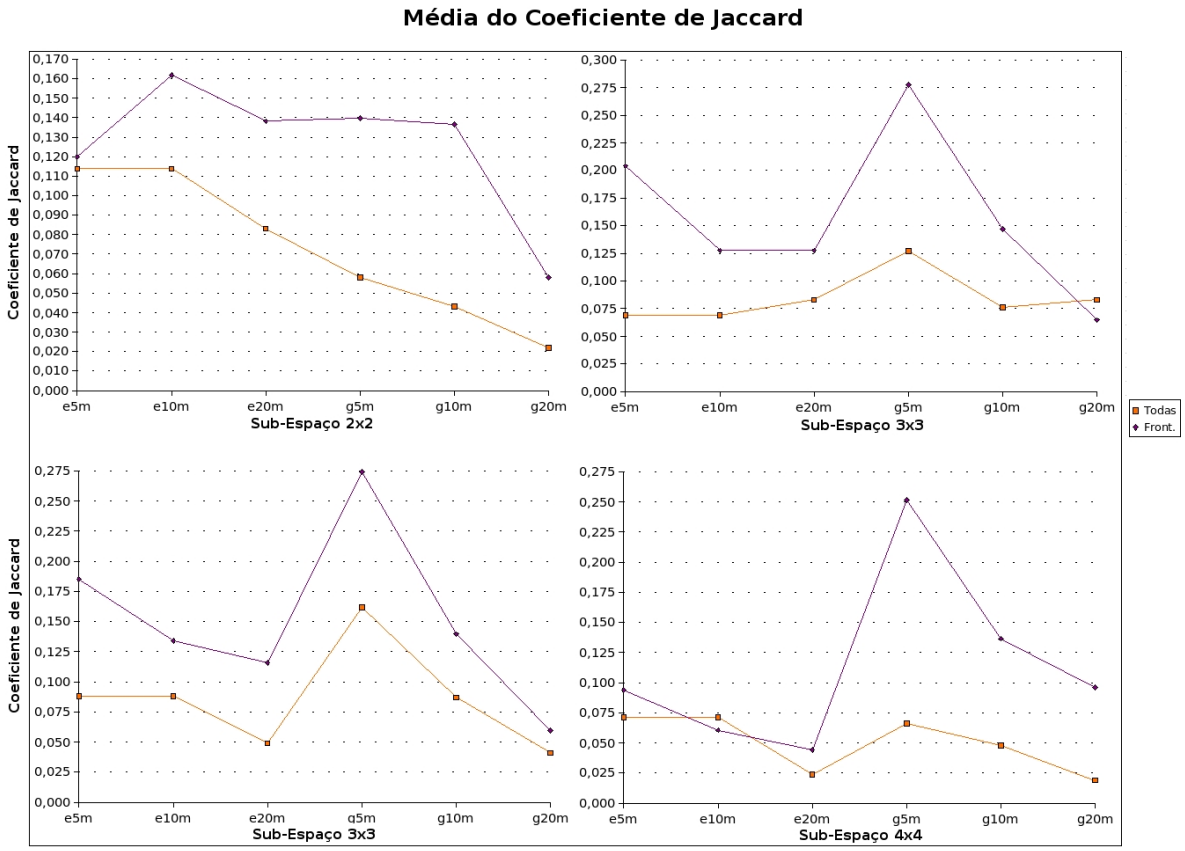

Figura 5.3: Comparação das médias do coeficiente de Jaccard de todas as soluções e apenas as soluções presentes na fronteira de Pareto, considerando todos os conjuntos de dados e todos os sub-espaços.

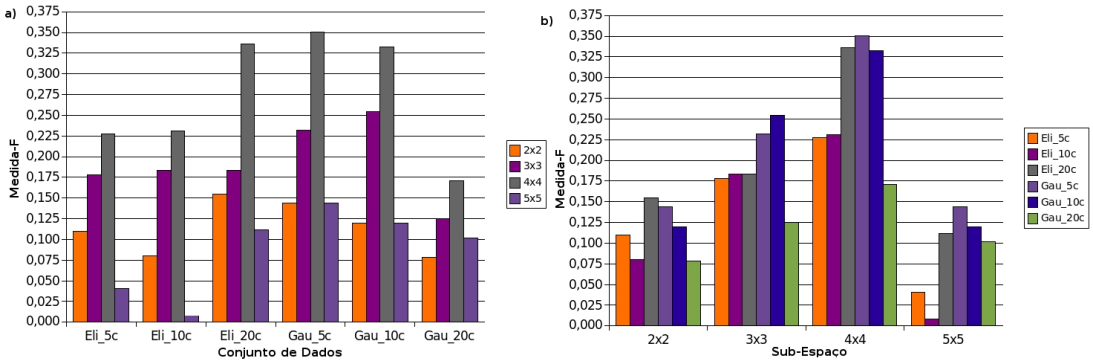

Figura 5.4: Gráficos com a média da Medida-F das soluções presentes na fronteira de Pareto, variando de acordo com a) os diferentes conjuntos de dados e b) os diferentes sub-espaços.

Medida-F

Conj. Dados	Eli_5c		**Eli_10c**		**Eli_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	0.055	0.027	0.053	0.024	0.100	0.350
3×3	0.069	0.036	0.087	0.045	0.104	0.062
4×4	0.133	0.056	0.117	0.055	0.384	0.084
5×5	0.030	0.024			-0.014	0.024
Conj. Dados	**Gau_5c**		**Gau_10c**		**Gau_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	0.019	0.039	0.009	0.206	0.036	0.028
3×3	0.084	0.074	0.086	0.079	0.104	0.060
4×4	0.086	0.099	0.117	0.125	0.080	0.051
5×5	0.014	0.170	0.009	0.206	0.007	0.243

Coeficiente de Jaccard

Conj. Dados	**Eli_5c**		**Eli_10c**		**Eli_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	0.114	0.867	0.073	0.043	0.083	0.026
3×3	0.069	0.343	0.064	0.050	0.083	0.062
4×4	0.088	0.053	0.096	0.408	0.049	0.088
5×5	0.071	0.023			0.024	0.201
Conj. Dados	**Gau_5c**		**Gau_10c**		**Gau_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	0.058	0.065	0.043	0.049	0.022	0.024
3×3	0.127	0.062	0.076	0.180	0.083	0.305
4×4	0.162	0.822	0.087	0.048	0.041	0.014
5×5	0.066	0.292	0.048	0.180	0.019	0.138

Tabela 5.4: Tabela com os valores da média (MED) e do desvio padrão (DP) do coeficiente de Jaccard e da combinação da medida de Pureza e da Integridade, através da média harmônica (Medida-F). Estes valores foram obtidos utilizando os valores das 2070 execuções do algoritmo para cada conjunto de dados e para cada sub-espaço.

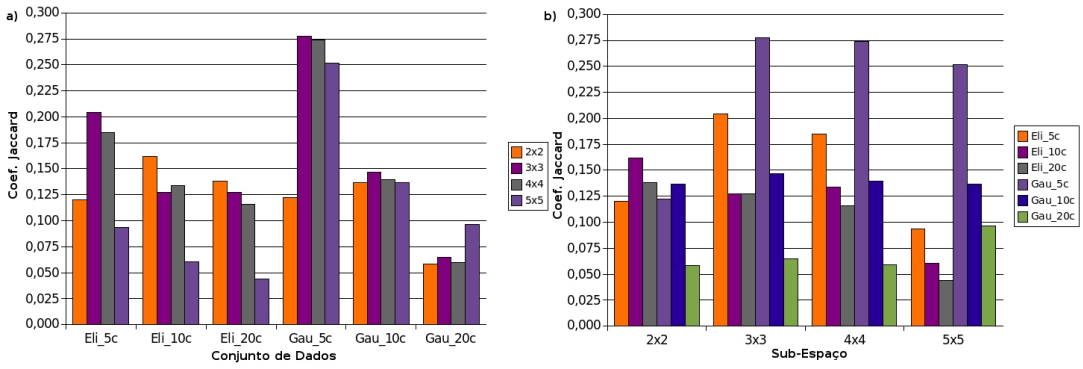

Figura 5.5: Gráficos com a média do coeficiente de Jaccard das soluções presentes na fronteira de Pareto, variando de acordo com a) os diferentes conjuntos de dados e b) os diferentes sub-espaços.

Medida-F

Conj. Dados	**Eli_5c**		**Eli_10c**		**Eli_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	0.110	0.027	0.080	0.030	0.154	0.060
3×3	0.178	0.042	0.184	0.011	0.184	0.011
4×4	0.227	0.092	0.231	0.035	0.336	0.044
5×5	0.040	0.028	0.008	0.007	0.112	0.000
Conj. Dados	**Eli_5c**		**Eli_10c**		**Eli_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	0.144	0.021	0.120	0.017	0.079	0.024
3×3	0.232	0.053	0.254	0.017	0.125	0.039
4×4	0.350	0.037	0.332	0.073	0.171	0.030
5×5	0.144	0.021	0.120	0.017	0.101	0.006

Coeficiente de Jaccard

Conj. Dados	**Eli_5c**		**Eli_10c**		**Eli_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	0.120	0.040	0.162	0.062	0.138	0.035
3×3	0.204	0.034	0.128	0.037	0.128	0.037
4×4	0.185	0.010	0.134	0.016	0.116	0.007
5×5	0.094	0.024	0.061	0.022	0.044	0.001
Conj. Dados	**Eli_5c**		**Eli_10c**		**Eli_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	0.138	0.092	0.137	0.038	0.058	0.018
3×3	0.278	0.014	0.147	0.013	0.065	0.006
4×4	0.274	0.042	0.140	0.015	0.059	0.006
5×5	0.252	0.042	0.137	0.038	0.096	0.005

Tabela 5.5: Tabela com os valores da média (MED) e do desvio padrão (DP) tanto do coeficiente de Jaccard como da combinação da medida de Pureza e da Integridade através da média harmônica (Medida-F). Estes valores dizem respeito às soluções presentes na fronteira de Pareto encontradas pelo nosso algoritmo.

- Para conjuntos de dados com clusters do tipo elipsóide, o tamanho do sub-espaço selecionado não causa grande influência na mudança da qualidade. Vale notar que no sub-espaço 4×4, o conjunto de dados Eli_20c teve uma boa classificação individual de seus objetos. Entretanto, esse comportamento é verificado em todos os sub-espaços exceto no sub-espaço 5×5, onde a qualidade cai consideravelmente.

- Analisando o gráfico das médias da Medida-F percebemos que na para a maioria dos conjuntos de dados Gaussianos e elipsóides a qualidade da classificação de pontos individuais cresce com o aumento no tamanho do sub-espaço, porém isto vale apenas até o sub-espaço máximo 4×4. Outro aspecto é que para os conjuntos de dados contendo clusters elipsóides quanto mais clusters melhor a classificação (até o sub-espaço máximo 4×4). Nos conjuntos de clusters gaussianos o número de clusters nos conjuntos de dados causa variação de até 20% no valor da Medida-F, mas não foi observado um comportamento padronizado relacionando o número de clusters do conjunto de dados e a Medida-F.

- O algoritmo possui mais facilidade para classificar pares de objetos em conjuntos de dados Gaussianos com poucos clusters. Esta característica diz respeito a diferença entre a média dos índices para conjuntos de dados com diferentes números de clusters. Podemos verificar que para todos os sub-espaços a qualidade geral está ordenada de forma que os conjuntos de dados com apenas 5 clusters foram os que apresentaram melhores resultados.

  Pela característica dos geradores de conjuntos de dados sintéticos utilizados (descritos no capítulo 4), o número de objetos é praticamente constante nos conjuntos gerados, o que muda é o número de clusters. Assim, quanto maior o número de clusters, menor será o número de objetos por cluster. Dessa maneira, percebemos que nosso algoritmo tem maior facilidade em encontrar clusters grandes, e maior dificuldade em considerar a mudança de valores de objetos presentes em clusters pequenos.

- O algoritmo não é bom para encontrar clusters em todos os atributos simultaneamente. Como podemos verificar tanto na figura 5.4 quanto na figura 5.5, de maneira geral, as médias dos índices de qualidade para os conjuntos de dados aumentam conforme o tamanho mínimo dos sub-espaços aumenta (definidos pelos parâmetros min_s e min_t). O ponto máximo desse aumento ocorre quando procuramos clusters em sub-espaços de 4 condições experimentais por 4 fatias de tempo. Entretanto, as médias dos índices diminui consideravelmente quando buscamos clusters em sub-espaços de 5 condições experimentais por 5 fatias de tempo. Isso ocorre porque o algoritmo não foi capaz de encontrar os clusters corretos em todos os atributos do conjunto de dados. Para resolver esse problema, o usuário deve executar o algoritmo da mesma forma que neste trabalho, ou seja, aumentando progressivamente o tamanho do sub-espaço até o ponto ótimo, ou próximo ao ótimo. Cabe notar também que a capacidade de expansão de clusters do algoritmo, ou seja, começar com parâmetros de sub-espaços pequenos e progressivamente expandir os pequenos clusters encontrados, não foi satisfatória. Este aspecto foi notado porque nas execuções do algoritmo com parâmetros de sub-espaço pequenos, como $min_s = 2$ e $min_t = 2$, muitos clusters sobrepostos foram criados, levando os valores da Medida-F para baixo.

- Existem soluções com qualidade heterogenea. Essa característica pode ser observada através do desvio padrão dos valores relativos aos índices de qualidade da tabela 5.4. Vamos considerar primeiramente a Medida-F, para os três conjuntos de dados com clusters Gaussianos e para o conjunto Eli_20c: conforme a Medida-F aumenta em relação junto ao tamanho dos sub-espaços, o desvio padrão diminui. Isto indica homogeneidade na qualidade das soluções encontradas. Porém, quando a Medida-F chegou a seu pico, no sub-espaço 4×4, o desvio padrão de todos os conjuntos de dados também aumentou, indicando que foram encontradas soluções muito boas, mas também outras muito ruins. Quando a média da Medida-F diminiu, no sub-espaço 5×5, o desvio padrão de todos os conjuntos de dados também diminuiu, exceto o do conjunto Gau_20c que permaceu praticamente estável.

  Agora considerando o coeficiente de Jaccard, enquanto o valor médio do índice para os conjuntos de dados com clusters Gaussianos aumentou no intervalo de sub-espaços 2×2 até 3×3 e a média do índice dos conjuntos de dados elipsóides diminuiu neste mesmo intervalo, o desvio padrão de todos os conjuntos de dados diminuiu, indicando similaridade da qualidade nas soluções encontradas. De maneira similar à Medida-F, quando a média dos valores do coeficiente de Jaccard chega a seu melhor valor para os conjuntos de dados, no sub-espaço 4×4, o desvio padrão também sobe, principalmente para os três conjuntos de dados Gaussianos, indicando uma alta heterogeneidade das soluções obtidas.

  A heterogeneidade das soluções em sub-espaços "pequenos" indica que no momento das execuções do algoritmo variando seus parâmetros, a granularidade da modificação dos parâmetros não precisa ser tão grande, isto é, pode-se utilizar "passos largos", pois os índices de qualidade não estão condensados. No outro extremo, a homogeneidade na qualidade das soluções em sub-espaços maiores indica que para procurar o melhor conjunto de parâmetros do algoritmo para um certo conjunto de dados, é preciso utilizar uma granularidade alta, ou seja, "passos pequenos". Concluindo, verificamos que conforme o tamanho do sub-espaço aproxima-se do tamanho do conjunto de dados, maior precisa ser a granularidade da variação nos valores de parâmetros do algoritmo.

Um outro aspecto que gostaríamos de relatar é a existência de soluções inválidas entre os experimentos. Consideramos inválidas as soluções que representam a execução do algoritmo com um conjunto de parâmetros cujos resultados produziram

- um dos dois índices de qualidade negativos,
- índices contendo *NaN (Not a Number)*,
- ou não encontraram nenhum cluster.

Índices negativos dizem respeito a execuções do algoritmo que levaram a resultados piores que os índices relativos à classificação aleatória (utilizada no processo de normalização). No caso dos índices contendo *NaN*, isto significa que a diferença entre o resultado obtido e a classificação aleatória foi tão pequeno que acabou desaparecendo, mesmo levando em conta a precisão trazida pelo tipo de dados *double*, utilizado neste estudo para armazenar os índices de qualidade. Na figura 5.6 podemos ver o percentual de soluções inválidas geradas nos experimentos de cada conjunto de dados em cada sub-espaço.

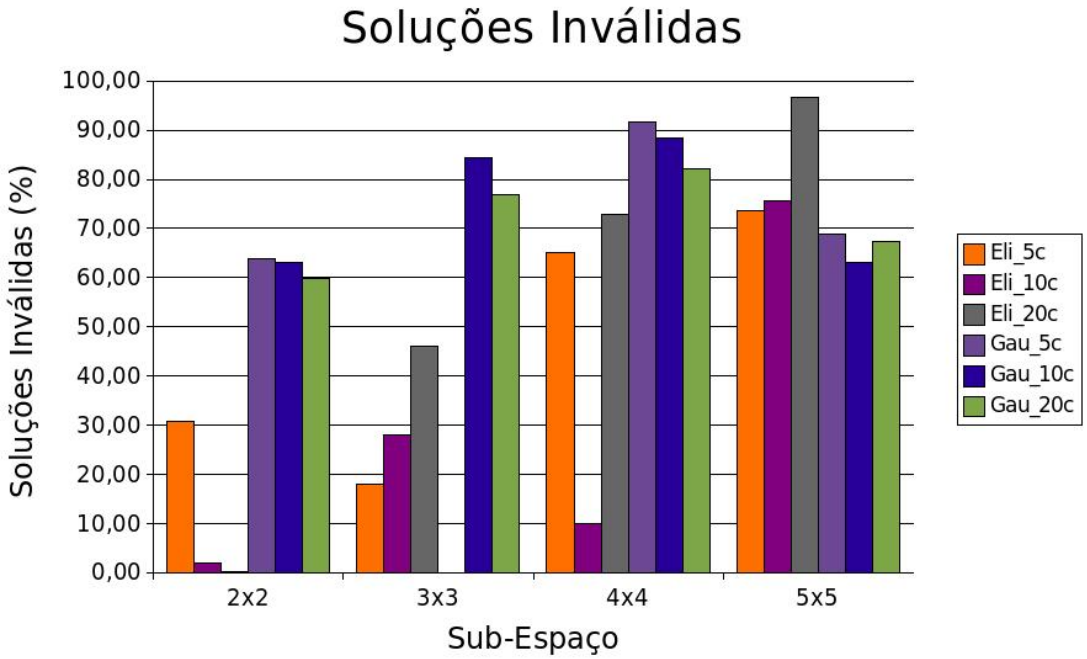

Figura 5.6: Soluções inválidas obtidas com as execuções do nosso algoritmo levando em conta todos os conjuntos de dados e todos os sub-espaços.

5.2.2 Resultados do TriCluster

Assim como na avaliação individual do nosso algoritmo desenvolvido neste projeto de mestrado, calculamos a média e o desvio padrão tanto da Medida-F quanto do coeficiente de Jaccard. Com esses valores criamos as tabelas 5.6 e 5.7. Além disso, encontramos as soluções presentes na fronteira de Pareto de todos os conjuntos de dados em todos os sub-espaço, a figura 5.7 ilustra alguns casos.

Além disso, realizamos a comparação entre as médias dos índices utilizados para todas as soluções obtidas nas execuções dos algoritmos e apenas as soluções presentes nas fronteiras de Pareto, considerando todos os conjuntos de dados e todos os sub-espaços. Podemos ver nas figuras 5.8 e 5.9 o resultado dessas comparações.

Para auxiliar o entendimento do comportamento do TriCluster em diferentes conjuntos de dados, levando em conta os diferentes sub-espaços, criamos os gráficos das figuras 5.10 e 5.11 com as médias dos dois índices de validação externa utilizados. Cabe notar que somente as soluções das fronteiras de Pareto foram utilizadas pois se levassemos em conta todos os resultados obtidos, o desvio padrão de todos os conjuntos de dados eram altos e conseqüentemente iriam distorcer as reais tendências deste algoritmo.

Assim, após avaliarmos as evidências apresentadas nas execuções do TriCluster, lembrando que a avaliação da classificação de pares é indicada pelo coeficiente de Jaccard e a qualidade da classificação individual é dada pela Medida-F, podemos inferir o seguinte a respeito deste algoritmo:

- O algoritmo não encontrou nenhum cluster nos conjuntos de dados e sub-espaços: Eli_5c no sub-espaço 5×5, Gau_5c nos sub-espaços 2×2 e 5×5 e Gau_10c no sub-espaço 2×2.

- O algoritmo realiza uma classificação de pares mais precisa no sub-espaços 2×2. Conforme o tamanho mínimo dos sub-espaços considerados aumenta (3×3, 4×4 e

Medida-F

Conj. Dados	**Eli_5c**		**Eli_10c**		**Eli_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	0.049	0.052	0.063	0.056	0.079	0.063
3×3	0.089	0.049	0.042	0.038	0.048	0.058
4×4	0.016	0.031	0.015	0.018	-0.015	0.585
5×5	–	–	0.014	0.009	-0.023	0.015
Conj. Dados	**Gau_5c**		**Gau_10c**		**Gau_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	–	–	–	–	0.002	0.008
3×3	-0.009	0.034	-0.022	0.025	0.033	0.048
4×4	-0.007	-0.002	-0.020	0.024	-0.018	0.009
5×5	–	–	-0.040	0.000	0.003	0.011

Coeficiente de Jaccard

Conj. Dados	**Gau_5c**		**Gau_10c**		**Gau_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	0.105	0.096	0.138	0.043	0.136	0.043
3×3	0.039	0.041	0.025	0.039	0.049	0.042
4×4	0.023	0.025	0.040	0.027	0.049	0.011
5×5	–	–	0.044	0.016	0.001	0.007
Conj. Dados	**Gau_5c**		**Gau_10c**		**Gau_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	–	–	–	–	-0.004	0.006
3×3	0.106	0.048	0.037	0.018	0.110	0.024
4×4	0.080	0.020	0.027	0.024	0.014	0.012
5×5	–	–	-0.015	0.001	0.006	0.012

Tabela 5.6: Tabela com os valores da média (MED) e do desvio padrão (DP) tanto do coeficiente de Jaccard como da combinação da medida de Pureza e da Integridade através da média harmônica (Medida-F). Vale notar que onde houver $-$ significa que o algoritmo não encontrou nenhum cluster.

Medida-F

Conj. Dados	**Gau_5c**		**Gau_10c**		**Gau_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	0.138	0.080	0.12	0.09	0.201	0.023
3×3	0.117	0.073	0.091	0.050	0.155	0.065
4×4	0.068	0.042	0.035	0.033	0.128	0.061
5×5	–	–	0.028	0.014	0.007	0.000
Conj. Dados	**Gau_5c**		**Gau_10c**		**Gau_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	–	–	–	–	0.022	0.003
3×3	0.037	0.060	0.004	0.042	0.187	0.009
4×4	0.043	0.029	0.019	0.052	0.004	0.028
5×5	–	–	−0.040	0.000	0.014	0.012

Coeficiente de Jaccard

Conj. Dados	**Gau_5c**		**Gau_10c**		**Gau_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	0.235	0.106	0.239	0.061	0.216	0.014
3×3	0.186	0.028	0.131	0.026	0.120	0.014
4×4	0.085	0.025	0.054	0.033	0.015	0.018
5×5	–	–	0.047	0.025	0.018	0.001
Conj. Dados	**Gau_5c**		**Gau_10c**		**Gau_20c**	
Sub-Espaço	**MED**	**DP**	**MED**	**DP**	**MED**	**DP**
2×2	–	–	–	–	0.014	0.003
3×3	0.165	0.066	0.065	0.023	0.141	0.025
4×4	0.095	0.027	0.038	0.027	0.008	0.016
5×5	–	–	−0.015	0.001	0.029	0.005

Tabela 5.7: Tabela com os valores da média (MED) e do desvio padrão (DP) tanto do coeficiente de Jaccard como da combinação da medida de Pureza e da Integridade através da média harmônica (Medida-F). Estes valores dizem respeito às soluções presentes na fronteira de Pareto encontradas pelo nosso algoritmo. Vale notar que onde houver − significa que o algoritmo não encontrou nenhum cluster.

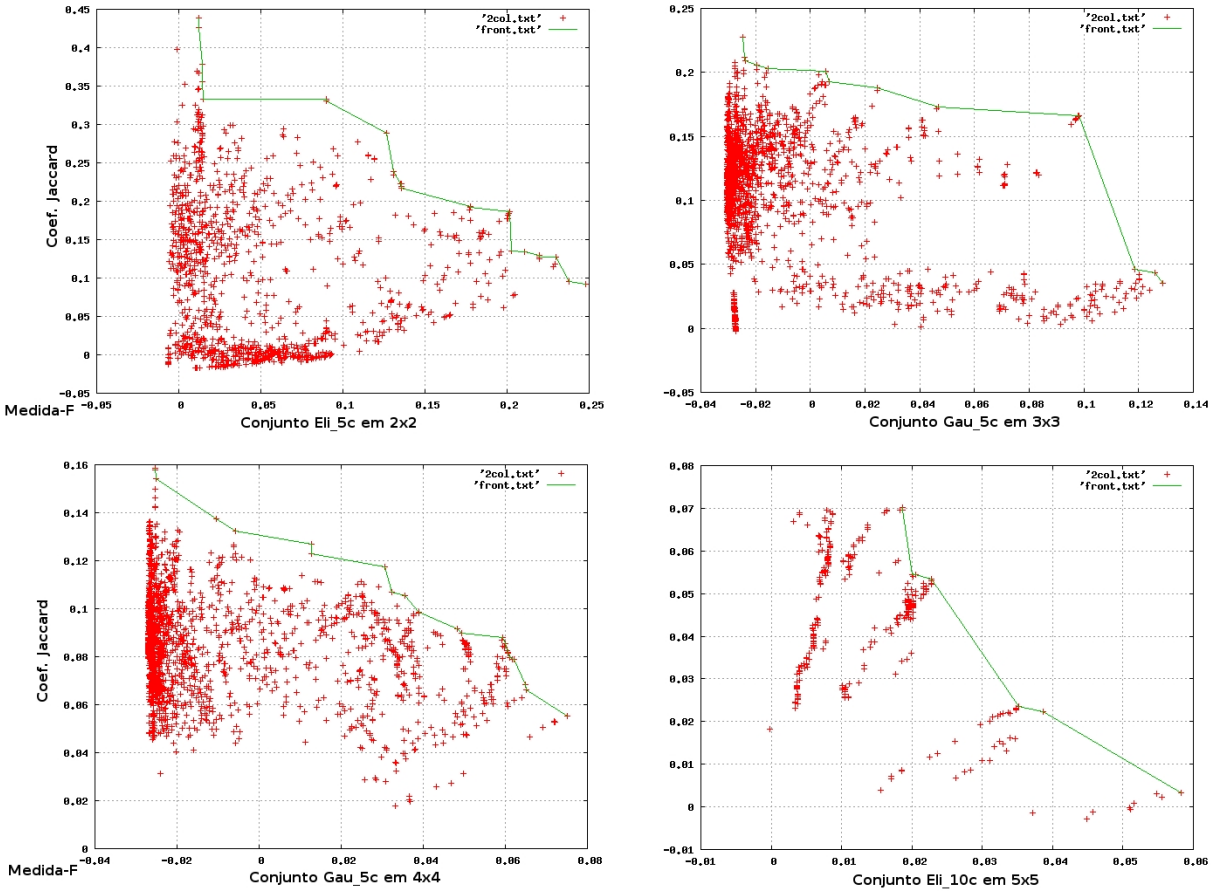

Figura 5.7: Execuções do algoritmo TriCluster com um determinado conjunto de parâmetros em alguns conjuntos de dados nos 4 sub-espaços considerados. A linha é a fronteira de Pareto, contendo as melhores soluções. Note que algumas execuções possuem índices de qualidade negativos, cujo significado será explicado posteriormente. Vale notar que não colocamos os gráficos de todos os conjuntos de dados em todos os sub-espaços por restrições de espaço.

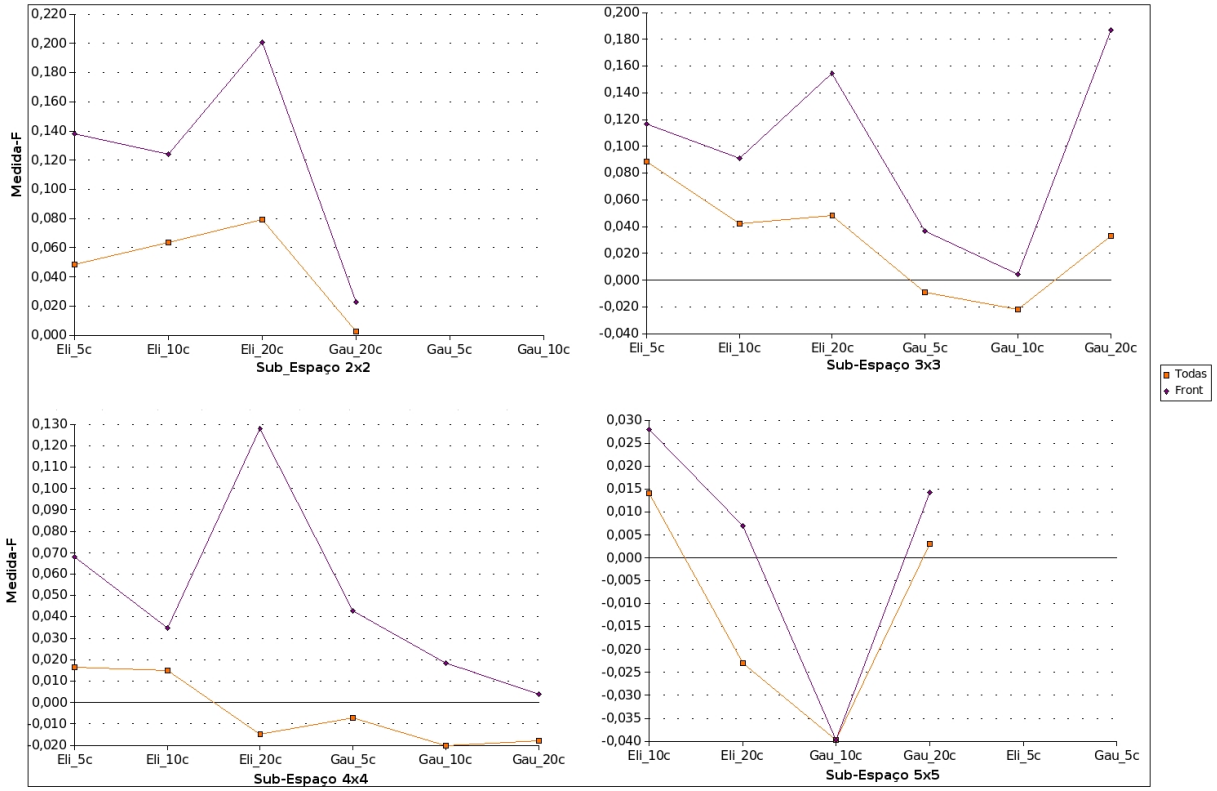

Figura 5.8: Comparação das médias da Medida-F de todas as soluções e apenas as soluções presentes na fronteira de Pareto, considerando todos os conjuntos de dados e todos os subespaços dos experimentos do TriCluster.

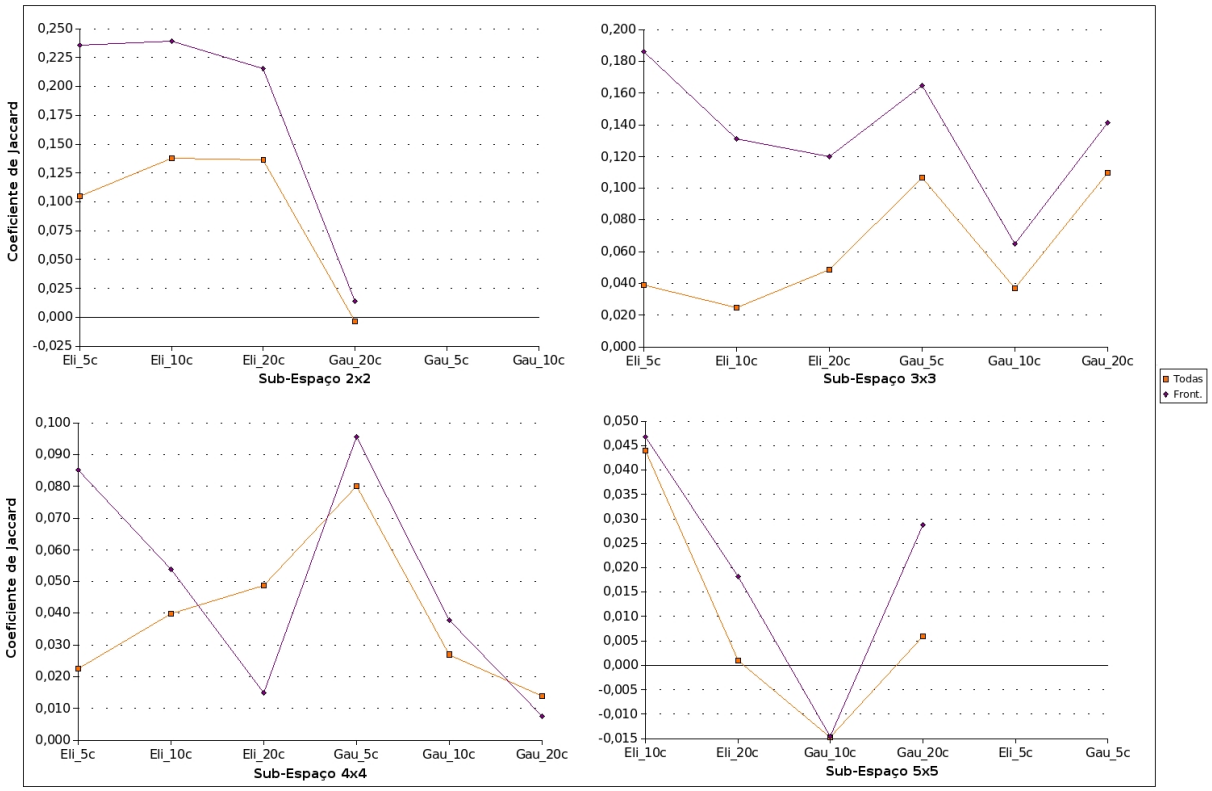

Figura 5.9: Comparação das médias do coeficiente de Jaccard de todas as soluções e apenas as soluções presentes na fronteira de Pareto, considerando todos os conjuntos de dados e todos os sub-espaços dos experimentos do TriCluster.

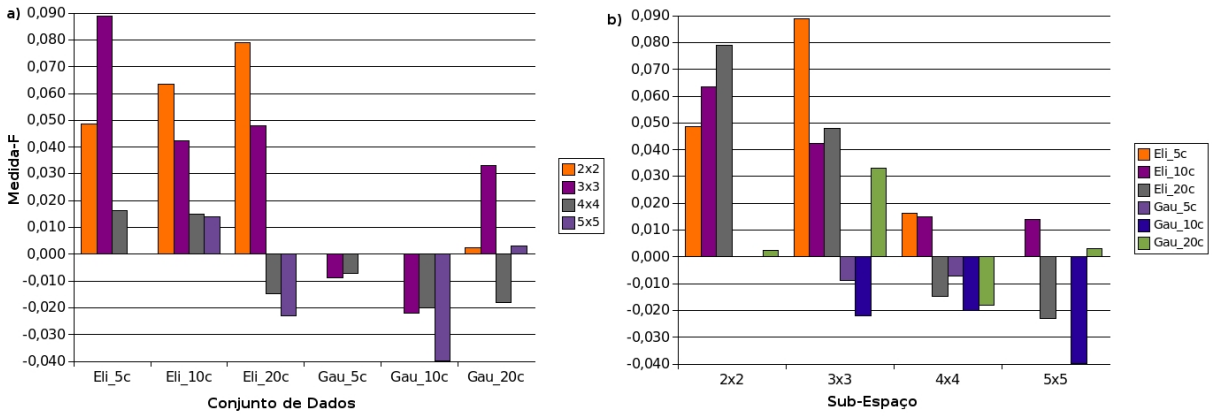

Figura 5.10: Gráficos com a média da Medida-F das soluções presentes na fronteira de Pareto, variando de acordo com (a) os diferentes conjuntos de dados e (b) os diferentes sub-espaços.

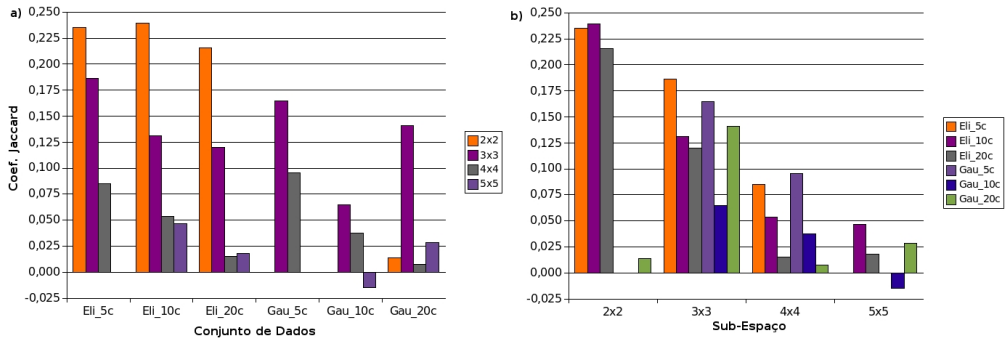

Figura 5.11: Gráficos com a média do coeficiente de Jaccard das soluções presentes na fronteira de Pareto, variando de acordo com a) os diferentes conjuntos de dados e b) os diferentes sub-espaços.

5×5) a qualidade de classificação de pares diminui consideravelmente. Entretanto, o algoritmo não encontrou clusters Gaussianos no sub-espaço 2×2 e por isso não realizou a classificação de pares.

- A classificação de pares de objetos é mais satisfatória para a maioria dos conjuntos de dados do tipo Elipsóides. Podemos ver em todos os sub-espaços da figura 5.11 a qualidade superior da classificação de pares para a maioria de conjuntos elipsóides, exceto para o sub-espaço 2×2.

- Para todos os conjuntos de dados, conforme o número de clusters aumenta, a qualidade de classificação de pares diminui, exceto para o sub-espaço 2×2.

- Considerando a figura 5.10b vemos que para o sub-espaço 2×2 nenhum conjunto de dados apresentou valores negativos em suas melhores soluções (presentes na fronteira de Pareto). Entretando, do sub-espaço 3×3 em diante o algoritmo não foi capaz de encontrar clusters em alguns conjuntos de dados e o que é pior, realizou em outros conjuntos uma classificação incorreta, o que resultou em índices de qualidade negativos indicativos de resultados piores que classificações aleatórias.

  Esse fenômeno é mais intenso em conjuntos de dados Gaussianos, cujos índices começam a ficar negativos a partir do sub-espaço 3×3, mas também se verifica nos conjuntos de dados elipsóides conforme o tamanho do sub-espaço considerado aumenta. Assim, percebemos também que esse algoritmo não é apropriado para buscar clusters considerando todas as dimensões do conjunto de dados de uma só vez.

- A variação de parâmetros realizada resultou em uma enorme quantidade de soluções inválidas e bastante heterogêneas, como pode ser verificado nas tabelas 5.6 e 5.7

Avaliamos também o percentual de soluções inválidas geradas pelo TriCluster. Na figura 5.12 apresentamos para cada conjunto de dados e cada sub-espaço o percentual de soluções inválidas fornecidas pelo algoritmo.

5.3 Comparação Entre as Soluções

Nas seções 5.2.1 e 5.2.2 apresentamos uma análise individual tanto do nosso algoritmo quanto do TriCLuster. Nesta seção apresentaremos uma comparação entre diversos as-

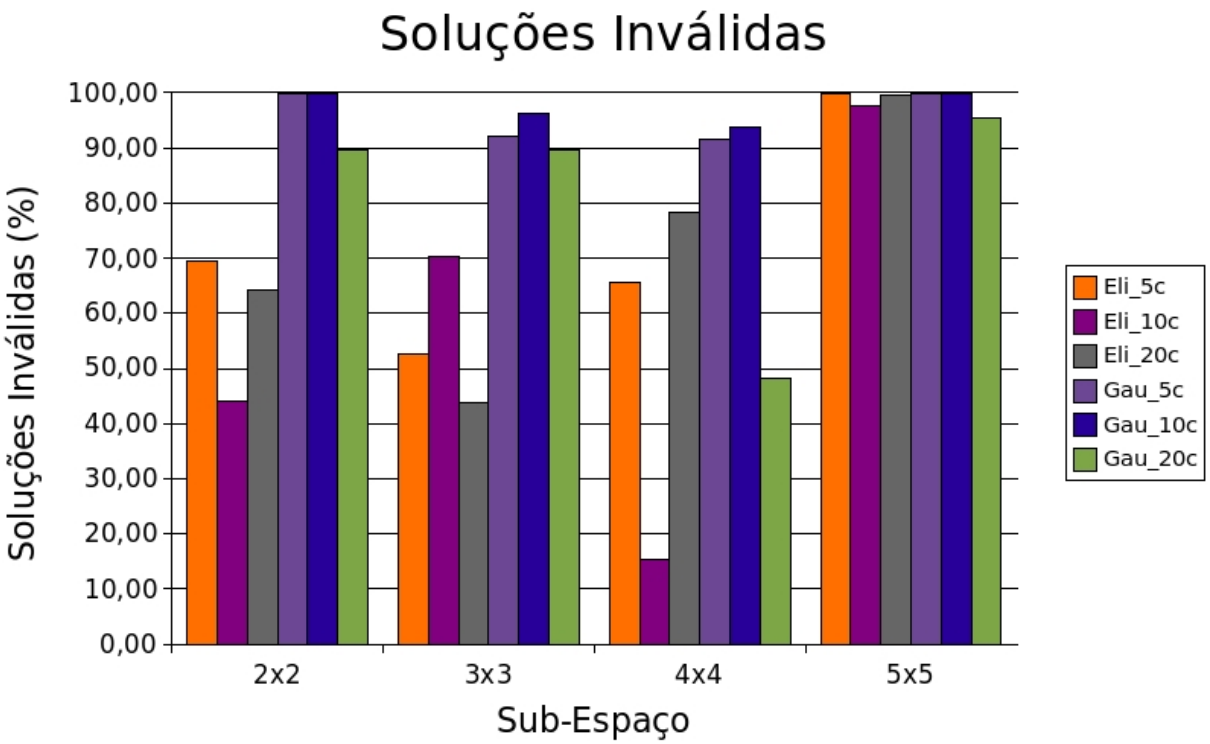

Figura 5.12: Percentual de soluções inválidas para cada conjunto de dados e cada sub-espaço utilizado nas execuções do TriCluster.

pectos dos dois algoritmos, utilizando tanto aspectos da validação externa quanto interna.

Primeiro iremos avaliar a média do coeficiente de Jaccard e da Medida-F de todas as soluções produzidas pelos dois algoritmos para avaliar em um contexto geral qual deles se saiu melhor nos conjuntos de dados. A seguir, avaliaremos comparativamente a qualidade das soluções presentes nas fronteiras de Pareto dos dois algoritmos para verificar qual deles gerou as melhores soluções. Mostraremos ainda mais duas comparações relativas a heterogeneidade das soluções tanto no contexto global quando das soluções da fronteira de Pareto. Além disso, compararemos o percentual de soluções inválidas e para concluir, serão mostrados os resultados obtidos no conjunto de dados reais.

5.3.1 Comparação geral de soluções

Apesar da grande quantidade de soluções ruins presentes nos resultados produzidos pelos dois algoritmos durante suas execuções com um certo conjunto de parâmetros, é interessante compará-los para verificar, dentro de um espaço de busca limitado, qual é o algoritmo que fornece as melhores soluções. Os algoritmos foram executados de forma a alcançar um nível de granularidade similar na variação de seus parâmetros, assim, é útil verificar dentro destes limites qual deles produziu os melhores resultados e em quais circunstâncias.

Observando a figura 5.13 observamos que nosso algoritmo saiu-se melhor que o TriCLuster levando em consideração as medidas de pureza e integralidade combinadas através da Medida-F em praticamente todos os conjuntos de dados e todos os sub-espaços considerados, com exceção de três conjuntos de dados: Eli_10c no sub-espaço 2×2, Eli_5c em 3×3 e Eli_10c em 5×5. Esse resultado nos diz que o nosso algoritmo é capaz de fornecer melhores soluções em um contexto geral quando executado com o mesmo nível

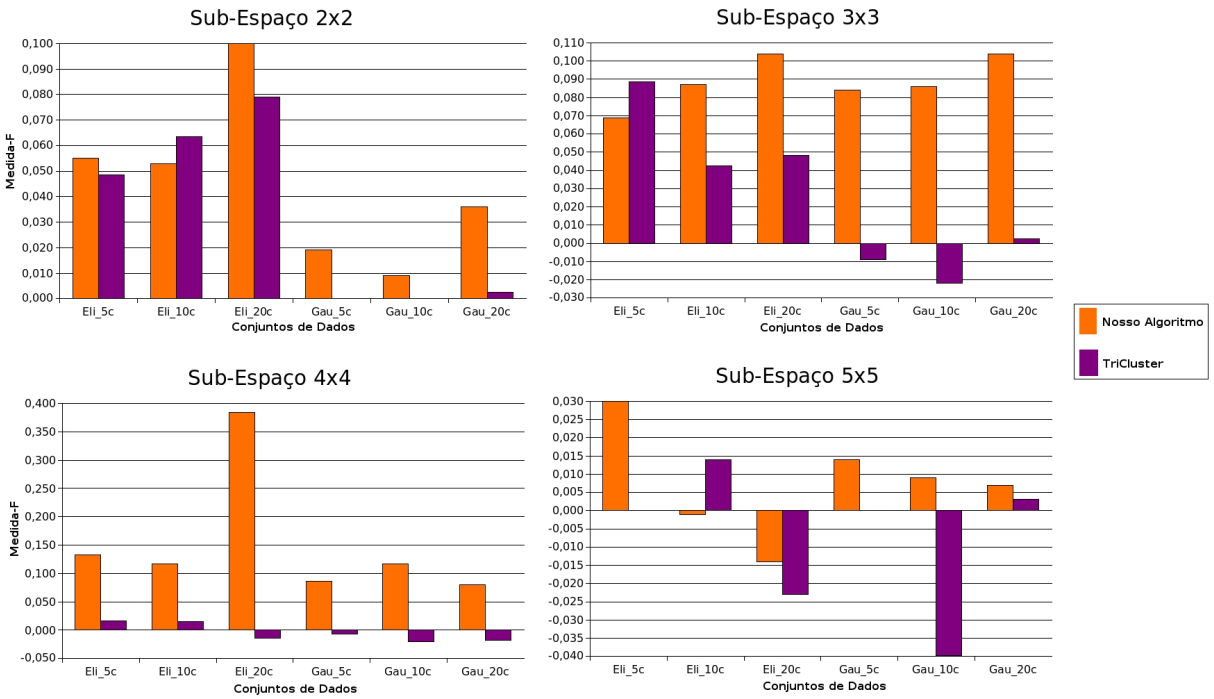

Figura 5.13: Gráfico comparativo entre as médias de Medidas-F obtidas pelos dois algoritmos em todos os conjuntos de dados e em todos os sub-espaços considerados.

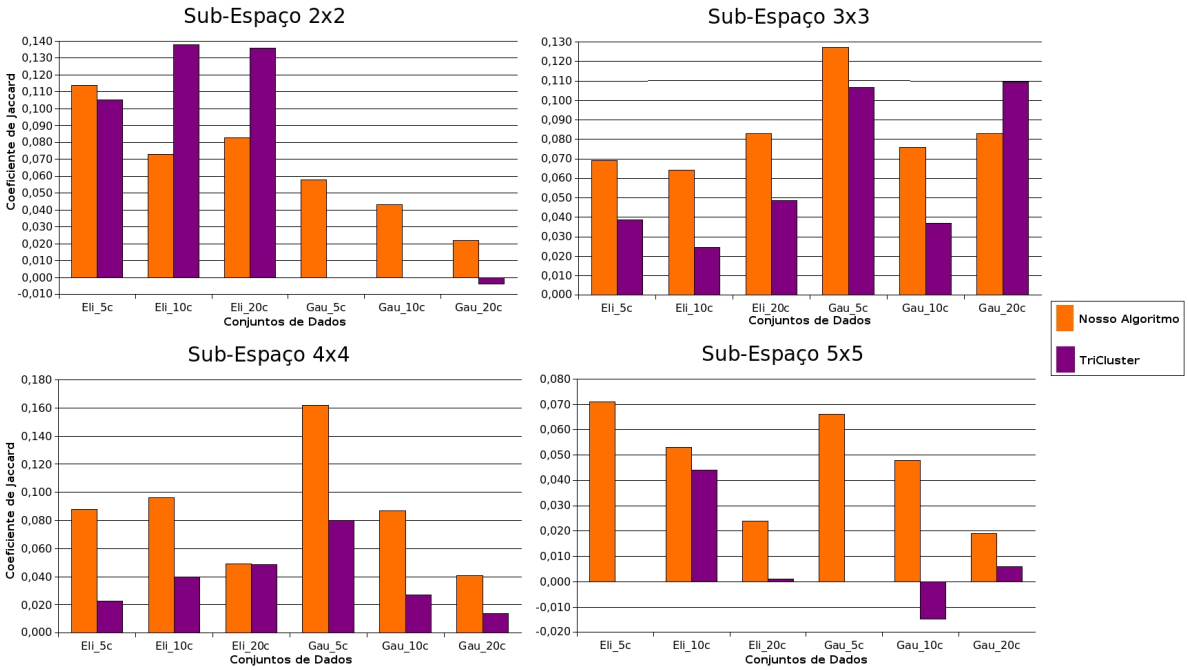

Figura 5.14: Gráfico comparativo entre as médias do coeficiente de Jaccard obtidas pelos dois algoritmos em todos os conjuntos de dados e em todos os sub-espaços considerados.

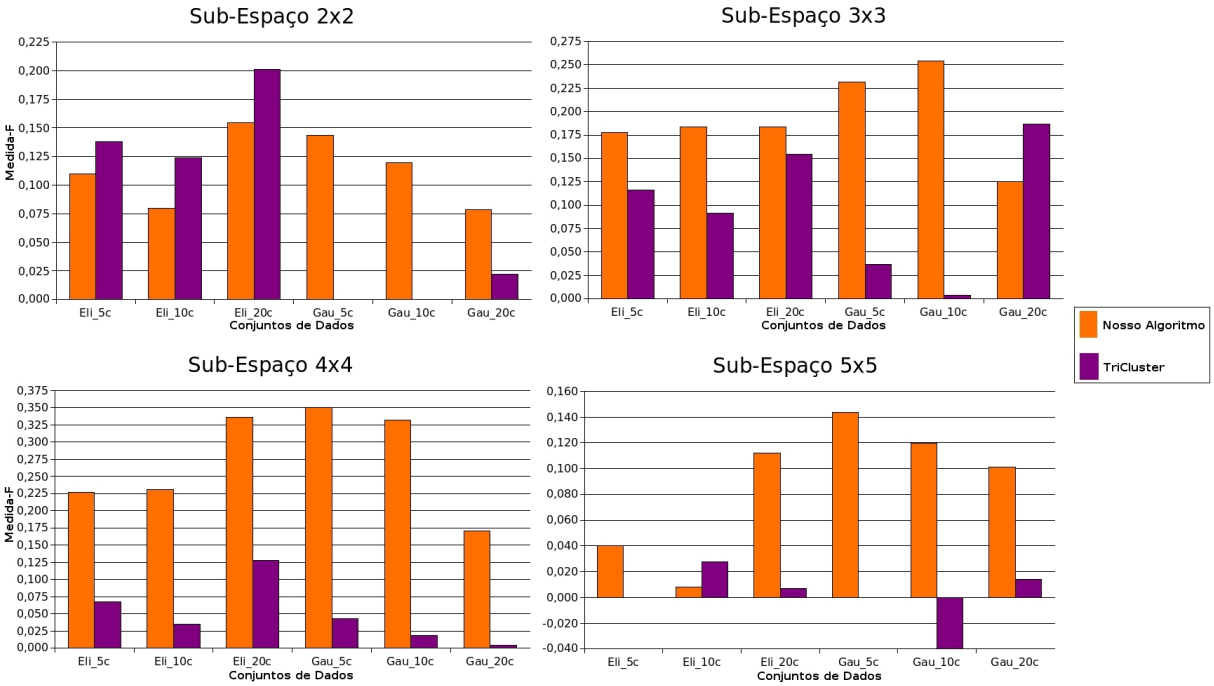

Figura 5.15: Gráfico comparativo entre as médias de Medidas-F obtidas pelos dois algoritmos em todos os conjuntos de dados e em todos os sub-espaços considerados.

de granularidade na variação de parâmetros que o TriCLuster.

Na figura 5.14 podemos ver que nosso algoritmo apresentou resultados melhores que o TriCLuster na classificação de pares de objetos. Esse comportamento manteve-se em todos os sub-espaços, com a exceção de outros três conjuntos de dados: Eli_10c e Eli_20c no sub-espaço 2×2 e Gau_20c no sub-espaço 3×3, onde o TriCluster obteve resultados melhores.

5.3.2 *Comparação das soluções da fronteiras de Pareto*

Como mencionado anteriormente, através da análise dos valores de índices presentes na fronteira de Pareto dos experimentos podemos entender quais são os pontos fortes e fracos dos algoritmos. Assim, comparar apenas os índices fornecidos pelos algoritmos para as soluções da fronteira pode nos indicar quais são os melhores resultados que os algoritmos podem fornecer dentro dos limites impostos nos experimentos para a variação de seus parâmetros.

Observando as figuras 5.15 e 5.16 percebemos que para os três conjuntos de dados elipsóides no sub-espaço 2×2 o TriCluster fornece resultados melhores que nosso algoritmo. Entretanto, conforme o tamanho do sub-espaço mínimo aumenta, a qualidade da classificação individual e de pares de objetos do nosso algoritmo aumenta gradativamente e a qualidade destas classificações do triCluster diminuem. Esse comportamento repete-se até o sub-espaço 4×4. Considerando os conjuntos de dados com clusters Gaussianos percebemos que para nosso algoritmo conforme o número de clusters aumenta, os valores da Medida-F e do coeficiente de Jaccard diminuem. No caso do TriCluster, como relatado anteriormente, ele não possui capacidade satisfatória para encontrar clusters Gaussianos, por isso seus valores da Medida-F e do coeficiente de Jaccard não seguem um comportamento padronizado e passível de análise.

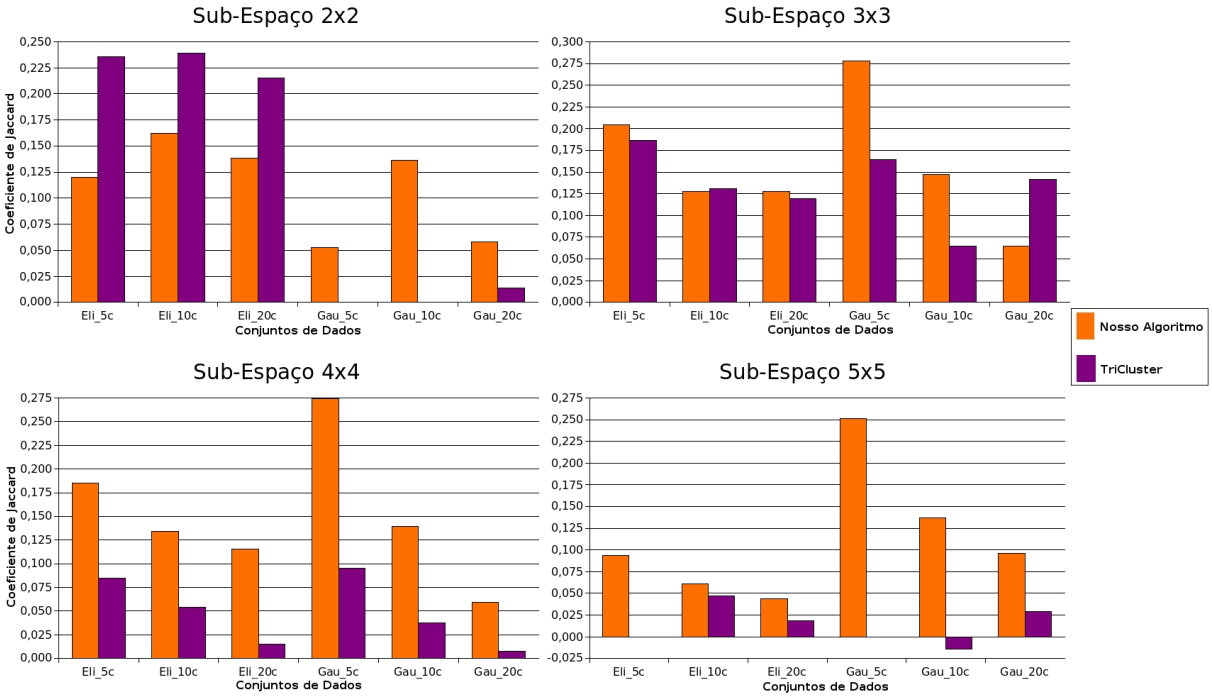

Figura 5.16: Gráfico comparativo entre as médias do coeficiente de Jaccard obtidas pelos dois algoritmos em todos os conjuntos de dados e em todos os sub-espaços considerados.

Podemos inferir através dessas observações que o triCluster possui uma capacidade de expansão de clusters satisfatória e nosso algoritmo não. Ou seja, começando de um sub-espaço pequeno (2×2) o TriCLuster é capaz de expandir os clusters Elipsóides encontrados e superar a qualidade de classificação individual e de pares de objetos do nosso algoritmo. Entretanto, quando a busca por clusters começa com um sub-espaço maior que 2×2 o TriCluster já não consegue superar os índices do algoritmo proposto nesse estudo. Por isso, para utilizar nosso algoritmo podemos iniciá-lo a partir de diferentes tamanhos de sub-espaço e obter bons resultados. Ao utilizar o TriCluster, para obter os melhores resultados devemos iniciá-lo com um sub-espaço pequeno e confiar em seu poder de expansão.

5.3.3 *Heterogeneidade de todas as soluções*

Avaliar a heterogeneidade das soluções utilizando o desvio padrão é útil para verificar o quanto a variação dos parâmetros afeta a dispersão das soluções no espaço numérico. Ou seja, quanto menor for o valor do desvio padrão de um dos dois índices de qualidade utilizados neste estudo, menor influência a variação de parâmetros tem sobre o resultado. Desta maneira, a variação dos parâmetros deve ser delicada pois se as soluções apresentam índices de qualidade muito próximos, uma mudança nos parâmetros maior do que deveria pode afastar o ponto de busca para longe das melhores soluções. Por outro lado, se o valor do desvio padrão for alto, isto significa que os índices de qualidade possuem valores distantes da média, por isso, a variação de parâmetros nos experimentos pode ser realizada em "passos maiores" em uma primeira etapa e em passos menores dentro de um intervalo que apresentou resultados satisfatórios em uma segunda etapa.

Vemos nas figuras 5.17 e 5.18 que nosso algorito possui índices de qualidade com dispersão maior que o TriCLuster, exceto em dois conjuntos de dados. Isto quer dizer que no contexto global nosso algoritmo foi capaz de explorar melhor o espaço de soluções para

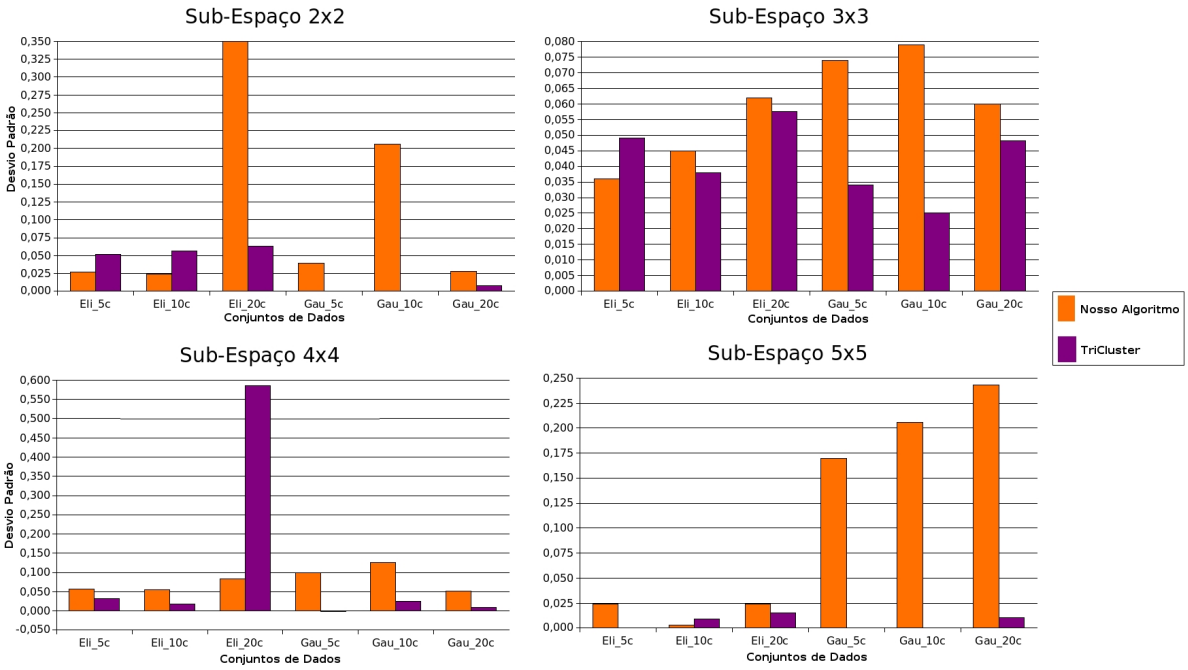

Figura 5.17: Gráfico comparativo entre o desvio padrão da Medida-F de todas as soluções fornecidas por nosso algoritmo e pelo TriCluster.

encontrar as melhores. Com esta exploração satisfatória seria possível direcionar melhor os refinamentos de parâmetros posteriores.

Além disso, podemos observar para as duas medidas de validação que, para clusters elipsóides o TriCLuster possui um grande desvio padrão mas seus valores de qualidade não são altos. Isto implica que talvez seja possível obter soluções boas para esses conjuntos de dados, mas para isso teríamos de executar o TriCluster com diversos níveis de refinamento, sempre pegando o intervalo de melhores soluções e diminuindo o passo na variação de seus parâmetros.

5.3.4 *Heterogeneidade das soluções da fronteira de Pareto*

Nas figuras 5.19 e 5.20 verificamos que as soluções encontradas pelo TriCluster pertencentes a fronteira de pareto possuem maior heterogeneidade que as soluções encontradas pelo nosso algoritmo na maior parte dos conjuntos de dados. Como mencionado anteriormente, quanto menor a dispersão das soluções, mais fácil é a tarefa de direcionar refinamentos posteriores com o objetivo de encontrar soluções melhores. Sob essa ótica, nosso algoritmo saiu-se melhor que o TriCluster, pois foi capaz de concentrar soluções melhores em um espaço reduzido.

5.3.5 *Comparação de soluções inválidas*

Na figura 5.21 observamos gráficos comparativos relativos ao percentual de soluções inválidas encontradas pelos dois algoritmos. Em praticamente todos os casos podemos verificar que o TriCluster forneceu mais soluções inválidas que nosso algoritmo, indicando que para uma grande série de parâmetros o TriCluster produz resultados inválidos. Além disso, enquanto o TriCluster teve aproveitamento nulo (100% inválido) em diversas situações, nosso algoritmo não teve nenhum caso assim. Entretanto, apesar do aproveitamento

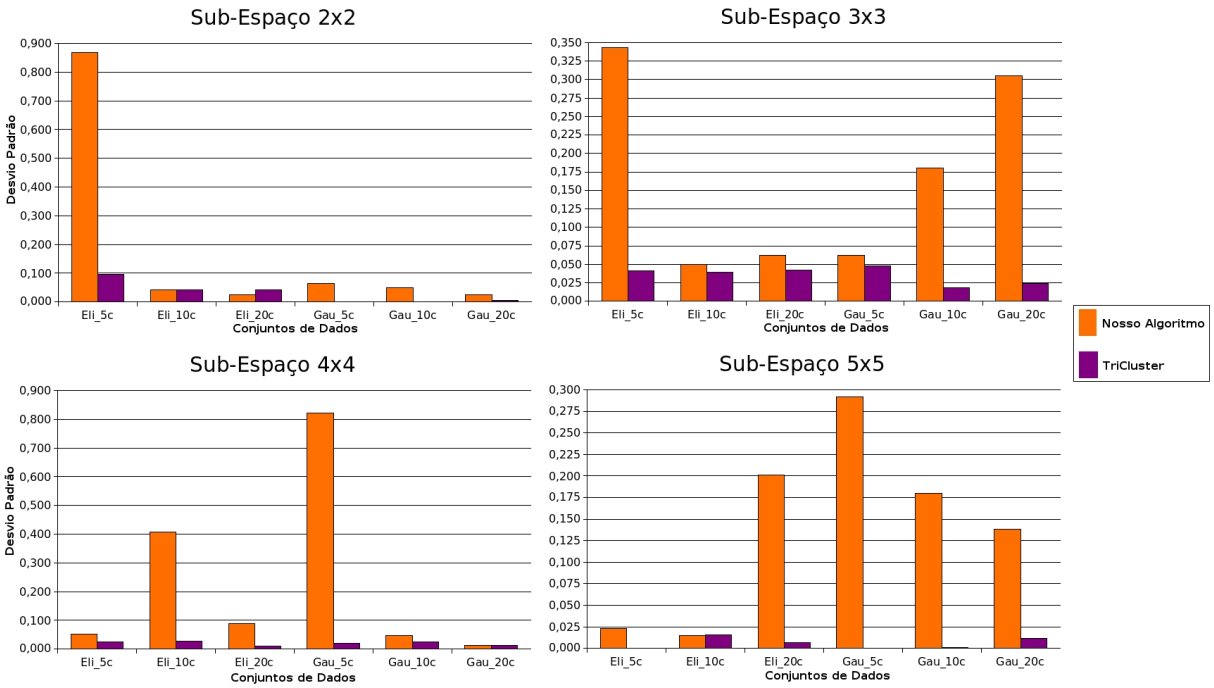

Figura 5.18: Gráfico comparativo entre o desvio padrão do coeficiente de Jaccard de todas as soluções fornecidas por nosso algoritmo e pelo TriCluster.

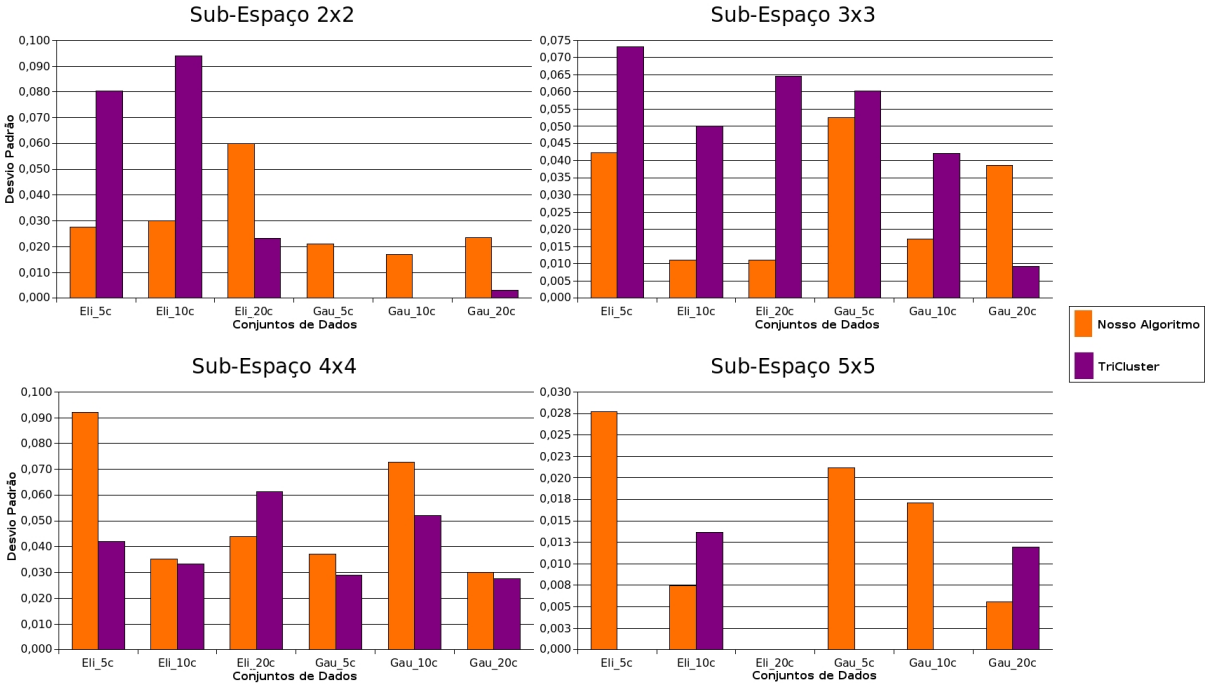

Figura 5.19: Gráfico comparativo entre o desvio padrão da Medida-F apenas das soluções presentes na fronteira de Pareto soluções fornecidas por nosso algoritmo e pelo TriCluster.

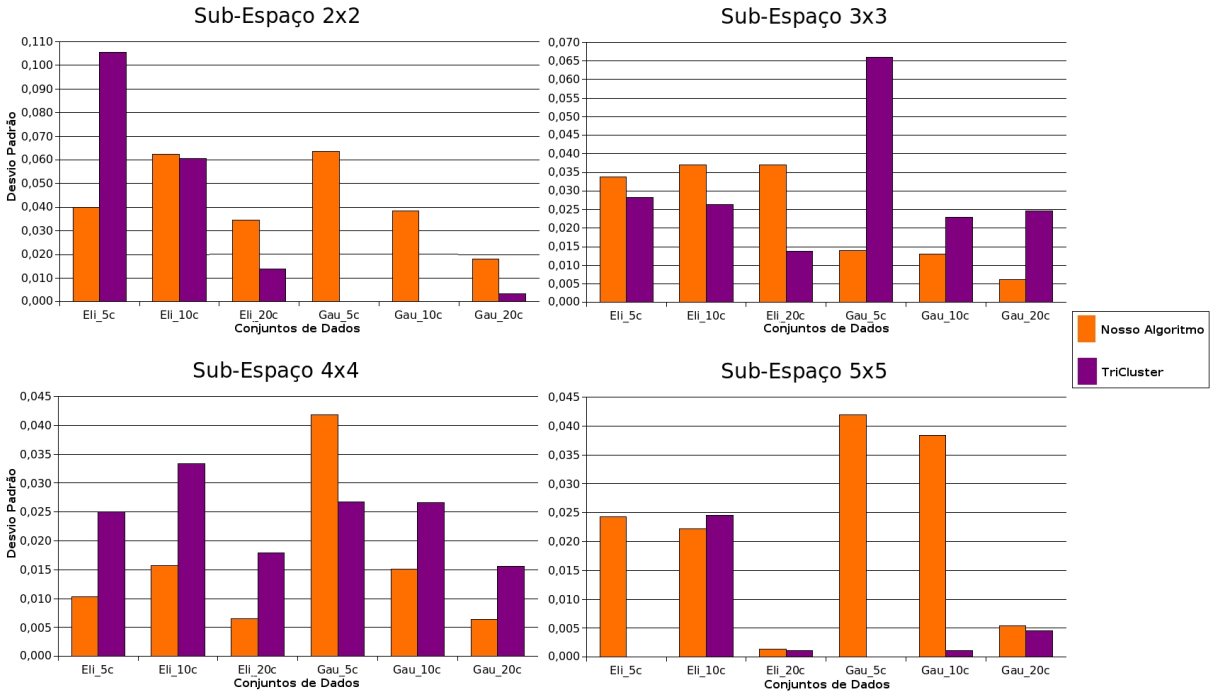

Figura 5.20: Gráfico comparativo entre o desvio padrão do coeficiente de Jaccard apenas das soluções presentes na fronteira de Pareto fornecidas por nosso algoritmo e pelo TriCluster.

do nosso algoritmo ser superior, o percentual de soluções inválidas ainda é alto para ambos, em grande parte dos casos, acima de 50% dos casos. Isto é uma fraqueza de ambos, pois, para variações em um espectro pequeno de valores, pode ser que uma boa parte dos resultados obtidos seja inválida.

5.3.6 *Execução do algoritmo com dados reais*

Os algoritmos foram executados no conjunto de dados reais descrito no capítulo 4. Variamos seus parâmetros de forma a alcançar 500 execuções cada. Calculamos para cada execução o valor de dois índices, medindo a compactação e a conectividade das soluções encontradas. Como as duas medidas devem ser minimizadas e gostaríamos de comparar apenas a melhor solução dos dois algoritmos, as combinamos através da média harmônica e ordenamos os resultados de forma crescente. O melhor resultado do nosso algoritmo resultou em 4 clusters com 13 condições experimentais cada e duas fatias de tempo. Nas nossas execuções, o melhor resultado do TriCluster foi uma solução com 48 clusters em 4 condições experimentais e 2 fatias de tempo.

Selecionamos o maior cluster do melhor resultado de nosso algoritmo e na figura 5.22 podemos ver quais genes fazem parte deste grupo e como eles se comportam de forma coerente ao longo das condições experimentais.

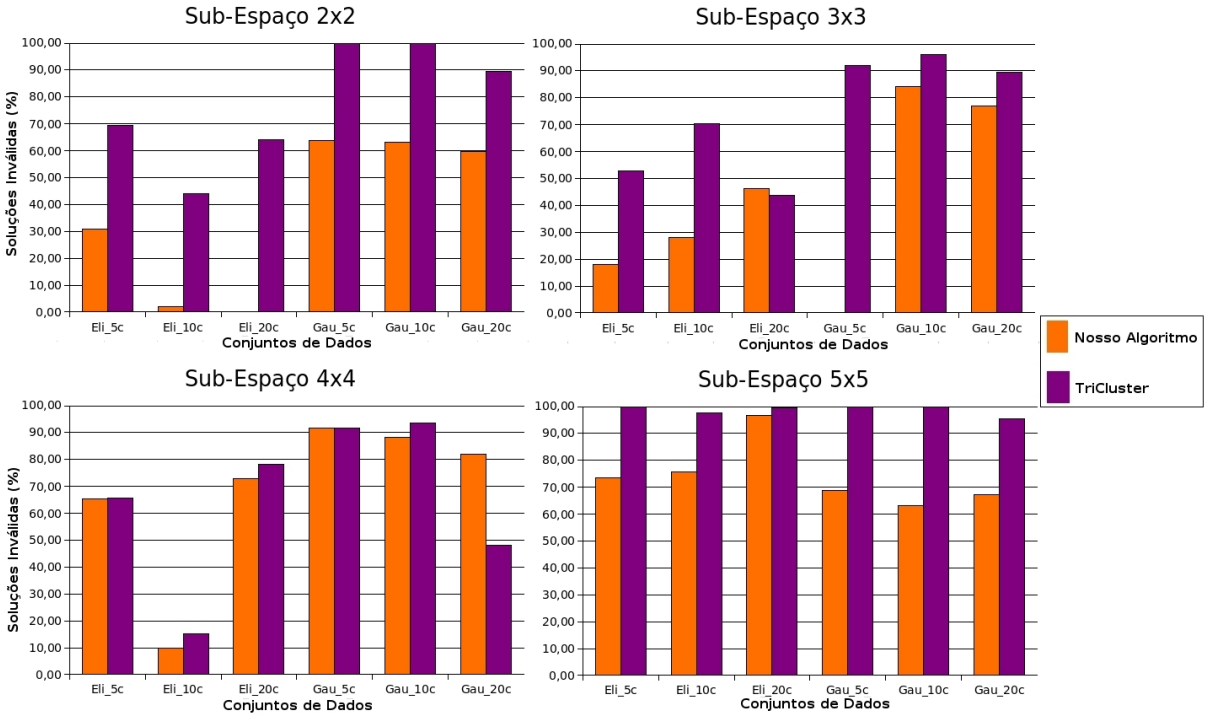

Figura 5.21: Comparação entre o percentual de soluções inválidas fornecidas pelos dois algoritmos avaliados neste estudo.

5.4 Conclusão

Após esta série de experimentos e análise de resultados concluímos que foi possível alcançar o objetivo de construir um algoritmo capaz de encontrar bons clusters em conjuntos de dados tridimensionais. Utilizando técnicas de validação recentes e aceitas na literatura fomos capazes de superar a solução existente (TriCluster) em diferentes aspectos. Entretanto é necessário lembrar que enquanto nosso algoritmo saiu-se melhor com os índices de qualidade utilizados e os conjuntos de dados propostos, pode ser que o TriCluster o supere se forem considerados outros índices de qualidade e outros tipos de conjuntos de dados.

Além disso, é muito importante lembrar que apesar das mais de 120.000 execuções dos algoritmos, não foram executados testes exaustivos e nem refinamentos dos resultados fornecidos pelos algoritmos.

O tempo de execução para o triCluster em dados reais, considerando a média de três execuções foi de 9 minutos e 8 segundos. Para nosso algoritmo, o tempo foi de 5 minutos e 44 segundos.

5.5 Trabalhos Futuros

A continuidade deste trabalho pode ser realizada na direção de extender o algoritmo para mais dimensões utilizando informações oriundas da biologia molecular, como sequências de proteínas, DNA, RNA, ou informações textuais na forma de ontologias. O volume de dados biotecnológicos produzido somente cresce, por isso são necessárias técnicas como esta, capazes de cruzar informações e fornecer resultados concisos. Além disso, pela es-

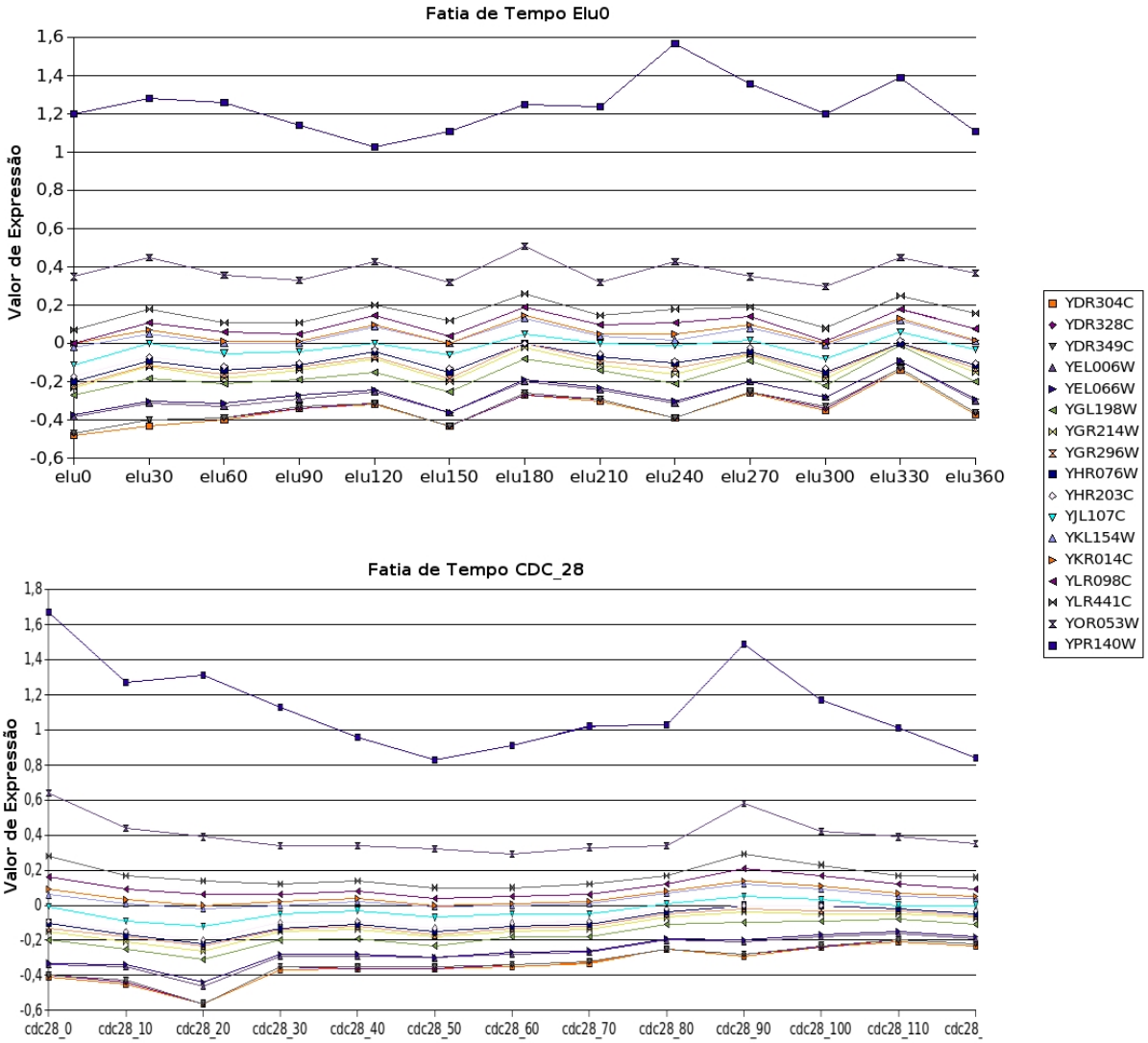

Figura 5.22: Cluster encontrado pelo nosso algoritmo no conjunto de dados reais.

trutura do nosso algoritmo, ele poderia ser reimplementado de forma paralela.

Além disso, outro aspecto desejável do algoritmo seria encontrar clusters com objetos que possuem tendências similares e não apenas valores próximos. Para isso seria necessário utilizar outras medidas de validação pois os dois índices de validação externa utilizados nesse estudo teriam pontuação ruim.

Outros conjuntos de dados de domínios diferentes (como financeiros ou geográficos) também poderiam ser testados, tanto reais quanto sintéticos, para uma maior compreensão das potencialidades do algoritmo.

Referências Bibliográficas

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. of the 1998 ACM SIGMOD Int. Conf. on Management of data*, pages 94–105. ACM Press, 1998.

[2] R.B. Altman and S. Raychaudhuri. Whole-genome expression analysis: challenges beyond clustering. *Current Opinion in Structural Biology*, 11:340–347, 2001.

[3] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University, 1961.

[4] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[5] P. Bradley and U. Fayyad. Refining initial points for k-means clustering. In *Proc. 15th Int. Conf. Machine Learning*, pages 91–99, 1998.

[6] Y. Cheng and G.M. Church. Biclustering of expression data. In *8th Int. Conf. on Intelligent systems for molecular biology*, pages 93–103, 2000.

[7] C. Ding and C. He. K-nearest neighbor consistency in data clustering: incorporating local information into global optimization. In H.M. Haddad, editor, *Proc. of 2004 ACM Symposium on Applied Computing*, pages 584–589, New York, 2004. ACM Press.

[8] M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceeding of National Academy of Science*, 95(25):14863–14868, 1998.

[9] M. Ester, H.P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of 2nd Int. Conf. on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[10] B.S. Everitt, S. Landau, and M. Leese. *Cluster Analysis*. Arnold, London, fourth edition, 2001.

[11] D.R. Gilbert, M. Schoroeder, and J.V. Helden. Interactive visualization and exploration of relationships between biological objects. *Tibtech*, 18:487–494, 2000.

[12] Z.G. Goldsmith and N. Dhanasekaran. The microevolution: Applications and impacts of microarray technology on molecular biology and medicine. *Int. Journal of Molecular Medicine*, 13:483–495, 2004.

[13] M. Halkidi. On clustering validation techniques. *J. Intell. Inform. Syst*, 17:107–145, 2001.

[14] J. Handl and J. Knowles. Exploiting the trade-off–the benefits of multiobjective in data clustering. In *Proc. of the Third Int. Conf. on Evolutionary Multicriterion Optimization*, pages 547–560, Berlin, 2005. Springe-Verlag.

[15] J. Handl, J. Knowles, and D.B. Kell. Computational cluster validation in post-genomic data analysis. *Bioinformatics*, 21(15):3201–3212, May 2005.

[16] A.K. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[17] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3), September 1999.

[18] D. Jiang, J. Pei, M. Ramanathan, C. Tang, and A. Zhang. Mining coherent gene clusters from gene-sample-time microarray data. In *Proc. of the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'04)*, Seattle, WA, USA, August 1994.

[19] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, Jan 1997.

[20] J. McQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the Fifth Berkeley Symposium on Mathematical Statistics and Probaility*, pages 281–297, 1967.

[21] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: A review. *ACM SIGKDD Explorations Newsletter*, 6(1):90–105, June 2004.

[22] G. Reis. What is generic programming? In *Library-Centric Software Design LCSD'05*, October 2005.

[23] G.C. Rota. The number of partitions of a set. *The American Mathematical Monthly*, 71(5):498–504, 1964.

[24] E. Segal, B. Taskar, A. Gasch, N. Friedman, and D. Koller. Rich probabilistic models for gene expression. *Bioinformatics*, pages 243–252, 2001.

[25] P.T. Spellman, G. Sherlock, M.Q. Zhang, V.R. Iyer, K. Anders, M.B., P.O. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell-cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. *Molecular Biology of the Cell*, pages 3273–3297, December 1998.

[26] A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18:136–144, 2002.

[27] G. Tusher, R. Tibshirani, and G. Chu. Significance analysis of microarrays applied to the ionizing radiation response. *PNAS*, 98:5116–5121, 2001.

[28] R. Xu and D. Wunsch II. Survey of clustering algorithms. *IEEE Trans. Neural Netw.*, 16(3):645–678, May 2005.

[29] J. Yang, W. Wang, H. Wang, and P. Yu. δ-clusters: capturing subspace correlation in a large data set. In *Proc.. 18th Int. Conf. on Data Engeneering*, pages 517–528, 2002.

[30] L. Zhao and M. Zaki. Tricluster: An effective algorithm for mining coherent clusters in 3D microarray data. *ACM SIGMOD 2005*, June 2005.

[31] G. Zweiger. Knowledge discovery in gene-expression-microarray data: mining the information output of the genome. *Tibtech*, 17:429–436, 1999.