FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Graph Mining para Concepção Racional de Novos Fármacos

Paulo Tiago Ferreira Seabra

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Prof. Rui Camacho (FEUP)

Co-orientador: Doutor Nuno Fonseca (CRACS/INESC Porto)

17 de Junho de 2011

Graph Mining para Concepção Racional de Novos Fármacos

Paulo Tiago Ferreira Seabra

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo júri:

Presidente: António Manuel Pereira (Doutor)
Vogal Externo: Miguel Rocha (Doutor)
Orientador: Rui Camacho (Prof. Doutor)

---

14 de Julho de 2011

Resumo

Esta dissertação tem como objectivo avaliar a ajuda que as técnicas de Graph Mining podem proporcionar no processo de concepção racional de fármacos. Em particular será avaliada a ajuda do Graph Mining na construção de modelos de previsão de toxicidade de novas moléculas tendo em conta que os testes de toxicidade são dos mais importantes em todo o processo de desenvolvimento de novos fármacos.
Foram estudados vários algoritmos de Graph Mining de forma a determinar quais os que seriam adequados para a construção de modelos de previsão de toxicidade. Os algoritmos seleccionados foram aplicados a três diferentes conjuntos de dados, fornecidos pelo projecto do Programa de Investigação de Toxicologia Computacional da Agência de Protecção Ambiental dos Estados Unidos da América. Cada conjunto de dados contém dois tipos de moléculas que foram testadas in vivo em ratos: moléculas tóxicas e moléculas não tóxicas. Uma das tarefas na dos algoritmos de Graph Mining é encontrar subestruturas comuns a conjuntos de grafos, sendo que neste problema da construção de modelos de toxicidade, as moléculas são modelizadas como grafos e pretende-se que os algoritmos encontrem subestruturas comuns às moléculas tóxicas e que não ocorram nas moléculas não tóxicas. As subestruturas frequentes encontradas podem ajudar na explicação da actividade tóxica das molécuals onde ocorrem.

Abstract

This thesis aims at using Graph Mining algorithms to improve the rational drug design process. We are particularly interested in the construction of toxicity predictive models.
Several Graph Mining algorithms were studied in a way so we could apply them to this task. The selected algorithms were applied to three different data sets from a project of the National Center for Computational Toxicology of the United States of America Environmental Protection Agency, that among other information, had data about the description of the molecules and its respective level of toxicity, results of tests made on mice. Graph Mining can be understood as the process of mining or finding substructures in information that can be represented in graphs, attending certain criteria defined by the user. In this case, the Graph Mining is applied to the search of substructures of molecules, sub-molecules so to speak, being the atoms of the molecules represented by nodes and their chemical bonds by edges, being the weight of the edge used to indicate the type of bond that the atoms share (simple, double, etc). This can be done dividing each data set in two groups, being one group constituted by toxic molecules and the other one, by non toxic molecules. Applying the algorithm we should be able to identify the substructures that have a high occurrence on the toxic molecules and a low occurrence on the non toxic molecules. These substructures may be helpful in the explanation of toxicity in the molecules they occur.

Agradecimentos

Gostaria de dar um sentido agradecimento em primeiro lugar ao Professor Rui Camacho pelo entusiasmo transmitido, esforço, ideias sugestivas e dicas durante o planeamento e desenvolvimento da dissertação.

Gostaria também de agradecer ao Doutor Nuno Fonseca por todas as sugestões e apoio prestado.

Gostaria por fim de agradecer a todo o restante pessoal da Faculdade de Engenharia da Universidade do Porto em geral, tanto Professores como alunos e colegas, que ao longo destes anos ajudaram-me na aquisição de atributos e competências que me permitiram a conclusão desta tese, e especialmente para alguns que de algum modo concluíram para o seu desenvolvimento.

Paulo Tiago Ferreira Seabra

Conteúdo

1 Introdução **1**
- 1.1 Contexto/Enquadramento . . . 1
- 1.2 Motivação e Objectivos . . . 3

2 Revisão Bibliográfica **5**
- 2.1 Concepção Racional de Fármacos . . . 5
  - 2.1.1 ADMET . . . 6
- 2.2 Graph Mining . . . 8
  - 2.2.1 Padrões . . . 9
  - 2.2.2 Algoritmos . . . 20
  - 2.2.3 Aplicações . . . 36

3 Graph Mining em estudos de Toxicidade **39**
- 3.1 Descrição . . . 39
- 3.2 MOSS . . . 40
  - 3.2.1 Procedimento . . . 41

4 Avaliação Experimental **47**
- 4.1 Dados . . . 47
  - 4.1.1 CPDBAS . . . 48
  - 4.1.2 DBPCAN . . . 48
  - 4.1.3 NCTRER . . . 49
- 4.2 Experiências . . . 50
- 4.3 Medidas de Avaliação . . . 53
- 4.4 Resultados . . . 55
- 4.5 Qualidade dos resultados . . . 57

5 Conclusões **61**
- 5.1 Satisfação dos Objectivos . . . 61

5.2 Trabalho Futuro 62

Referências Bibliográficas **63**

Anexos **69**

5.3 Resultados da Validação Cruzada 70
5.4 Descrições irregulares 72
5.5 RecolherDados 73
5.6 JuntarDados 76
5.7 DelRep 81
5.8 Executar 83
5.9 Read 89
5.10 processaDados 101
5.11 Executar2 105
5.12 Processa 106
5.13 Convertor 113

Lista de Figuras

2.1 Típico Power Law 11
2.2 Coeficiente Local. 14
2.3 Exemplo de um Núcleo Bipartido 17
2.4 Árvore de pesquisa da ciclina. 21
2.5 Relação da informação embutida com o tamanho dos fragmentos em ligações 25
2.6 Moléculas ciclina, cisteína e serina e respectiva árvore de pesquisa 32
2.7 Comparação de tempo gasto por fase do MoFa, gSpan, FSSM e Gaston 36

3.1 Exemplo da codificação SMILE e SLN 43

LISTA DE FIGURAS

Lista de Tabelas

4.1 Caracterização quanto à distribuição do tipo de moléculas dos conjuntos de dados utilizados em relação à toxicidade. 49
4.2 Conjuntos de dados e respectivas maiores médias entre as diferenças de ocorrências nas moléculas tóxicas e não tóxicas. 56
4.3 Exactidão - Médias das 10 melhores frequências de subestruturas nos 10 conjuntos de treino e teste. 58
4.4 Médias de Precisão e Recall das 10 melhores frequências de subestruturas nos 10 conjuntos de teste. 58

5.1 Médias das 10 melhores frequências de subestruturas nos 10 conjuntos de treino e teste do conjunto CPDBAS. . . . 71
5.2 Médias das 10 melhores frequências de subestruturas nos 10 conjuntos de treino e teste do conjunto DBPCAN. . . . 71
5.3 Médias das 10 melhores frequências de subestruturas nos 10 conjuntos de treino e teste do conjunto NCTRER. . . . 72

LISTA DE TABELAS

Capítulo 1

Introdução

Este primeiro capítulo serve para apresentar o enquadramento, motivação e objectivos do trabalho, identificando os problemas que esta dissertação aborda.

1.1 Contexto/Enquadramento

A informação disponibilizada publicamente está a crescer de dia para dia. Longínquos eram os tempos onde as bibliotecas e as conversas corriqueiras eram as únicas fontes de informação disponíveis. Com este aumento de informação, principalmente com a introdução e a expansão da Internet, o desejo de uma forma para a filtrar, de modo a obter apenas a informação desejada, surgiu naturalmente, e com isto o Data Mining.
Nestes últimos anos os grafos têm ganho uma grande relevância na área da representação de informação. Apesar de representarem a informação de uma forma bastante simples, através de vértices que podem ou não estar interligados entre si através de arestas, podem representar estruturas muito complexas, como é o caso da Internet, informação geográfica e até estruturas moleculares, onde na área da Química e da Bioinformática os grafos têm assumido uma particular relevância.

Com o surgimento de conjuntos de dados (datasets) sobre compostos químicos de diversas áreas, surgiu a necessidade de uma forma de ser possível analisar toda esta informação de uma forma minimamente eficiente e interactiva. Logo, os grafos foram usados para representar compostos químicos dado serem esquemas de representação adequados para codificar dados e informação deste tipo de estrutura.
Estes compostos estão divididos em subestruturas que são responsáveis pela sua actividade química.
Estas estruturas tipicamente formam padrões, onde haverão subestruturas que tendem a surgir naturalmente em outros compostos, podendo ser detectados usando algoritmos de detecção de padrões e podendo ser posteriormente classificados mediante um dado fim.
Um problema que surge frequentemente na Bioquímica diz respeito à descoberta de partes comuns entre moléculas. O que permite, por exemplo, que no desenvolvimento de novos medicamentos seja possível prever os seus efeitos antes de estes serem fabricados. Outro exemplo seria através da verificação da ocorrência de fragmentos de um grupo de moléculas, que não ocorrem em outro grupo, o que pode explicar reacções que estes compostos tenham (ou não) mostrado, em relação ao comportamento do outro grupo.
Uma das formas de resolver este problema é recorrer aos grafos como forma de representação das moléculas, onde cada átomo é representado por um nó e as ligações entres estes são representadas pelas arestas, e então usar Graph Mining para procurar subgrafos frequentes nas moléculas das bases de dados, que também são disponibilizadas para diferentes estudos.

1.2 Motivação e Objectivos

Drug Design, ou concepção de novos fármacos é o trabalho inventivo de estudar, desenvolver e criar novos medicamentos.
Estes medicamentos são normalmente moléculas orgânicas que a partir de reacções químicas activam ou inibem biomoléculas como as proteínas, resultando em benefício terapêutico para o paciente.
Este tipo de desenvolvimentos pode arrastar-se durante anos, envolvendo diversas fases e testes que requerem especialistas de diferentes áreas da química e farmácia, o que conduz a custos bastante elevados.

A abordagem adoptada neste estudo é apenas direccionada a uma fase de testes, no desenvolvimento de novo fármacos, relativa à toxicidade e consiste essencialmente em usar métodos computacionais SAR[1], que tratam as relações entre a estrutura de um composto e a sua actividade biológica, para melhor analisar o problema da toxicidade entre as moléculas através de Graph Mining.
Neste caso a partir de um conjunto de dados relativos a diferentes tipo de análises em relação à toxicidade será usado Graph Mining para pesquisar a informação disponibilizada nos conjuntos de dados, permitindo comparações, estudos e análises de uma forma muito mais fácil e intuitiva. Por exemplo, se o um composto é tóxico podemos procurar subestruturas moleculares que ocorram com frequência em compostos tóxicos e que sejam infrequentes ou mesmo ausentes em compostos não tóxicos. Estas subestruturas podem ajudar os especialistas a compreender o que é que torna um composto tóxico.

Nos capítulos seguintes são descritos os procedimentos tomados assim

[1] Structure-Activity Relationship

como os dados e ferramentas utilizados ao longo do desenvolvimento.
Por fim são expostas as experiências feitas e os resultados alcançados, assim como as conclusões retiradas destes.

Capítulo 2

Revisão Bibliográfica

Neste capítulo é descrito o estado da arte em relação ao Graph Mining, assim como uma breve descrição do enquadramento da Concepção Racional de novos Fármacos.
É também efectuada uma revisão tecnológica às principais ferramentas utilizáveis no âmbito do projecto, justificando futuras escolhas.

2.1 Concepção Racional de Fármacos

A Concepção Racional de Fármacos[1] é um processo de criar novos medicamentos ou fármacos usando métodos computacionais. Usa uma abordagem baseada na informação acerca de estruturas químicas para análise computacionalmente auxiliada, e identificação de grupos químicos candidatos a fármacos.
Desde sempre o homem usou fármacos para catalisar positivas mudança químicas no seu corpo. Inicialmente talvez a um nível inconsciente ou de um modo primitivo, mas os medicamentos sempre existiram e foram usados. Durante a história o seu desenvolvimento envolvia demasiadas

[1] Rational Drug Design

tentativas-erro, o que nem sempre corria bem. Ao longo do tempo a mudança da concepção de fármacos para métodos mais organizados e científicos tornou-se inevitável, novas técnicas passaram a ser testadas e consequentemente utilizadas. Nos dias de hoje, com o grande avanço tecnológico a que fomos sujeitos, seria inconcebível não incorporar nos métodos de desenvolvimento de fármacos este tipo de ajuda, utilizando métodos computacionais. Usando software de design molecular, capaz de dar uma melhor perspectiva ao utilizador, há um significativo corte no tempo de realização de certas tarefas que se tornam automatizadas e mais cómodas e com isto, uma redução de custos e de tempo de produção pode ser notoriamente verificada.
Na concepção racional de fármacos é inicialmente escolhido um grupo preambular de moléculas, considerado interessante para um determinado fim definido pelos especialistas, que vai passar pelas cinco fases ADMET[1], onde acabará por formar um medicamento, um grupo de moléculas orgânicas que a partir de reacções químicas activam ou inibem bio-moléculas como as proteínas, resultando em benefício terapêutico para o paciente.

2.1.1 ADMET

ADMET é o acrónimo de Absorção, Distribuição, Metabolismo, Excreção e Toxicidade que são as fases a ter em conta durante o desenvolvimento de um novo medicamento.
Como já foi referido, de modo geral estes testes são efectuados em várias fases, primeiramente consultando grandes bases de dados que apresentam enúmeras moléculas, fazendo de seguida uma triagem, onde é escolhido um conjunto potencialmente interessante. Este mesmo conjunto passa depois por diversos processos onde as suas características de:

[1] Ver secção 2.1.1

- (A)bsorção
- (D)istribução
- (M)etabolismo
- (E)xcreção
- (T)oxicidade

são avaliadas, passando em cada fase apenas um determinado número de moléculas que corresponde aos requisitos em cada caso em específico.
Na Absorção, o composto vai ser testado pela forma como é absorvido e integrado no organismo. A Distribuição analisa a forma como o fármaco se desloca no organismo e os pontos onde pode ou não chegar. O Metabolismo é o processo que avalia as reacções químicas provocadas pelo composto e as suas consequências. A Excreção é o processo que determina a forma como os produtos em excesso do metabolismo são eliminados do organismo. E por fim o teste da Toxicidade avalia o potencial tóxico do composto químico para o organismo.

Por último são realizados testes em animais, sendo que os compostos que mostrarem resultados serão posteriormente testados humanos (pessoas voluntárias) e após sanado um conjunto complexo de testes e análises, o fármaco é lançado no mercado.

2.2 Graph Mining

Um grafo é uma estrutura matemática contituída por um conjunto de nós ou vértices que podem ou não estar ligados entre si por arestas. Nestes últimos anos, os grafos têm vindo a ganhar cada vez mais importância devido às suas várias aplicações, e no modo como mantêm a eficiência na representação de informação, mesmo sendo uma estrutura básica simples. São usados na matemática, assim como em ciências da computação, onde podem representar diversos tipos de informação. As recentes redes sociais, onde os utilizadores e objectos específicos como grupos e eventos, representam nós e as suas ligações com outros objectos, arestas (por exemplo, um utilizador sendo "amigo" de outro, tem uma representação através de dois nós ligados entre si por essa ligação que é a aresta) assim como redes de computadores, onde os nós são representados pelos computadores e as ligações entre si pelas arestas, são exemplos disso mesmo. Áreas como a Bioquímica ou a Biologia também vêm utilidade neste tipo de representação de informação. Sistemas biológicos de proteínas podem ser "representados" por grafos, onde as proteínas funcionam como nós e as arestas representam as ligações com as proteínas com quem devem trabalhar em conjunto de modo a executar uma determinada tarefa biológica.
Tanto a nível cientifico como comercial, os grafos como estruturas de dados têm sido cada vez mais importantes como forma de modelar informação. A prospecção de dados aplicada à informação dos grafos tem sido muito investigada recentemente na área de Data Mining.
O Graph Mining surge com a necessidade de estudar e analisar grandes quantidades de informação que possa estar representada em forma de grafo. Consiste então em explorar grandes quantidades de dados representados por estruturas de dados que podem ser representadas por grafos, procurando padrões consistentes, que estejam de acordo com certos requisitos pré-definidos, detectando assim novos subconjuntos de dados relaciona-

dos entre si. O Graph Mining analisa os grafos detectando esses padrões e consequentemente possíveis anomalias no contexto a que o grafo se destina. Na geração de grafos sintéticos a partir de dados reais estes conceitos são de extrema importância. A partir dos padrões observados que é possível concluir a sua consistência no grafo e a sua inclusão na geração de grafos sintéticos dará um maior realismo ao sistema. Com a detecção destes padrões, as anomalias serão mais facilmente detectadas.
A geração de grafos sintéticos é importante na medida em que permite a simulação de certos dados. Por exemplo, imaginemos um grafo que representa as relações entre pessoas numa certa e determinada sociedade, a partir dos padrões encontrados seria possível gerar um grafo sintético que poderia ser manipulado de modo a simular um determinado acontecimento, como por exemplo, um terramoto, e como este teria impacto na sociedade. Os padrões têm ainda a utilidade de poder ser usados como forma de compressão de um grafo maior. Tendo em conta que os padrões representam regularidades na informação dos grafos, um grafo comprimido seria uma espécie de resumo de um grafo maior mantendo o essencial: o seu realismo em termos de informação.

Tendo sido mostrada a utilidade de identificar padrões num grafo surge o próximo problema: Como os detectar?

2.2.1 Padrões

Em Graph Mining os padrões são repetições de relacionamentos de informações que se verificam ao longo de um grafo. Por outras palavras, são as características e regras que se podem distinguir num dado grafo. Imaginemos que recolhemos uma amostra de água, o padrão principal seria a ligação de dois átomos de hidrogénio a um átomo de oxigénio.

Estes padrões apresentam inúmeras utilidades, desde a simplificação de comparações entre grafos, onde grafos com padrões similares, podem ser considerados semelhantes, até à geração de grafos sintéticos onde os padrões irão ser a base da sua construção. Dentro dos padrões de grafos há certos tipos que se verificam com uma maior regularidade podendo ser considerados padrões de padrões. A identificação destes padrões permite uma maior facilidade de análise e detecção dos mesmo aquando o estudo dos grafos. Os principais são: as Leis de potência, os Pequenos diâmetros e os Efeitos de comunidade.

2.2.1.1 Leis de potência

Num grafo normal, a distribuição dos nós e das arestas é à partida normal, ou seja de um certo modo aleatória, onde os diferentes nós têm aproximadamente o mesmo número de arestas.
No entanto, há casos onde as distribuições tendem a diferenciar-se em pontos específicos. Como no exemplo da Internet [Faloutsos 1999], verificamos que uns nós têm a tendência a apresentar um baixo valor no número de ligações, enquanto outros vértices por sua vez, apresentam um grau bastante elevado. Neste caso os computadores pessoais, serão à partida representados pelos nós que têm poucas ligações, enquanto os nós de alto grau serão routers de distribuição. O padrão Power laws, segue esta teoria de distribuição. Ou seja, a distribuição tem uma maior ênfase nas "potências", portanto nos nós com maior grau. Uma distribuição sob a lei da potência apresenta alguns parâmetros com uma elevada repetição enquanto uma maior parte tem uma frequência bastante menor. Exemplo disto é a distribuição de palavras numa dada linguagem. Por exemplo, em Português a palavra "e" é muitíssimo frequente, enquanto uma palavra como "pai" tem uma frequência bastante menor, ou mesmo a distribuição do número

de pessoas nas cidade de um país.
Em termos matemáticos uma distribuição é considerada uma distribuição Power Law quando apresenta duas variáveis de distribuição x e y, de modo a que: $y(x) = Ax^{-y}$ onde $y > 1$, sendo A uma constante positiva e y o expoente da lei da potência.
Em termos computacionais os passos que devem ser tomados para encontrar este tipo de padrões devem ser divididos em três fases [Chakrabarti e Faloutsos 2006]: Primeiramente, criar um grafo de dispersão de modo a calcular o grau de distribuição, depois deverá ser calculado o expoente da lei da potência e por fim deve ser definida a qualidade de ajuste, sou seja analisar a discrepância entre os valores reais e os valores verificados e determinar uma variável de ajuste.

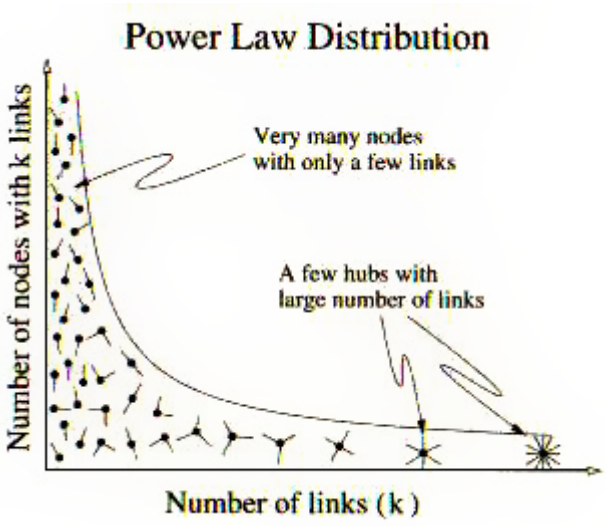

Figura 2.1: Típico Power Law [Philippe De Wilde, School of Mathematical and Computer Sciences, Heriot-Watt University].

Os padrões Power law apresentam ainda algumas variações. Entre elas destacam-se os Exponential Cutoffs e os lognormals [Pennock 2002].
Um Exponential Cutoff ocorre quando uma distribuição Power Law apresenta um rápido decréscimo entre os nós com um grau razoável e os nós de

grau elevado. Em termos matemáticos, um Power Law Exponential Cutoff é simplesmente um power law multiplicado por uma função exponencial. Exemplo desta distribuição é a rede de aeroportos, sendo os vértices os aeroportos e as arestas, as ligações entre estes, através de voos. O cut-off é notório pois um aeroporto é sempre limitado pela sua capacidade, logo chegando a um determinado número de ligações, estas vão cessar ou diminuir drasticamente.
Os lognormals são considerados exponenciais discretos Gaussianos [Bi 2001]. Por outras palavras, a distribuição das arestas é feita principalmente em determinados nós, de número reduzido, enquanto a maioria dos outros vértices têm um número de ligações bastante baixo. Basicamente é um Power law bastante acentuado. Exemplos desta distribuição são visíveis em determinados subgrafos da Internet, como páginas privadas, ex. página da Faculdade de Engenharia da Universidade do Porto, o vértice será a página principal/servidor que gere todas as ligações entre utilizadores, onde os utilizadores normalmente não partilham ligações entre si.

2.2.1.2 Pequenos diâmetros

Antes de descrever este padrão, convém elucidar o leitor sobre o que é o diâmetro de um grafo. O diâmetro de um grafo é simplesmente máximo o caminho mais curto entre dois nós do dado grafo. Ou seja, sabendo o caminho mais curto entre dois vértices para todos os vértices, o diâmetro do grafo será o caminho com maior valor entre estes.
Um padrão dos pequenos diâmetros[1] é basicamente um padrão onde o grafo estudado apresenta um baixo diâmetro. Basicamente será um grafo com nós estrategicamente ramificados, de modo a chegarem com baixo

[1] Small diameters.

custo a qualquer outro nó.
Em termos computacionais analisar este tipo de padrão é bastante custoso, pois é necessário verificar para cada nó, o caminho mais curto entre este e todos os outros nós do grafo.
O exemplo de um grafo de pequeno diâmetro seria um subgrafo de uma rede social, relativamente a uma específica lista de amigos. Através de um amigo, seria relativamente fácil chegar aos outros todos dados os amigos em comum (funcionalidade que pode ser observada no facebook por exemplo).

2.2.1.3 Estrutura de comunidades

O padrão de Estruturas da comunidades[1] é um padrão onde os nós presentes no grafo estão agrupados em diferentes grupos, ou seja, determinados conjuntos de nós estão mais perto entre si do que de outros nós. Este tipo de distribuição está geralmente presente em redes sociais [Moody 2001], onde há grupos baseados na raça, idade ou mesmo por determinados gostos e interesses.
Para medir o grau de coesão destes grupos é usado um coeficiente de clustering. Por outras palavras, um coeficiente de clustering é um valor que indica em que medida um nó se integra num dado grupo. O coeficiente de clustering local é medido em um vértice, contando os vértices que estão ligados a este, os seus vizinhos, que dividirão o número de arestas existentes entre eles. Deste modo estaremos a analisar o grau de ligação entre eles e o modo de como se enquadram em termos de contexto com o vértice em questão. Sendo Kj o número de vizinhos e Nj o número de ligações entre estes, temos

[1] Community Structure.

$C_i = \frac{N_j}{K_j}$, sendo $C_i = 0$ se $K_j = 0$.

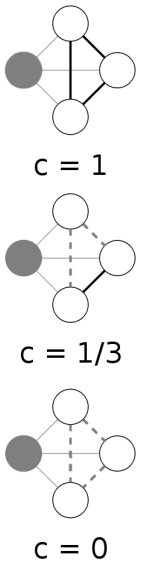

Figura 2.2: Coeficiente Local.

Para calcular o coeficiente de clustering num grafo em geral, são usadas geralmente duas fórmulas.
A primeira definição sugere que o clustering global é baseado em triplos de vértices. Um triplo pode ser formado por três nós onde dois estão ligados a um mesmo nó (triplo aberto), ou por três nós todos ligados entre si (triplo fechado). O coeficiente global é calculado dividindo o número total de triângulos vezes três, que são os triplos fechados, pelo número total de triplos no grafo (abertos ou fechados).

$$C = \frac{\text{3 x número de triplos fechados}}{\text{número do total de triplos no grafo}}$$

A outra definição [Watts e Strogatz 1998] calcula o coeficiente de clustering global, calculando a média de todos os coeficientes locais, ou seja de cada nó, de um grafo.

$$C = \sum_{i=1}^{N} \frac{C_i}{N}.$$

Em ambos os casos este tipo de cálculos em grandes grafos envolve grandes consumos em termos computacionais.
Há uma grande variedade de métodos utilizados para extrair comunidades de grafos, o que é relativamente normal tendo em conta os diversos campos da informação que utilizam o modelo de grafos para organizar a informação.

2.2.1.3.1 Dendogramas Os dendogramas representam um tipo específico de diagrama, em forma de árvores, exibindo uma hierarquia. Os nós são agrupados em hierarquias estando uns acima dos outros. Esta distribuição é feita, inicialmente atribuindo um valor, a cada par de nós do grafo, dependente do peso, e a distância entre nós (o número mínimo de vértices necessário para para uni-los através de uma ligação) ou o número de caminho independentes de nós entre os dois nós (dois caminhos são independentes de nós se os únicos nós que partilham são as pontas). De seguida, são tidos em conta todos os nós do grafo e adicionadas arestas, uma a uma de forma decrescente de valor. Todas as componentes conectadas são consideradas comunidades e cada iteração um conjunto de comunidades. No fim teremos os nós do grafo como folhas e as comunidades estarão representadas nos nós internos da árvore.

2.2.1.3.2 Stress O método seguinte é chamado stress[1], começa-se por retirar arestas do grafo em cada iteração dependentemente do seu valor de intermediação de arestas, ou seja, dependo do número de caminhos mais curtos, entre todos os nós do grafo, a que esta aresta pertence. A ideia é

[1] Conhecido como Edge betweenness ou stress

que as arestas que ligam as comunidades possuem um valor alto de intermediação de arestas.

2.2.1.3.3 Corte-mínimo Fluxo-máximo A formulação do corte-mínimo fluxo-máximo define uma comunidade como um conjunto de nós que possuem mais arestas intra-comunidade do que arestas inter-comunidade [Flake 2000]. A extracção de comunidades num grafo é efectuada com o corte mínimo. Escolhendo um conjunto de nós raiz que é sabido que pertencem a uma comunidade, é encontrado o conjunto mínimo de arestas que desconectam o grafo de modo a que todos os nós raiz formem um próprio grafo conexo. Este processo é aplicado recursivamente a este novo grafo conexo, sendo calculados em cada a iteração novos nós raiz, através do vector próprio da matriz de adjacências do grafo, até ser encontrado um bom grafo, que será a comunidade corresponde aos nós raiz.

2.2.1.3.4 Partição de Grafos Outra técnica muito popular para clustering de grafos é a Partição de Grafos[2]. Esta técnica divide inicialmente o grafo em duas partes, consideradas duas comunidades que depois deverão ser divididas entre elas. Há duas medidas que geralmente são mais utilizadas para criar as partições dentro das divisões. Uma tenta encontrar a melhor maneira de dividir o grafo, minimizando o número de arestas a cortar de modo a separar o grafo em dois subgrafos de tamanho aproximadamente igual. Esta medida é conhecida por METIS [Karypis e Kumar 1998]. A outra medida divide o grafo de acordo com as variáveis de coverage (a divisão das arestas dentro da comunidade pelo número total de arestas) e conductance (proporção entre o número de arestas dentro da comunidade e uma função baseada em pesos e nos tamanhos das divisões) [Brandes 2003]. Esta técnica é considerada bastante lenta e também apresenta o

[2]Graph partitioning

inconveniente de não possibilitar ao utilizador as definições sobre as comunidades.

2.2.1.3.5 Núcleos Bipartidos A técnica dos núcleos bipartidos, é uma técnica que inicialmente usa uma query elaborada pelo utilizador de acordo com as preferências deste, para encontrar um determinado top número de nós que preenche os requisitos da query. Depois são encontrados todos os nós que apontam ou são apontados (em caso de digrafos) para estes vértices, o que forma um subgrafo que representa uma comunidade de vértices que é relevante para o utilizador [Gibson 1998]. De seguida é aplicado o algoritmo HITS [Kleinberg 1999] a este subgrafo que encontra o top dez dos nós authority (nós que são apontados) e hub (nós que apontam para os authority), através do vector próprio da matriz A_tA, onde A é a matriz de adjacências do grafo. Esta técnica for aperfeiçoada [Kumar 1999], de modo a que não seja necessária uma query inicial do utilizador, usando núcleos bipartidos como nós raiz para encontrar comunidades. O núcleo bipartido é constituído por dois conjuntos de nós (digamos L e R), de forma a que cada nó do conjunto L aponte para todos os nós do conjunto R.

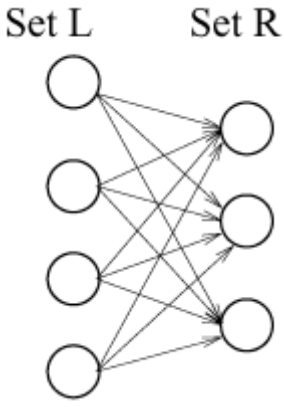

Figura 2.3: Exemplo de um Núcleo Bipartido, sendo o Grupo R o grupo inicial de nós encontrado.

2.2.1.3.6 Métodos Locais Grande parte das técnicas de determinação de comunidades envolve análises e pesquisas globais no grafo, de modo a recolher a informação necessária para determinar os clusters, o que pode levar a problemas de escalabilidade, ou seja, causar provocar dificuldades em termos de processamento e aumento de necessidade de recursos para analisar grafos de maiores dimensões.
Para combater este problema foi sugerido um algoritmo de clustering baseado apenas em informações locais [Virtanen 2003]. Para isso é definida uma métrica para qualquer grupo candidato a cluster, e usando a técnica do Arrefecimento Simulado grupos de nós vão sendo testados no modo de como se adequam a essa métrica.
Este método apresenta grandes vantagens a nível de escalabilidade mas os resultados em termos de consumo de memória e disco não são claros.

2.2.1.3.7 Associações Cruzadas Outro método com um bom grau de escalabilidade foi desenvolvido recentemente.
Este método não usa parâmetros pré-definidos para o clustering e segue o principio MDL (Minimum Description Length) [Chakrabarti 2004]. Este princípio define o grau de adequação de clustering em termos de qualidade de compressão. Por outras palavras, quanto melhor for possível definir um grupo de nós através de um clustering, melhor será a sua classificação, o que significa que perdas de informação com uma hipotética compressão serão penalizadas.
Este algoritmos segue uma fórmula linear, dependendo o seu tempo de execução apenas do número de arestas.

2.2.1.3.8 Leis de Kirchhoff Um método bastante interessante para encontrar comunidades num grafo é um método baseado nas leis de Kirchhoff,

considerando o grafo como um circuito eléctrico [Wu 2004].
Neste "circuito" as arestas apresentam todas a mesma resistência. O procedimento começa com a injecção de um Volt num dado nó e zero Volts são aplicados a um outro nó aleatoriamente calculado, que não se deverá encontrar na mesma comunidade. De seguida, usando as leis de Kirchhoff, que diz que em nó a soma das correntes eléctricas que entram é igual à soma das correntes que saem, é calculada a voltagem em todos os nós e os nós são divididos em dois grupos. Um possível processo de divisão dos nós seria, encontrar a média da voltagem e uma comunidade seria definida por ser inferior à média e outra por ser superior.
Este método apresenta a vantagem de ter um tempo de processamento relativamente reduzido, mas a qualidade de clustering fica dependente do número de iterações, onde quanto maior, mais optimizada será a solução apresentada.
Esta técnica apresenta ainda o inconveniente da selecção dos nós correctos a serem aplicados os zero volts. Os autores propõem testes aleatórios com repetições, mas os resultados destes testes ainda são incertos.

2.2.2 Algoritmos

Geralmente, as principais tarefas de Data Mining num grafo incluem pesquisar todos os seus subgrafos.

Um dos processos básicos do Graph Mining envolve então pesquisar os subgrafos de um grafo. As duas principais técnicas utilizadas para este efeito são a Geração de Candidatos, onde novos subgrafos são criados a partir de subgrafos mais pequenos, e Computação de Suporte onde a frequência dos novos subgrafos é determinada.
Em relação à Geração de Candidatos normalmente é efectuada por uma de duas formas, ou se fundem dois subgrafos que partilhem um mesmo núcleo ou se extende o subgrafo aresta a aresta. No primeiro caso para além de haver o risco de serem gerados subgrafos que nem sequer existem na base de dados, o processo de detectar núcleos em comum é um pouco dispendioso. No segundo caso apesar de não haver perdas com o processo de determinação de núcleos em comum há também a possibilidade de serem gerados subgrafos não existentes na base de dados. No entanto há uma medida que colmata esta falha, aquando a Computação de Suporte, o candidato a subgrafo é testado em todos os grafos da base de dados só validado se de facto existir. Apesar de este processo aumentar o tempo de computação necessário, esta técnica continua a ser a mais rápida no que toca a extensão de subgrafos.
A Computação de Suporte para subgrafos candidatos também pode ser feita de duas maneiras.
Da primeira forma, como já mencionado, o subgrafo pode ser testado em todos os grafos da base de dados, o que é dispendioso computacionalmente. No entanto uma melhoria a este método, faz com que o subgrafo seja testado apenas em grafos onde o seu grafo pai ocorrer. O que significa que seria necessária uma lista de grafos onde o subgrafo ocorre, o que será bastante oneroso em termos de memória caso a base de dados seja muito

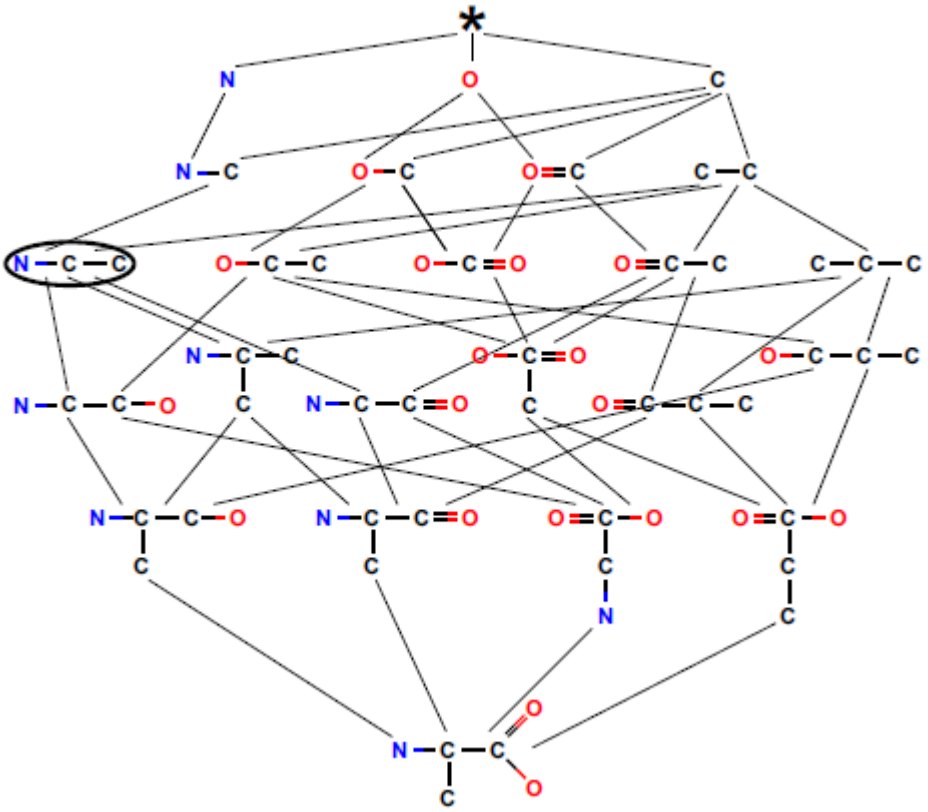

Figura 2.4: Na figura [Fischer 2004] podemos observar um grafo que representa o composto de ciclina e todos os seus subgrafos. No primeiro nível temos todos os fragmentos possíveis de apenas um átomo, no segundo, os fragmentos que têm uma ligação formando uma submolécula de dois átomos, no terceiro submoléculas de três átomos, e por aí adiante até à própria molécula constituída por seis átomos. De notar que é também exibido o tipo de ligação química, neste caso todas simples ou duplas. Também poderá ser verificado que, por exemplo, não existe nenhum fragmento no segundo nível constituído pelos átomos N e C porque este não é um subgrafo encontrado neste exemplo. O subgrafo assinalado, é o subgrafo relevante mais frequente presente na molécula.

grande.

Outro modo de fazer a computação de suporte envolve a utilização de listas embutidas. Ou seja uma lista com a informação sobre os nós e arestas do subgrafo e os nós e arestas correspondentes no grafo. Assim se um novo fragmento for adicionado ao subgrafo, a única coisa a testar será a compatibilidade do novo fragmento inserido na posição corresponde no grafo, dado que a sua posição já é sabida. Este método é bastante mais rápido, mas também bastante mais caro em termos de memória devido a todas as listas embutidas aplicadas aos subgrafos.

De notar que apesar de o foco estar a ser feito sob o Graph Mining existem outras abordagens para resolver este tipo de problemas, uma delas também bastante utilizada que utiliza os princípios da lógica indutiva e descreve um grafo por expressões lógicas.

De seguida são apresentados alguns algoritmos mais específicos de Graph Mining, que na maior parte dos casos foram desenvolvidos com vista a serem aplicados a grafos que representam moléculas, sendo considerados interessantes para este estudo.

2.2.2.1 gSpan

O gSpan (graph-based Substructure pattern mining) é um dos mais populares algoritmos de Graph Mining. Sendo um algoritmo com o código disponível publicamente, as suas expansões e versões são variadas.
O algoritmo original surgiu em 2002 por Xifeng Yan e Jiawei Han [Yan 2002] e resumidamente, tem como função pesquisar subestruturas frequentes sem geração de candidatos, tratando apenas de subgrafos conexos, de modo a economizar tempo de execução. O algoritmo mapeia cada grafo com um único código DFS (Depth-First Search) que vai depender do tempo da sua descoberta na árvore de pesquisa e organiza os grafos de acordo com uma ordem lexicográfica dependente dos códigos DFS. Depois prossegue com uma pesquisa em profundidade, sendo o primeiro algoritmo de pesquisa de subgrafos frequentes a fazê-lo, baseada na ordem anterior, de forma a encontrar subgrafos conexos frequentes de modo eficiente. Enquanto cresce um ramo da árvore de pesquisa, o gSpan está simultaneamente a verificar subgrafos frequentes, combinando ambos os processos num só, o que o torna mais rápido.
O gSpan verifica se todo o conjunto de dados pode ser colocado em memória e caso seja possível, o algoritmo pode ser aplicado directamente, caso contrário será feita uma projecção da informação em grafo [Pei 2001] e então aplicado o algoritmo.
Os testes deste algoritmo mostram que o seu desempenho, em termos de execução comparada com o algoritmo FSG, é melhor em cerca de entre 15-100 vezes [Yan 2002].

2.2.2.2 Fragment mining híbrido com MoFa e FSG

Este é um algoritmo híbrido criado para colmatar a baixa performance de algoritmos já existentes de detecção de submoléculas (subgrafos) frequentes [Meinl 2004], em termos de escalabilidade. Combina dois diferentes tipos de estratégia de pesquisa através dos algoritmos MoFa [Borgelt 2002] e FSG [Kuramochi 2001]. O algoritmo MoFa (Molecule Fragment Miner) utiliza uma pesquisa em profundidade e gera novos fragmentos de grafos estendendo fragmentos mais pequenos, mantendo listas integradas com a informação para acelerar o processo de computação, mas desta forma aumentando o consumo de memória, tornando-se num algoritmo demasiado pesado para bases de dados com mais de vinte mil moléculas. Por outro lado o FSG (Frequent Subgraph Discovery) usa uma pesquisa em largura e cria novos fragmentos juntando subgrafos que partilhem as mesmas moléculas como núcleo, tentando verificar nos subgrafos na base de dados, subgrafos com a mesma estrutura para cada novo candidato (testes de isomorfismo). Este algoritmo é relativamente rápido nos períodos iniciais de pesquisa, mas à medida que os fragmentos vão crescendo a velocidade de execução desce, verificando-se mesmo um crescimento exponencial, a partir de meio da árvore de pesquisa, em termos de consumo de memória e de tempo.

Deste modo a abordagem híbrida tem relativamente bons resultados a complementar estes dois algoritmos. Tendo em conta que o algoritmo MoFa tem como principal problema o facto de necessitar de grandes quantidades de memória para estados iniciais da pesquisa, usar o algoritmo FSG inicialmente resolveria o problema, dado que este não guarda listas integradas, testando apenas o candidato a subgrafo em termos de isomorfismo, necessitando então de muito menos memória inicialmente. Mas com o aumento dos fragmentos, os testes tornam-se cada vez mais custosos o que justifica a introdução do MoFa, pois o algoritmo MoFa a partir de uma fase onde os

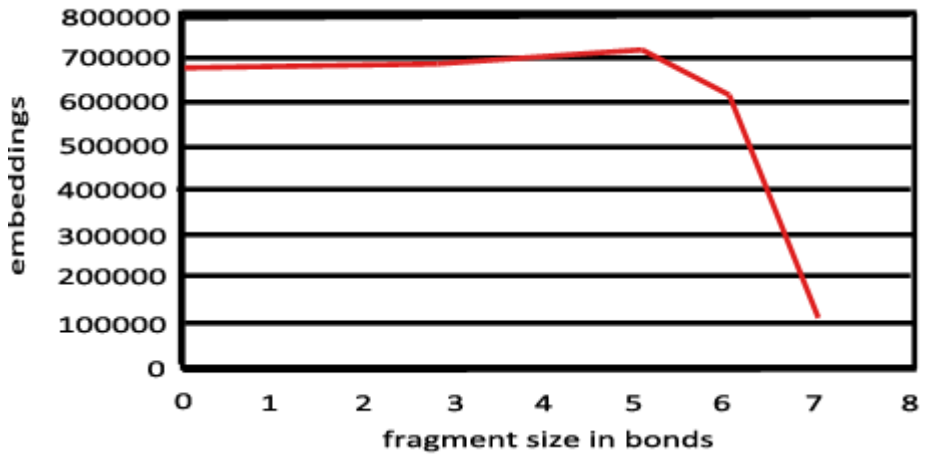

Figura 2.5: Relação da informação embutida com o tamanho dos fragmentos em ligações [Meinl 2004].

fragmentos chegam às seis ou sete ligações, o número de informação embebida diminui devido à simetria dos fragmentos. Isto é, as bases de dados moleculares tipicamente apresentam um número de anéis entre cinco ou seis átomos e tendo um subgrafo que apenas consiste num desses anéis é possível que este encontre as suas informações embebidas repetidamente em várias moléculas/fragmentos. Mas quando a pesquisa chega a uma fase onde todas estas cinco ou seis ligações se encontram atribuídas, esta informação é descartada das listas embebidas pois já não é necessária, o que diminui drasticamente o consumo de memória.

Este efeito é uma das principais motivações para esta abordagem híbrida. A ideia central é então usar inicialmente o FSG e mudar para o MoFa assim que os fragmentos descobertos atingirem um determinado critério, sendo depois os fragmentos do FSG usados como sementes para o MoFa.

2.2.2.3 FFSM

O FFSM (Fast Frequent Subgraph Mining) de Huan, Wang, e Prins [Huan 2003] é outro algoritmo de pesquisa de subgrafos frequentes. Este algoritmo usa uma pesquisa vertical e uma técnica para redução de candidatos redundantes, objectivando a eficiência dos testes.
Representa todos os grafos em matrizes e atribui a grafos isomorfos o mesmo código canónico determinado pela CAM (Canonical Adjacency Matrix).
Apresenta também como principais diferenças, uma nova forma canonica de grafo, dois processos de melhoramento de candidatos: FFSMJoin e FFSM-Extension, uma ferramenta que garante que todos os subgrafos frequentes são enumerados de forma não ambígua e não faz testes de isomorfismo, mantendo um conjunto de informações embutido em cada subgrafo. No entanto são apenas guardadas informações sobre os nós, não sobre as arestas, o que aumenta a rapidez das operações de adição e extensão pois as listas embutidas nos novos fragmentos podem ser calculadas a partir das informações dos nós.
É referido pelos autores que este algoritmo apresenta um desempenho mais eficiente que o algoritmo gSpan.

2.2.2.4 MIGDAC

MIGDAC é o acrónimo de Mining Graph DAta for Classification [Lam 2008], um algoritmo para descobrir e classificar conjuntos de padrões interessantes num dado grafo.
Este é um método de Graph Mining mais específico, aplicado a estruturas químicas, onde representa cada composto químico como um grafo e transforma-o num conjunto hierárquico de grafos usando o algoritmo MAGMA [Lam 2008], (Multi-Level Attributed Graph Mining Algorithm). Deste modo, o objectivo é poder representar mais informação que nos formatos tradicionais (como a hierarquia por exemplo) e simplificar estruturas de grafos mais complexas. O algoritmo é depois aplicado para extrair conjuntos de padrões específicos em termos de interesse. É então calculada uma medida de interesse para cada subgrafo frequente encontrado e usando uma medida de limite distinguem-se os subgrafos interessantes dos menos interessantes. Basicamente os subgrafos mais interessantes são aqueles que conseguem caracterizar uma classe única, ou seja, um grupo de subgrafos com as mesmas características. Estas classes são depois comparadas com uma amostra de um composto desconhecido, onde depois de uma comparação é feito um cálculo através de correspondência e definido um peso de evidência, que classifica a amostra como uma das classes. Seguidamente é avaliado o efeito do químico segundo o seu peso de evidência. Este algoritmo defende que para encontrar padrões úteis, é necessário não só considerar a frequência dos subgrafos como também o seu potencial de unicidade.

2.2.2.5 Gaston

O algoritmo Gaston (GrAph Sequence Tree extractiON) é outro algoritmo muito aclamado na área de Graph Mining [Nijssen 2004].
Dada uma base de dados de grafos, este algoritmo pesquisa por todas as subestruturas frequentes, permitindo a definição de parâmetros como a frequência mínima, confiança mínima, interesse mínimo e máxima frequência.
De forma simples, este algoritmo usa uma pesquisa de complexidade crescente começando por caminhos frequentes, depois árvores livres e então grafos cíclicos, de modo a manter um funcionamento eficiente.
O Gaston guarda informações para gerar apenas refinamentos dos subgrafos mais frequentes e para conseguir testes de isomorfismo mais rápidos.
Experiências em bases de dados moleculares revelam que as subestruturas mais frequentes num grafo são as árvores livres. Uma árvore livre é uma árvore que não possui um nó raiz. A principal diferença neste algoritmo é que ordena de forma eficiente os objectos (caminhos, árvores e grafos) em cada uma das suas fases. Como considera fragmentos como caminhos ou árvores inicialmente e só considera realmente grafos com ciclos no fim, grande parte do processamento pode ser feito eficientemente, tendo apenas na última fase que testar o problema do isomorfismo. O Gaston define uma ordem global dependente das arestas que fecham os ciclos e apenas gera ciclos que são maiores que o anterior.

2.2.2.6 Identificação de farmacóforos comuns usando um algoritmo de detecção de cliques frequentes

Um farmacóforo é, em termos de farmácia, a região da molécula que é activa em termos biológicos. O conhecimento dessa região possibilita a criação de novos ou melhores medicamentos. Foram desenvolvidos dois algoritmos baseados na detecção de cliques frequentes presentes nos grafos moleculares.
O primeiro algoritmos, conhecido por MCM (Multiple Conformer Miner) pesquisa todos os cliques (a definição geral de clique é, em um grafo não-orientado, é um subconjunto de seus vértices tais que cada dois vértices do subconjunto são conectados por uma aresta, ou seja, todos os vértices têm de estar ligados com todos os vértices) frequentes presentes em pelo menos uma conformação (qualquer um dos possíveis arranjos espaciais de átomos numa molécula, que possam resultar de rotações dos seus constituintes) de cada molécula, usando pesquisa em profundidade. Em cada passo, é gerado um novo candidato clique aumentando o tamanho do clique actual com a adição de um novo nó e arestas. Todas as arestas adicionadas são retiradas do conjunto de cliques frequentes gerados no início do algoritmo. O clique cresce de forma a não ser enumerado múltiplas vezes de um código atribuído a todos os cliques. Depois da geração de candidatos os cliques são enumerados e se for verificado que são frequentes, serão guardados e usados para pesquisa.

O segundo algoritmo é conhecido por UCM (Unified Conformer Miner) e explorando as similaridades entre diferentes confórmeros da mesma molécula, obtém resultados de uma forma bastante mais rápida que o primeiro. O UCM aumenta a complexidade computacional do MCM tendo em conta que usa o facto de haver similaridade estrutural entre as conformações de uma molécula, o que produz melhores resultados em tempo de execução.

Ambos os algoritmos encontram todos os farmacóforos comuns no dado conjunto de dados garantidamente, o que é confirmado pela validação do conjunto de moléculas para as quais os farmacóforos foram determinados experimentalmente.

2.2.2.7 Estratégias de corte para aumentar a velocidade dos algoritmos

Tendo em conta os recentes desenvolvimentos e actividade na área da exploração computacional de moléculas com Graph Mining, vários algoritmos foram criados para esse fim. Uma das principais características que é tida em conta aquando a avaliação destes algoritmos é a sua escalabilidade, ou seja, a sua capacidade para manter um desempenho semelhante ou linear em termos de consumo de tempo e memória quando utilizado na análise de grandes quantidades de informações.

Por este motivo foram pensadas possíveis estratégias que ajudassem um algoritmo a ser escalável e melhorassem o seu desempenho em termos de consumo de recursos.

Serão agora apresentadas duas estratégias, que têm em vista o melhoramento do algoritmo MoFa já mencionado, e que levam a um considerável aumento de velocidade de computação na pesquisa de fragmentos moleculares fechados [Borgelt 2004].

Vários algoritmos de Graph Mining utilizam listas embutidas nos fragmentos, para saberem que moléculas podem ser criadas e que ligações podem ser correctamente adicionadas. O MoFa além deste sistema utiliza também o sistema de pesquisa em profundidade como já foi mencionado.

Este algoritmo tem conhecidos problemas de consumo de memória em certas circunstâncias, o que leva à necessidade de métodos de corte. Estes métodos basicamente evitam o processamento de informações não relevantes, deste modo aliviando a memória utilizada. Estas técnicas de corte podem ser divididas em três principais categorias: cortes baseados em tamanho, cortes baseados em base e cortes estruturais que são considerados os mais importantes. De forma simples, estes últimos cortes baseiam-se num conjunto de regras que definem uma ordem local de extensões de um fragmento, o que evita certas pesquisas redundantes.

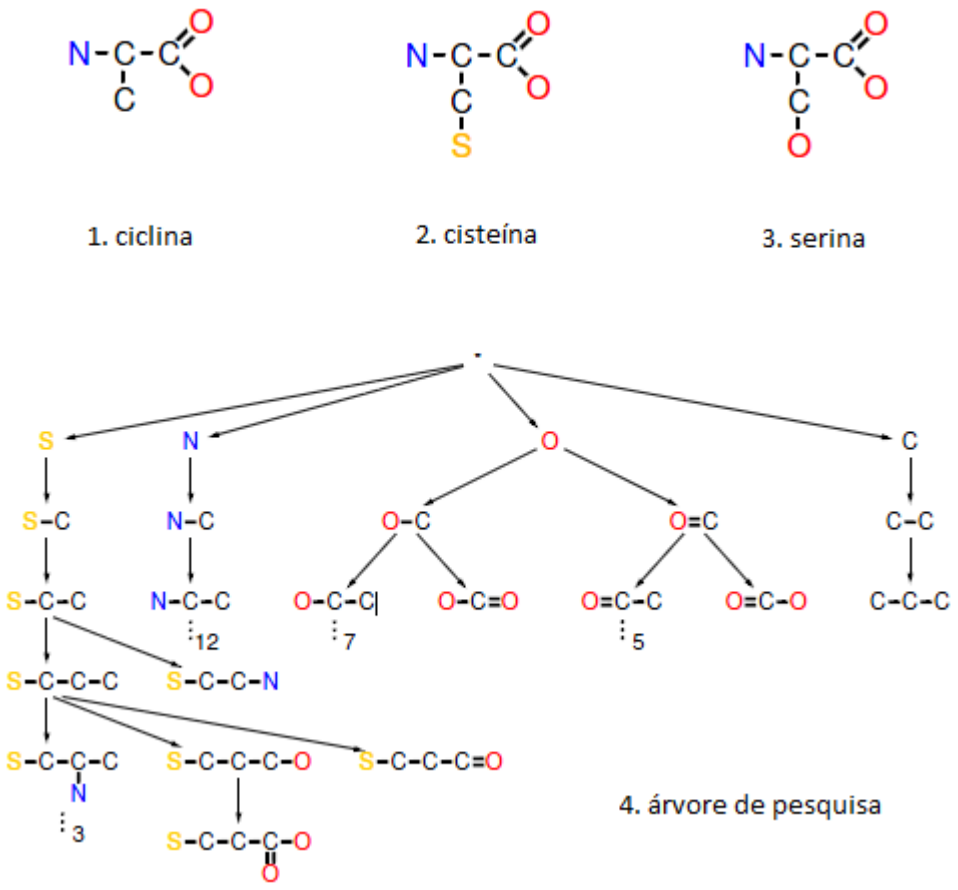

Figura 2.6: Moléculas ciclina, cisteína e serina e respectiva árvore de pesquisa [Borgelt 2004].

Na figura 2.6, temos um primeiro nível onde estão representados os átomos individuais, no segundo os pares conectados e por aí fora. Os pontos indicam subárvores que não estão descritas de modo a manter a imagem compreensível e os números associados representam o número de fragmentos nessas árvores. Esta imagem é importante para elucidar sobre o funcionamento dos cortes.

Primeiro os átomos estão ordenados pela ordem de frequência com que aparecem nas moléculas, começando do que tem menor frequência para o que tem maior, da esquerda para a direita, neste caso o átomo de enxofre. As moléculas vão ser então pesquisadas de modo a saber se possuem o átomo de enxofre e a sua localização é guardada. Neste exemplo o enxofre (S) pode apenas ser encontrado na cisteína o que leva a incorporação desta. Este fragmento de um átomo é então estendido, com a adição de uma aresta que o ligará a um átomo de carbono (C), o que resulta o fragmento S-C no nível seguinte. O resto das extensões é gerado nos níveis seguintes de forma análoga. Se o fragmento suporta mais do que uma extensão, como é o caso, estas são ordenadas de acordo com as regras locais já mencionadas [Borgelt 2002]. O principal motivo desta ordem é de prevenir a geração de certas extensões para evitar repetições. Por exemplo, no caso de S-C-C-C-O na segunda coluna, não é adicionado um átomo N (nitrogénio) porque o subgrafo S-C-C-C-N já foi considerado anteriormente na primeira coluna, tal como todas as extensões que incluem um átomo de enxofre e um de nitrogénio, não são consideradas pois já o foram na primeira coluna cuja raiz é o enxofre. Do mesmo modo nem o enxofre nem o nitrogénio são ponderados na coluna que tem como raiz o átomo de oxigénio.

No entanto, mesmo com estas evasões a repetições, a árvore de pesquisa não vai ser completamente atravessada. Sabendo que um fragmento deve ser frequente e a sua extensão vai reduzir essa frequência, tendo em conta que pode ser contido em menos moléculas, as subárvores podem ser cortadas logo que deixem de satisfazer determinados critérios definidos pelo

utilizador (cortes baseados em base). A árvore pode também ser limitada a um tamanho de fragmentos gerados (cortes baseados em tamanho). Basicamente o utilizador pode estabelecer certos critérios que limitarão a geração de fragmentos e consequentemente o consumo de recursos computacionais. Por exemplo, definir a procura para encontrar fragmentos frequentes relevantes nas moléculas activas e que sejam pouco frequentes em moléculas inactivas.

Como descrito no algoritmo MoFa, nem toda a pesquisa redundante pode ser eliminada, no entanto há possibilidades de melhorias neste sentido. Destas melhorias duas foram consideradas mais interessantes [Borgelt 2004], a primeira é chamada de cortes de irmãos equivalentes, onde é feita a verificação de filhos equivalentes num nó da árvore de pesquisa, a segunda é chamada de cortes de extensão perfeita que apenas segue um dos ramos da árvore de pesquisa se a extensão correspondente satisfazer um certo critério, pressupondo que os fragmentos são fechados.
Os fragmentos fechados consistem em fragmentos em que nenhum superfragmento contém a base, ou seja, o mesmo número daquelas específicas moléculas. Portanto, um subgrafo G é fechado se não existir um supergrafo de G que tenham a mesma base de G. O facto de restringirmos a pesquisa a fragmentos moleculares fechados não implica perda de informação, uma vez que todos os fragmentos podem ser construídos a partir de fragmentos fechados, simplesmente tendo em conta todas as subestruturas dos fragmentos fechados que não são fechadas e atribuindo-lhes o máximo de valores de bases dos fragmentos fechados que são seus descendentes. Logo, restringir a pesquisa a fragmentos fechados é uma boa maneira de reduzir a informação que esta percorrerá. Presumidamente, os químicos geralmente também não estão interessados em fragmentos frequentes que não são fechados, devido à sua redundância de informação [Borgelt 2004]. Na técnica de cortes de irmãos equivalentes, é feita uma verificação de

fragmentos similares, que tenham a mesma estrutura (irmãos), que irão gerar subárvores similares, o que é desnecessariamente redundante. Então, tendo em conta que o algoritmo MoFa considera nós irmãos, é feita uma ordenação baseada na informação do fragmento pai. De seguida, para cada nó, os nós anteriores são verificados em termos de equivalência, ou seja, se são irmão e em caso afirmativo são saltados.
A técnica de corte de extensão perfeita é baseada na observação de que por vezes há uma grande quantidade de fragmentos que são comuns em muitas moléculas. Desde que a pesquisa não tenha chegado a um fragmento que seja o máximo comum, não é necessária a sua evolução na árvore de pesquisa. A razão é simples, o fragmento máximo comum faz parte de todos os fragmentos fechados que podem ser encontrados no conjunto actual de moléculas, logo é suficiente seguir um caminho na árvore de pesquisa que levará ao fragmento máximo comum. Basicamente, se há uma estrutura comum em todos as moléculas consideradas cujo o fragmento actual é só uma parte, então qualquer extensão que não cresça o fragmento actual em direcção ao fragmento máximo comum pode ser adiada até que este máximo comum seja atingido.
Em de conclusão, pode-se dizer que ambas as técnicas aceleram um pouco o processo de pesquisa, principalmente a técnica de cortes de extensão perfeita em casos de fragmentos fechados.

Como era expectável os algoritmos analisados têm todos os seus prós e contras.
Todos os algoritmos são relativamente escaláveis. Todos mantêm uma performance aproximadamente linear com o aumento das bases de dados.
Um estudo comparativo [Worlein 2005] mostrou que o algoritmo mais

	IC93				HIV CA+CM			
	MoFa	gSpan	FFSM	Gaston	MoFa	gSpan	FFSM	Gaston
Duplicate filtering/pruning	11.3%	3.1%	0.1%	1.8%	12.3%	1.4%	0.2%	1.0%
Support computation	9.3%	62.9%	3.7%		9.6%	70.7%	3.3%	
Embedding list calculations	19.1%	-	60.4%	87.8%	18.1%	-	62.7%	95.9%
Extending of subgraphs	29.9%	17.3%	10.2%		31.1%	14.9%	8.1%	
Joining of subgraphs	-	-	0.1%	-	-	-	0.1%	-

Figura 2.7: Estudo comparativo [Worlein 2005] sobre as fases principais do processo de pesquisa de subgrafos e quanto tempo, relativo ao tempo total, cada um dos quatro algoritmos mencionados gasta nestas, para duas grandes bases de dados: IC93 e HIV CA + HIV CM.

lento seria o MoFa, mas este apresenta mais vantagens em termos de funcionalidades do que os outros.

As listas embutidas ao contrário do que se pensa, não dão uma grande vantagem de performance na pesquisa de fragmentos frequentes. Por exemplo, o gSpan não as utiliza, mas apresenta um desempenho aproximado ao do FFSM e ao do Gaston, apenas notando-se uma maior diferença com o aumento dos fragmentos. Por outro lado as listas podem provocar problemas devido à alocação de memória.

A geração de candidatos e as listas embutidas mostram-se mais importantes em termos de eficiência do que as estratégias de corte, tratando-se de evitar duplicados.

O Gaston apresenta uma boa vantagem no facto de evitar a duplicação de fragmentos.

Algoritmos testados em java apresentam diferentes resultados de performance, dependendo da versão da máquina virtual do Java.

2.2.3 Aplicações

O Graph Mining é uma área relativamente recente no contexto da extracção de informação. Mas apesar de ser uma técnica recente na área

de extracção de informação, já é utilizado em várias áreas, onde funciona como forma de extrair informação acerca das relações entre as entidades, sendo possível tirar conclusões acerca de comportamentos, e fazer simulações sobre o que aconteceria dados potenciais acontecimentos, através dos padrões observados, ou mesmo detectar anomalias, tendo em conta certos padrões.

Na área da Bioinformática, os grafos são usados para representarem estruturas de compostos químicos, interacções de genes e redes metabólicas usando vértices e arestas normais ou direccionadas. O Graph Mining é bastante útil na medida em que, a descoberta de subestruturas frequentes fornecem informações interessantes da base de dados que podem ser utilizadas de várias formas, como por exemplo, classificação de compostos químicos.

Estudos sobre interacção entre proteínas, toxicidade das moléculas ou identificação de actividade em subcompostos foram fortemente beneficiados. Grande parte destes avanços deve-se à maior disponibilidade de conjuntos de dados sobre estas áreas.

Outra importante aplicação do Graph Mining inclui a Internet. A Internet é muitas vezes vista como uma rede gigante de computadores conectados. Em geral pode ser vista como uma comunidade dividida em domínios, sendo cada domínio um grupo de nós (computadores pessoais) sob uma certa administração, o nó do router servidor. O Graph Mining é útil nesta área pois entre outras coisas permite estudos e análises sobre fluxos de tráfego e conexões, que podem ser utilizados em coisas como detecção de anomalias e problemas.

Também nas redes sociais o Graph Mining assumiu um papel de grande protagonismo.

Uma rede social é uma estrutura, tipicamente organizada em grafo, que representa entidades e as relações entre si. Estas relações podem significar várias coisas, como amizade, interesse, pertença, etc. As análises de re-

des sociais surgiram como uma forma de analisar estas relações, sendo possível fazer ilações sobre o comportamento do grupo de indivíduos estudado. Estas conclusões retiradas a partir das análises das redes sociais, permitem entre várias outras coisas, a empresas fazerem estudos sobre o impacto que o lançamento ou a modificação de um produto tiveram no mercado. Ou como certos grupos se relacionam entre si e entre pessoas de outros grupos. Basicamente a partir destas informações será mais fácil fazer previsões sobre acontecimentos sociais e o seu impacto em determinadas sociedades. Esta área da pesquisa e extracção de informação tem sido muito explorada inclusive por sociólogos.

Capítulo 3

Graph Mining em estudos de Toxicidade

Neste capítulo é descrito o projecto de tese em maior detalhe.
São abordados os dados utilizados, assim como os algoritmos mais adequados ao tema e o restante desenvolvimento do projecto.

3.1 Descrição

A proposta desta tese consiste uso de técnicas de Graph Mining para a pesquisa de informação sobre subestruturas de moléculas, procurando utilizar estas subestruturas na explicação das causas da actividade tóxica destas.
Será usado um algoritmo de Graph Mining que extrai informação sobre subestruturas de determinadas moléculas, fornecidas num conjunto de dados, onde estas são modelizadas por um grafo. Na representação adoptada, os vértices representam os átomos e as arestas as ligações químicas entre si, podendo estas serem simples, duplas e triplas.
O algoritmo deve ainda aceitar dois tipos diferentes de moléculas, sendo um grupo, o das moléculas tóxicas e o outro o das moléculas não tóxicas. Tendo em conta estes dois grupos o algoritmo deverá permitir encontrar

as subestruturas mais frequentes num grupo de moléculas e que são simultaneamente as menos frequentes no outro grupo.
Neste caso o ideal será encontrar as subestruturas frequentes das moléculas tóxicas que sejam pouco frequentes nas moléculas não tóxicas, sendo estas potencialmente passíveis de serem as causadoras da toxicidade na sua superestrutura.

3.2 MOSS

Tendo em conta o estudo efectuado sobre algoritmos actuais de Graph Mining, na área da bioinformática, mencionado anteriormente o algoritmo MoFa aparenta ser o único concebido especialmente para a pesquisa de moléculas, no entanto os algoritmos gSpan e Gaston apresentam um desempenho melhor.
Com o surgimento do MOSS, que apresenta como base o algoritmo MoFa, mas com o processamento especial do gSpan, estas falhas são colmatadas.
O MOSS (Molecular Substructure Miner) [Borgelt 2002], também conhecido por MoFa (- Molecular Fragment Miner) foi o algoritmo escolhido para fazer a pesquisa das submoléculas dos grupo de moléculas tóxicas que não ocorrem nas não tóxicas nos vários conjuntos de dados seleccionados.
A aplicação foi desenvolvida inicialmente pela empresa Tripos, Inc. em colaboração com Christian Borgelt, tendo sido mais tarde concluída por este.
Este programa contém um algoritmo que é baseado no algoritmo Eclat [Borgelt 2003] para a pesquisa de subconjuntos frequentes na pesquisa de conjuntos. O modo de processamento no entanto é baseado no algoritmo gSpan, mais propriamente na sua extenção CloseGraph [Yan and Han 2003]. O MOSS foi inicialmente designado para ser usado no contexto do desenvolvimento de novos fármacos e na previsão de sínteses. No

entanto após a versão 5.3 do algoritmo este sofreu modificações que lhe permitiram fazer pesquisas não só de conjuntos de dados de moléculas, como também qualquer conjunto de dados arbitrários que fossem representados em grafos e estivessem no formato de entrada típico do MOSS.

3.2.1 Procedimento

Dad um conjunto de dados de grafos, o MOSS encontra todas as subestruturas fechadas de acordo com uma frequência mínima de aparição definida pelo utilizador, excluindo as subestruturas cujas estruturas correspondentes ocorram com a mesma frequência.
Dados dois conjuntos diferentes denominados por conjunto de foco e conjunto complementar, são comparadas subestruturas que ocorram com grande frequência no conjunto de foco (neste caso, as moléculas tóxicas) e com rara frequência nas moléculas do conjunto complementar, ou seja, os fragmentos têm que aparecer com uma frequência mínima, definida na parte de foco e que não podem aparecer mais que a frequência máxima definida na parte complementar da base de dados, havendo uma descriminação de subestruturas dos dois conjuntos de moléculas.

Este foi o algoritmo de pesquisa de grafos eleito por diversas razões. Primeiramente, foi o único algoritmo encontrado que permitia a inserção de dois grupos diferentes como entrada, onde um grupo seria determinado como foco e o outro como complemento, tendo como opção por um lado encontrar as subestruturas mais frequentes no grupo de foco e por outro as estruturas menos frequentes no grupo de complemento, confrontando-as.

O MOSS permite também uma grande liberdade de pesquisa ao utilizador

apresentando um conjunto de parâmetros tão diversos como:
- Diferentes tipos de entrada, assim como de saída, podendo estes serem SMILES (o usado), SLN ou SDfile/Ctab.
- Tipo de nós/átomos a excluir.
- Inversão de grupos de foco e de complemento.
- Tamanho mínimo e máximo das subestruturas.
- Frequência mínima no grupo de foco.
- Frequência máxima no grupo de complemento.

Por último, o facto de o MOSS ter sido desenvolvido especialmente para a pesquisa de moléculas como grafos, adequa-se perfeitamente às necessidades encontradas. O modo de processamento baseado no gSpan torna-o num híbrido que suplanta tanto o básico MoFa como o gSpan em termos de processamento e eficiência.
O algoritmo descrito foi aplicado aos conjuntos de dados que serão mencionados na secção Dados (4.1), estes dados sofreram uma extração de informação, pois não se encontravam de acordo com as entradas, que serão agora descritas.

3.2.1.1 Entradas

As entradas (inputs) podem ter três tipos diferentes de codificação e representação de moléculas, SMILES (ver figura 3.1) (Simplified Molecular Input Line Entry System), que é o tipo mais popular e foi desenvolvido em 1986 por David Weiniger no Laboratório de Pesquisa Ambiental dos Estados Unidos da America. SLN (SYBYL® Line Notation) foi uma representação desenvolvida pela empresa Tripos, Inc. para o seu software SYBYL. Por fim, é aceite também o formato SDfile/Ctab (Structure-Data file / Connection table) que fazem parte dos ficheiros de formato CTfile desenvolvidos pela MDL Information Systems, Inc.. Têm uma representação multi-linha que fornece vários tipos de informação acerca das moléculas descritas, informação esta que é ignorada pelo MOSS.

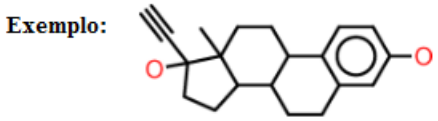

Figura 3.1: Exemplo da codificação SMILE e SLN na representação de uma molécula.

O tipo de codificação usado neste projecto é o SMILES.
Estas entradas estão organizadas de modo a que cada linha corresponda à descrição de uma molécula, sendo que cada linha contém um identificador da molécula, um valor, que corresponde ao facto de esta ser tóxicas ou não tóxica, sendo o grupo de moléculas tóxicas o grupo de foco e o grupo de moléculas não-tóxicas o grupo complementar, o que resulta que sejam pesquisadas as submoléculas tóxicas que não ocorram frequentemente nas não tóxicas, sendo possivelmente estas a causa da toxicidade, e uma descrição das moléculas na notação SMILE.

<id> , <valor> , <descrição>

Exemplo: 20973,0,[O-][N+](=O)C1=CC=C(O1)C=NN2CCNC2=O

<id >pode ser numérico ou string.
<valor >é um valor real compreendido entre 0 e 1, sendo as moléculas com valor superior a 0.5 consideradas no conjunto de foco e as restantes no conjunto complementar. Este valor pode ser alterado com o comando -t# na linha de comandos (sendo # o novo valor entre 0 e 1), ou invertido usando o comando -z na linha de comandos.
<descrição >é a descrição das moléculas, onde são descritos os seus átomos e ligações químicas entre si, segundo a sua notação SMILE correspondente.

Os ficheiros de dados fornecidos não estavam de acordo com as entradas do MOSS tendo sido desenvolvidos scripts que extraíram e organizaram os dados de foram a estes serem aceites no MOSS, que são explicados mais à frente.

3.2.1.2 Saída

A execução do MOSS sob o ficheiro de entrada fornece um output (resultado) que está organizado da seguinte maneira:

<id>, <descrição>, <nós>, <arestas>, <s_abs>, <s_rel>, <c_abs>, <c_rel>

<id> é um identificador normalmente da submolécula resultante.
<descrição> é a descrição da submolécula na notação SMILES.
<nós> o número de nós/átomos que a submolécula apresenta.
<arestas> o número de arestas/ligações que a submolécula apresenta.
<s_abs> é o número de frequência com que esta subestrutura aparece no grupo de foco.
<s_rel> é o número relativo de vezes, ou seja, a percentagem de vezes que a subestrutura aparece no grupo de foco tendo em conta todas as moléculas deste.
<c_abs> é o número de frequência com que esta subestrutura aparece no grupo complementar.
<c_rel> é o número relativo de vezes, ou seja, a percentagem de vezes que a subestrutura aparece no grupo complementar tendo em conta todas as moléculas deste.

Estes números são essenciais para análise de resultados por isso foram criados dois scripts, "Processa"e "Read"(descritos nos anexos), que usam estes resultados para cálculos de somas, médias, desvios padrão e diferenças, organizando também a informação de uma forma mais intuitiva em folhas excel, descritos com maior detalhe na secção de resultados.

Capítulo 4

Avaliação Experimental

Neste capítulo são descritos os resultados alcançados na avaliação experimental, assim como todas as experiências levadas a cabo para os alcançar e os dados utilizados nesta avaliação.

4.1 Dados

DSSTOX é o acrónimo de Distributed Structure-Searchable Toxicity, que é um projecto do Programa de Investigação de Toxicologia Computacional da Agência de Protecção Ambiental dos Estados Unidos da América. Este projecto pesquisa dados relativamente a toxicidade presentes em certas moléculas e estruturas químicas, fornecendo-os gratuitamente[1] de modo a que qualquer pessoa tenha acesso a estes, promovendo a investigação e o desenvolvimento.

Os dados são os mesmos usados no estudo comparativo de classificação de algoritmos que usam descritores moleculares em toxicologia [Pereira 2009].

Nesta tese serão abordados três conjuntos de dados principais em relação à toxicidade das moléculas e estruturas químicas. Estes dados sobre a toxicidade foram obtidos através de testes feitos a ratos em laboratório, são estruturados de uma forma especial, definida pela DSSTOX, de modo a

[1] http://www.epa.gov/ncct/dsstox/index.html

serem facilmente pesquisados. O formato dos dados é chamado de SDF (Structure Data Format) e foi desenvolvido e publicado pela Molecular Design Limited (MDL). Este modelo consiste em simples ficheiros de texto com a informação distribuída por campos que representam as estruturas químicas analisadas às quais estão associados os campos de informação.

4.1.1 CPDBAS

Carcinogenic Potency Database Summary Tables.
As tabelas dos sumários da base de dados da potência carcinogénica das moléculas representam o nível de toxicidade presente nestas moléculas em relação às hipóteses de aparecimento de cancro. Ou seja, são expostos os valores do nível toxicidade, relativamente a substâncias ou radiações presentes nas moléculas representadas, que sejam propensas à sensibilização do organismo do ser vivo para o surgimento de cancro.

Este conjunto de dados contém moléculas cuja descrição não estava correctamente definida tendo sido necessário recorrer à descrição destas em InChI[1], para ser possível traduzir esta para o formato correcto em SMILES, através do programa openbabel[2].
Apresentava também erros ao nível de informação das moléculas, tendo sido encontrados vários descritores de moléculas sem qualquer informação sobre a mesma, tanto a nível de toxicidade como da constituição da própria molécula.

4.1.2 DBPCAN

Water Disinfection By-Products with Carcinogenicity Estimates.
Este conjunto de dados contém estimativas previstas de potencial carcinogénico

[1] International Chemical Identifier
[2] http://openbabel.org/

para duzentos e nove químicos detectados em amostras de água potável que fora exposta a um tratamento de desinfecção de água. Este estudo foi feito inicialmente para comparações entre águas expostas a este tipo de tratamentos e é útil na medida em que pode ser usado também para comparações e estudos entre outro tipo de químicos, para fins medicinais e de produção de medicamentos.

4.1.3 NCTRER

National Center for Toxicological Research Estrogen Receptor Binding. Esta investigação toxicológica sobre a actividade do estrogénio fornece-nos informação acerca de uma avaliação quantitativa da capacidade de uma substância química se unir a um receptor de estrogénio. Esta base de dados contém 232 químicos, seleccionados tendo em conta as suas características estruturais.

conjunto de dados	activo	inactivo
CPDBAS	1059	1213
DBPCAN	80	98
NCTRER	131	93

Tabela 4.1: Caracterização quanto à distribuição do tipo de moléculas dos conjuntos de dados utilizados em relação à toxicidade [Pereira 2009].

4.2 Experiências

Os conjuntos de dados analisados estavam descritos no formato .sdf (Structure Data Files) contendo para cada molécula inúmeros campos, entre os quais se encontravam:
- <DSSTox_RID>, que é a identificação interna da molécula.
- <STRUCTURE_SMILES>, que descreve a molécula na estrutura SMILES já referenciada.
- <STRUCTURE_InChI>, InChI[1] é outro modo de descrever os átomos e ligações da molécula, neste caso muito importante dadas as irregularidades encontradas em algumas das moléculas CPDBAS.
- <ActivityOutcome>, campo que determinava a toxicidade da molécula podendo ser diferente noutros conjuntos de dados.

Para extrair apenas a informação necessária foram criados os scripts RecolherDados[2], JuntarDados[2] e mais tarde o processaDados[2]. O script RecolherDados percorre todas as linhas de cada ficheiro sdf[3] retirando apenas a informação relativa à identificação da molécula, a sua descrição em SMILES e o valor da toxicidade. No entando os campos do valor da toxicidade eram diferentes nos diferentes conjuntos de dados, revelando-se a informação ambígua em alguns deles. Por este motivo foram usadas outras listas, que já se encontravam divididas, sendo por isso duas para cada conjunto de dados, uma possuíndo os identificadores das moléculas tóxicas e a outra os identificadores das moléculas não tóxicas. Neste contexto foi elaborado o script JuntarDados que analisava cada uma destas listas e ia aos ficheiros anteriores retirar a informação relativa à descrição da molécula de acordo com os identificadores da lista, juntando no fim ambas

[1] International Chemical Identifier
[2] Ver mais detalhes na secção Anexos.
[3] Formato dos conjuntos de dados fornecidos

as listas obtendo um ficheiro[1] de entrada para a aplicação MOSS.
Foi mais tarde descoberto que estas listas apresentavam repetições de identificadores, o que foi corrigido usando o script DelRep[2] que eliminava qualquer identificador repetido que fosse encontrado num único ficheiro SMILE.
As irregularidades na descrição de certas moléculas do conjunto de dados CPDBAS foram detectadas após a tentativa de execução do MOSS com estes dados e a rejeição de execução deste.
As descrições defeituosas das moléculas foram detectadas e com a posse das suas descrições em InchI, recorreu-se à ferramenta openBabel[3] para transformar estas nas respectivas descrições correctas em SMILES[2].
Nos testes efectuados com o algoritmo MOSS foram abrangidos diversos parâmetros, havendo também uma variação em termos de valor dentro destes.
No total cada conjunto de dados foi processado 86 vezes tendo sido os resultados guardados em ficheiros numerados de 0 a 85 com o nome de xresi.sub, onde o x é a variável que contém o nome do conjunto de dados e i o número de teste (ex. CPDBASres0). O script Executar[2] foi o responsável por estas execuções autónomas com diferentes parâmetros.
O resultado 0 é deriva de uma execução feita manualmente com os parâmetros por defeito do MOSS, todos os restantes testes foram executados através do script Executar (ver em anexo).
O resultado 1 provém da execução feita a partir do script Executar com os parâmetros por defeito do MOSS executado pela linha de comandos.
Os resultados de 2 a 11 resultam de execuções que têm em vista alterar o tamanho mínimo das subestruturas a encontrar, começando em 1 até 10, sendo por omissão este tamanho 1.

[1] Ficheiro de entrada exemplo na secção Anexos
[2] Ver mais detalhes em anexo.
[3] A openBabel é uma ferramenta de código aberto que pode ser encontrada em: http://openbabel.org/

Os resultados de 12 a 52 derivam de execuções que avaliam as diferenças de execução em termos de tamanho máximo das subestruturas, por defeito este tamanho é infinito, não apresentando limite. Aqui são testados tamanhos máximos que variam entre 10 a 50.
Os resultados de 53 a 68 provêm de alterações nos parâmetros por defeito a nível de frequência mínima no grupo de foco. Este é testado entre 5% e 20%, sendo 10% o valor por omissão.
Os resultados de 69 a 83 resultam da alteração do parâmetro de frequência máxima no grupo complementar, tendo sido testados valores entre 1% e 15% sendo o valor por defeito 2%.
O resultado 84 mostra as subestruturas resultantes quando adicionada a opção de não uso do algoritmo Greedy na computação MIS (Maximum Independent Set).
O último resultado 85 provem da execução do MOSS de modo a que este não se restrinja apenas a procurar subestruturas (subgrafos) fechadas.

4.3 Medidas de Avaliação

Os resultados foram avaliados principalmente segundo três medidas: Exactidão[1], Precisão[2] e Recall.

$$\text{Exactidão} = \frac{\text{número correcto de instâncias classificadas}}{\text{número total de instâncias}}$$

A exactidão é calculada através das médias de frequência dos padrões encontrados nas moléculas de um dado conjunto de dados. Esta medida permite-nos ter uma ideia geral do desempenho de um dado estudo.

$$\text{Precisão(x)} = \frac{\text{número correcto de instâncias encontradas da classe x}}{\text{número de instâncias encontradas classificadas como x}}$$

$$\text{Recall} = \frac{\text{número correcto de instâncias encontradas da classe x}}{\text{número total de instâncias da classe x}}$$

A precisão e o recall são duas métricas que são usadas frequentemente para avaliar o desempenho de um algoritmo de reconhecimento de padrões.
Estas duas medidas são usadas geralmente na avaliação de dois grupos distintos. Sendo um dos grupos considerado relevante, a precisão mede a qualidade de instâncias relevantes encontradas, ou seja, dentro de um número encontrado de instâncias consideradas do pertencentes a um grupo relevante, é considerado o número de instâncias que são de facto pertencentes a esse grupo. É uma forma de medir a qualidade dos resultados.
No caso da métrica recall, é calculada através da divisão do número correcto de instâncias de uma classe x, a dividir por todas as instâncias da classe x existentes. De um modo simples, um alto valor de recall significa

[1] Accuracy.
[2] Precision.

que o algoritmo não falhou nenhuma instância, mas deverá ter também resultados que não são interessantes possivelmente baixando a precisão. Por outro lado uma alta precisão mostra que os resultados obtidos possuem boa qualidade, sendo quase todos relevantes, mas que poderá haver a ausência de outras instâncias relevantes devido aos apertados critérios de filtro na origem da alta precisão, o que implica um baixo valor de recall.
O desejável é que o valor de recall seja próximo de 1 de modo a todos os exemplos da classe x sejam bem classificados e que a precisão seja também próxima de 1 de modo a que o modelo não classifique instâncias de outras clases como sendo da classe x.

4.4 Resultados

Os resultados que apresentam uma maior discrepância entre ocorrência de subestruturas tóxicas das não tóxicas ocorreram em execuções do MOSS onde os parâmetros aumentavam a frequência máxima com que estas subestruturas ocorriam nas moléculas não-tóxicas, o que mostra que as subestruturas frequentes obtidas nas moléculas tóxicas são de certo modo frequentes também nas não tóxicas, o que pode levar a crer que as subestruturas encontradas não são as responsáveis pela toxicidade das suas superestruturas.

Através do script Executar já mencionado anteriormente, foi possível descobrir quais os parâmetros que obtêm os melhores resultados para os diferentes conjuntos de dados.
Esta descoberta deveu-se ao seguinte conjunto de procedimentos:
Primeiro foram detectadas as dez subestruturas de cada um dos resultados que apresentassem a maior diferença entre a sua frequência nas moléculas tóxicas e a sua frequência nas moléculas não tóxicas, deste modo evidenciando as subestruturas frequentes nas moléculas tóxicas que eram menos frequentes nas não tóxicas, sendo estas passíveis de serem a causa da toxicidade.
Estas subestruturas foram guardadas através do script Executar num ficheiro excel, onde são descritas através da sua identificação e valor da diferença.
Considerando o conjunto de todas as dez subestruturas com maior diferença, foi guardada a sua soma, média e desvio padrão.
Através da média das diferenças foram determinados os parâmetros que produziram melhores resultados, pois quanto maior a média de diferenças, maior a diferença média entre a frequência das subestruturas nas moléculas tóxicas e as não tóxicas, que era o pretendido.

De acordo com os resultados temos:

CPDBAS	DBPCAN	NCTRER
93,9	32,9	42,1

Tabela 4.2: Conjuntos de dados e respectivas maiores médias entre as diferenças de ocorrências nas moléculas tóxicas e não tóxicas.

Na tabela 4.2 é facilmente perceptível que o conjunto de dados CPDBAS apresenta as maiores médias de diferenças sendo este mais propenso a poder justificar a toxicidade das suas moléculas com as subestruturas encontradas, pois possuim subestructuras que tendem mais a aparecer nas moléculas tóxicas do que nas não tóxicas.
Nos três diferentes conjuntos de dados há um conjunto de parâmetros que se evidenciam produzindo as maiores diferenças entre as ocorrências dos padrões encontrados nas moléculas tóxias com as não tóxicas. São estes os resultados das execuções do MOSS entre 74 e 83, onde foram usados os parâmetros que variam a frequência máxima no grupo complementar, sendo esta por defeito 2%, e nos conjuntos de testes entre 74 e 83 esta é testada de 6% a 15%.

4.5 Qualidade dos resultados

De modo a comprovar que os resultados não seriam enviesados recorreu-se à técnica do cross validation.
Cross validation[1] é uma técnica da área da estatística usada para generalizar os resultados de um dado experimento, independentemente do conjunto de dados utilizado.
Consiste em dividir a amostra de dados em vários subconjuntos (normalmente dez) de modo a que a análise inicial a um deles seja depois comparada com a análise dos restantes. É uma técnica utilizada frequentemente em Inteligência Artificial de forma a validar os modelos gerados a partir de uma amostra.
Cada um destes dez conjuntos seria o conjunto de teste e seria utilizado para comparar os resultados deste com os resultados do modelo correspondente, que seria constituído por todos os outros nove subconjuntos.

Para executar o Cross Validation foram usadas um conjunto de listas já pré-definidas [Pereira 2009] com a divisão aleatória das moléculas em dez grupos para as moléculas tóxicas e outros dez para as moléculas não tóxicas. Para ser possível a execução destas listas no MOSS foi criado o script processaDados[2] que verificava o código da moléculas e abria os ficheiros resultantes do script RecolherDados para retirar a sua descrição em smile, atribuindo também a cada molécula o respectivo valor de toxicidade (0 ou 1, sendo 1 as não tóxicas).
Cada conjunto de dados ficou então com dez ficheiros de entrada, que seriam os modelos e dez ficheiros de saída que seriam os conjuntos de teste. Foi utilizado o script Executar2[2] para executar estes 60 ficheiros, sendo os argumentos manipulados de modo a que cada conjunto de dados obtivesse

[1] Validação cruzada
[2] Ver mais detalhes em anexo.

um número razoável de subestruturas como resultado, entre 9 e 20. O script Processa[2] foi então utilizado para processar os dados dos resultados, que foram organizados de modo a que as subestruturas encontradas no ficheiro de teste fossem as mesmas do que as encontras no modelo de treino, para permitir uma comparação, guardando-os num ficheiro excel. Dados esses que podem ser verificados na tabela 4.3.

Conjunto	Treino	Teste
CPDBAS	47,6%	53,4%
DPBCAN	52,8%	69,7%
NCTRER	43,6%	52,2%

Tabela 4.3: Exactidão - Médias das 10 melhores frequências de subestruturas nos 10 conjuntos de treino e teste.

De acordo com os valores calculados a partir da métrica da precisão (ver tabela 4.4) é possível dizer que os resultados possuiam qualidade, na medida em que a maior parte das subestruturas encontradas tinham a sua grande predominância de frequência nas moléculas tóxicas como pretendido.
Quanto ao Recall, os valores obtidos (ver tabela 4.4) mostram que em média cada padrão encontrado está frequente em apenas uma pequena parte (entre 8,3tóxicas do respectivo conjunto de dados. Ou seja, possivelmente nem todas as moléculas consideradas tóxicas apresentam subestruturas que sejam frequentes também nas outras moléculas tóxicas.

Conjunto de dados	Precisão	Recall
CPDBAS	80,2%	8,3%
DPBCAN	99,3%	32,9%
NCTRER	98,6%	22,0%

Tabela 4.4: Médias de Precisão e Recall das 10 melhores frequências de subestruturas nos 10 conjuntos de teste.

Apesar dos resultados do grupo de treino apresentarem uma certa discrepância com os resultados do grupo de teste, não permitindo grandes

previsões, os resultados dos diferentes grupos de treino mostram uma certa consistência entre si, o que revela que no geral o algoritmo será eficiente. A adição de mais dados poderia diminuir esta discrepância.
No geral os valores da exactidão apresentam um valor baixo o que sugere que os padrões encontrados poderão não ser suficientemente bons para explicar a toxicidade nas moléculas.

Capítulo 5

Conclusões

Neste capítulo são apresentadas as conclusões do trabalho.
São também comentados os resultados obtidos, assim como feitas referências a um eventual trabalho futuro.

5.1 Satisfação dos Objectivos

A área de Graph Mining é uma área muito activa hoje em dia. Desde aplicações nas áreas da informática como a Internet e as redes sociais até à bioinformática, onde interacções entre proteínas, redes metabólicas e pesquisa molecular ganharam novos contornos, o Graph Mining vem ganhando uma grande importância.
Uma das conclusões a retirar deste trabalho é que a aplicação de um algoritmo de Graph Mining ao problema da construção de modelos de Toxicidade é bastante vantajosa. Em particular, a escolha do algoritmo MOSS neste sentido mostrou-se acertada, dado que o este algoritmo permitiu cumprir um dos principais objectivos que era a pesquisa de subestruturas frequentes em moléculas tóxicas. Deste modo foi exequível obter subestruturas que são passíveis de podem estar na origem da toxicidade das moléculas onde ocorrem. No entanto estes resultados só poderão ser "validados"por especialistas.

Não houve, durante a realização do trabalho, quaisquer problemas em termos de processamento, tendo o algoritmo/aplicação se revelado bastante rápido com tempos de execução inferiores a 1 segundo, aquando execuções individuais. Aquando grupos de cerca de 30 execuções tinham tempos à volta dos 10 segundos.

5.2 Trabalho Futuro

Como trabalho futuro é sugerida a utilização de mais algoritmos de Graph Mining de modo a poder ser comparado o seu desempenho na tarefa de construção de modelos de previsão de toxicidade.
É também sugerida a utilização de mais conjuntos de dados. A toxicidade pode ter origem em diferentes "fontes"e daí ser desejável ter uma grande diversidade de conjuntos de dados onde as diferentes origens da toxicidade possam estar representadas.

Referências Bibliográficas

1. BALANI, S.K., MIWA G.T., GAN L.S. , WU J.T., LEE F.W. 2005. Strategy of utilizing in vitro and in vivo ADME tools for lead optimization and drug candidate selection. Millennium Pharmaceuticals, Inc., Cambridge, MA 02139, USA.

2. BI, Z., FALOUTSOS, C., AND KORN, F. 2001. The DGX distribution for mining massive, skewed data. In Conference of the ACM Special Interest Group on Knowledge Discovery and Data Mining. ACM Press, New York, NY, 17–26.

3. BORGELT, C., BERTHOLD, M.R. 2002. Mining molecular fragments: finding relevant substructures of molecules. Data Mining, 2002. ICDM 2002. Proceedings. 2002 IEEE International Conference on , vol., no., pp. 51- 58

4. BORGELT C. 2003. Efficient Implementations of Apriori and Eclat, Proc. 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL). CEUR Workshop Proceedings 90

5. BORGELT C., MEINL T., BERTHOLD M.R. 2004. Advanced pruning strategies to speed up mining closed molecular fragments Systems, Man and Cybernetics, 2004 IEEE International Conference on , vol.5, no., pp. 4565- 4570 vol.5, 10-13 Oct.

6. BORGWARDT, K., YAN X. Graph Mining and Graph Kernels.

7. BRANDES,U.,GAERTLER, M., ANDWAGNER,D. 2003. Experiments on graph clustering algorithms. In European Symposium on Algorithms. Springer Verlag, Berlin, Germany.

8. CHAKRABARTI D., FALOUTSOS C. 2006. Graph Mining: Laws, Generators, and Algorithms. Journal ACM Computing Surveys (CSUR) Volume 38 Issue 1.

9. FALOUTSOS, M., FALOUTSOS, P., AND FALOUTSOS, C. 1999. On power-law relationships of the Internet topology. In Conference of the ACM Special Interest Group on Data Communications (SIGCOMM). ACM Press, New York, NY, 251–262.

10. FISCHER I., MEINL T. 2004. Graph Based Molecular Data Mining - An Overview. IEEE SMC 2004 Conference Proceedings pp. 4578-4582

11. FLAKE, G.W.,LAWRENCE, S., ANDGILES, C. L. 2000. Efficient identification of Web communities. In Conference of the ACM Special Interest Group on Knowledge Discovery and Data Mining. ACM Press, New York, NY.

12. GIBSON, D., KLEINBERG, J., AND RAGHAVAN, P. 1998. Inferring web communities from link topology. In ACM Conference on Hypertext and Hypermedia. ACM Press, New York, NY, 225–234.

13. HUAN, J., WANG W., PRINS, J. 2003. Efficient mining of frequent subgraphs in the presence of isomorphism. ICDM 2003. Third IEEE International Conference on , vol., no., pp. 549- 552, 19-22 Nov. 2003

14. KARYPIS, G. AND KUMAR, V. 1998. Multilevel algorithms for multi-constraint graph partitioning. Tech. Rep. 98-019, University of Minnesota.

15. KLEINBERG, J. 1999. Authoritative sources in a hyperlinked environment. J. ACM 46, 5, 604–632.

16. KURAMOCHI M., KARYPIS G. 2001. Frequent Subgraph Discovery, icdm, pp.313, First IEEE International Conference on Data Mining (ICDM'01)

17. KURAMOCHI M., KARYPIS G. 2005. Finding Frequent Patterns in a Large Sparse Graph*. Data Min. Knowl. Discov. 11, 3 (November 2005), 243-271.

18. LAM, W. W. M., CHAN K. C. C. 2008. A Graph Mining Algorithm for Classifying Chemical Compounds. Bioinformatics and Biomedicine, IEEE International Conference on, pp. 321-324, 2008 IEEE International Conference on Bioinformatics and Biomedicine.

19. LAM, W. W. M., CHAN K. C. C., CHIU D. K. Y., WONG A. K. C., 2007. MAGMA: An Algorithm for Mining Multi-level Patterns in Genomic Data. bibm, pp.89-94, 2007 IEEE International Conference on Bioinformatics and Biomedicine (BIBM 2007)

20. Meinl, T.; Berthold, M.R. 2004. Hybrid fragment mining with MoFa and FSG, Systems, Man and Cybernetics, 2004 IEEE International Conference on , vol.5, no., pp. 4559- 4564 vol.5, 10-13 Oct.

21. MCGLOHON M., FALOUTSOS, C. 2008 Graph Mining, Techniques for Social Media Analysis. International Conference on Weblogs and Social Media

22. NIJSSEN S., KOK J. N. 2005. The Gaston Tool for Frequent Subgraph Mining. Electron. Notes Theor. Comput. Sci. 127, 1 (March 2005), 77-87.

23. PALMER, C., GIBBONS, P. B., AND FALOUTSOS, C. 2002. ANF: A fast and scalable tool for data mining in massive graphs. In Conference of the ACMSpecial Interest Group on Knowledge Discovery and Data Mining. ACM Press, New York, NY.

24. PEI J., HAN J., MORTAZAVI-ASL B., PINTO H., CHEN Q., DAYAL U., HSU M. C. 2001. PrexSpan: Mining sequential patterns efficiently by prex-projected pattern growth. In ICDE'01, pages 215-224, April.

25. PENNOCK,D. M., FLAKE,G.W.,LAWRENCE, S.,GLOVER, E. J., ANDGILES,C. L. 2002. Winners don't take all: Characterizing the competition for links on the Web. Proceedings of the National Academy of Sciences 99, 8, 5207–5211.

26. PEREIRA M., COSTA V. S., CAMACHO R., FONSECA N. A., SIMÕES C., BRITO R. M. 2009. Comparative Study of Classification Algorithms Using Molecular Descriptors in Toxicological DataBases. In Proceedings of the 4th Brazilian Symposium on Bioinformatics: Advances in Bioinformatics and Computational Biology (BSB '09), Katia S. Guimarães, Anna Panchenko, and Teresa M. Przytycka (Eds.). Springer-Verlag, Berlin, Heidelberg, 121-132.

27. PODOLYAN Y., KARYPIS G. 2009. Common Pharmacophore Identification Using Frequent Clique Detection Algorithm. Journal of Chemical Information and Modeling 2009 49 (1), 13-21

28. ROBERTS A. 2005. Guide to Weka

29. VIRTANEN, S. 2003. Clustering the Chilean Web. In Latin AmericanWeb Congress. IEEE Computer Society Press, Los Alamitos, CA.

30. WORLEIN M., MEINL T., FISCHER I., PHILIPPSEN M. 2005 A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM,

and Gaston. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, pp. 392-403. Springer, Heidelberg (2005).

31. WU, F. AND HUBERMAN, B. A. 2004. Finding communities in linear time: A physics approach. European Physics J. B 38, 2, 331–338.

32. WU J., CHEN L., 2008 A Fast Frequent Subgraph Mining Algorithm Young Computer Scientists, International Conference for, pp. 82-87, 2008 The 9th International Conference for Young Computer Scientists.

33. YAN, X. AND HAN, J. 2002. gSpan: Graph-based substructure pattern mining. In IEEE International Conference on Data Mining. IEEE Computer Society Press, Los Alamitos, CA.

Anexos

Neste trabalho foram elaborados diversos scripts e programas auxiliares todos desenvolvidos em Java.
Aqui é feita uma breve descrição destes, e mais à frente apresentado o seu código.

Foi inicialmente criado um script que analisava todos os ficheiros sdf (structure data file) e retirava apenas a informação necessária, a identificação, o código smile e o valor da toxicidade. - script RecolherDados
Devido a ambiguidades nos valores de toxicidade num dos conjuntos de dados (CPDBAS) foi criado outro script que analisava listas fornecidas que apenas possuíam o identificador e o valor de toxicidade, sendo que o script necessitava de ir retirar aos ficheiros sdf anteriores o código smile para cada ID e organizar a informação de modo a ser aceite como entrada na aplicação MOSS. - script JuntarDados
As listas fornecidas apresentavam repetições pelo que foi criado um script para eliminar estas. - script DelRep
Consequentemente foi criado um script que executava o programa MOSS automaticamente e com os diferentes parâmetros para todos os conjuntos de dados. - script Executar
Outro script que analisava as saídas do anterior foi criado, calculando também métricas como a soma, a média e o desvio padrão de um top 10 de diferenças também aí calculado e gravava estes num ficheiro excel. - script

Read
Foi criado um novo script para converter as listas da analise de resultados, com os conjuntos de treino e de teste para cross validation em ficheiros de entrada para o MOSS. - script processaDados
Logo a necessidade de um script para testar automaticamente as listas de cross validation no MOSS, recorrendo à execução deste surgiu. Este script calculava a diferença entre os dois ficheiros resultantes do MOSS e gravava estas diferenças num ficheiro excel. - scripts Executar2 e Processa
Por fim foi criado um script na eminência da utilização dos algoritmos gSpan, Gaston e Parsemis, transformando os dados presentes nos ficheiros sdf em entradas para estes. - script Convertor

Aqui na secção dos Anexos são também disponibilizadas as tabelas com os resultados da Validação Cruzada relativas aos três diferentes conjuntos de dados e a descrição correcta das moléculas irregulares do conjunto CPDBAS.

5.3 Resultados da Validação Cruzada

As listas de cross validation fornecidas apresentavam um menor número de moléculas na décima divisão de teste (cv10) do que nas restantes nove, o que pode justificar o facto de haver uma maior diferença entre os valores de treino e os valores de teste nesta divisão, pois apresentando um menor número de moléculas, menos moléculas foram retiradas do conjunto de treino e os resultados no teste não serão tão precisos.

CPDBAS			
Treino		Teste	
Tóxicas	Não Tóxicas	Tóxicas	Não Tóxicas
13,39	2,86	5,251	2,235
13,01	2,75	6,313	2,570
13,11	3,12	5,754	0,726
12,54	2,74	8,045	2,514
13,06	3,00	5,698	1,285
13,38	2,76	4,190	2,570
12,82	3,14	6,872	0,782
13,43	2,97	4,413	1,508
12,29	2,94	9,106	1,285
12,85	3,13	6,034	0,559

Tabela 5.1: Médias das 10 melhores frequências de subestruturas nos 10 conjuntos de treino e teste do conjunto CPDBAS.

DBPCAN			
Treino		Teste	
Tóxicas	Não Tóxicas	Tóxicas	Não Tóxicas
27,64	0,57	12,222	0,000
26,81	0,45	12,222	0,556
27,64	0,23	8,889	1,667
27,08	0,57	10,556	0,000
25,56	0,57	15,556	0,000
27,08	0,45	11,111	0,556
25,28	0,45	18,333	0,000
26,53	0,57	12,778	0,000
25,69	0,57	17,778	0,000
25,97	0,44	16,111	0,000

Tabela 5.2: Médias das 10 melhores frequências de subestruturas nos 10 conjuntos de treino e teste do conjunto DBPCAN.

NCTRER			
Treino		Teste	
Tóxicas	Não Tóxicas	Tóxicas	Não Tóxicas
22,65	0,72	11,667	1,250
21,11	0,96	12,917	0,000
21,03	0,96	13,333	0,000
20,00	0,84	11,250	0,833
20,00	1,08	17,500	0,000
20,09	0,84	16,667	0,417
22,48	0,84	9,583	0,833
21,11	1,08	13,333	0,000
20,00	0,84	17,500	0,000
21,90	1,00	0,417	0,000

Tabela 5.3: Médias das 10 melhores frequências de subestruturas nos 10 conjuntos de treino e teste do conjunto NCTRER.

5.4 Descrições irregulares

As moléculas seguintes foram as detectadas irregulares no conjunto de dados CPDBAS, estando aqui apresentadas na sua forma de descrição em SMILES correcta, sendo seis no total:

1. 20235,0,[Ca+2].[Cl-].[Cl-]
2. 21271,0,[Cl-].[Na+]
3. 20358,0,C([C@@H]1[C@@H]2[C@@H]([C@H]([C@H](O1)O[C@@H]1[C@@H]
4. 21264,0,O=[Se]=O
5. 20622,0,[Cl-].[Cl-].[Cl-].[Fe+3]
6. 21178,0,[Cl-].[K+]

5.5 RecolherDados

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
//import java.io.InputStreamReader;
import java.io.PrintWriter;

/**
 *
 * @author seabra
 */
public class RecolherDados {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException {
        // TODO code application logic here

        processaDados("NCTRER.sdf");
        processaDados("EPAFHM.sdf");
        processaDados("DBPCAN.sdf");
        processaDados("CPDBAS.sdf");
```

```
}

static void processaDados(String entrada) throws IOException{

    //começar a abrir ficheiros e a ler

    //BufferedReader nomeFicheiro = new BufferedReader
    (new InputStreamReader(System.in));
    //String entrada = "NCTRER.sdf";//nomeFicheiro.readLine();
    String[] ent = entrada.split(".sdf");
    System.out.println("A processar: " + ent[0]);


    File saida = new File(ent[0] + "_temp.smi");


    if(!saida.exists()){
        saida.createNewFile();
    }

    FileWriter fw = new FileWriter(saida);
    PrintWriter pw = new PrintWriter(fw);

  BufferedReader br = new BufferedReader(new FileReader(entrada));
    int flag = 0;
    int count = 0;

    String linha = "";
```

```
while((linha = br.readLine()) != null){

    //System.out.println(linha);
    if (flag == 1){
        pw.println("," + linha);
        //System.out.println("Imprime: " + count + "," + linha);
        flag = 0;
        count++;
    }
    else if (flag == 2){
        pw.print(linha);
        //System.out.println("Imprime: " +  "," + linha);
        flag = 0;
    }
    else{

   if (linha.equalsIgnoreCase("> <STRUCTURE_SMILES>")){
            flag = 1;
            //System.out.println("Smile");
        }
        if (linha.equalsIgnoreCase("> <DSSTox_RID>")){
            flag = 2;
            //System.out.println("Actividade");
        }
    }

}

pw.close();
```

```
    }

}
```

5.6 JuntarDados

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;


public class JuntarDados {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException {
        // TODO code application logic here

        processaDados("cpdbas");
        processaDados("dbpcan");
        processaDados("epafhm");
        processaDados("nctrer");
```

```
}

static void processaDados(String entrada) throws IOException{

    /*
    System.out.println("A processar: " + entrada);


    File saida = new File(entrada + "_tratado");

    String[] ent = entrada.split("_");

    if(!saida.exists()){
        saida.createNewFile();
    }

    FileWriter fw = new FileWriter(saida);
    PrintWriter pw = new PrintWriter(fw);

  BufferedReader br = new BufferedReader(new FileReader(entrada));

    String linha = "";
    int toxicidade = 2;

    if (ent[1].equalsIgnoreCase("toxicos"))
        toxicidade = 1;
    else if(ent[1].equalsIgnoreCase("naotoxicos"))
        toxicidade = 0;
    else System.out.println("Erro na toxicidade!");
```

```
while((linha = br.readLine()) != null){
    pw.println(linha + "," + toxicidade);

}
pw.close();
 *
 * */


// >  <DSSTox_RID>

//introduzir smiles

// USAR O script 1 para por o <DSSTox_RID> nos ficheiros
//extrair os smiles de lá, verificando se os rid coincidem e se
//sim passar para la o smile correspondente


String entrada2 = entrada.toUpperCase();

BufferedReader brEnt = new BufferedReader(new FileReader(entrada2 +
    "_temp.smi"));

BufferedReader br1 = new BufferedReader(new FileReader(entrada +
    "_toxicos"));
BufferedReader br2 = new BufferedReader(new FileReader(entrada +
    "_naotoxicos"));
```

```
File final1 = new File(entrada.toUpperCase() + "_toxicos.smi");
File final2 = new File(entrada.toUpperCase() + "_naotoxicos.smi");

if(!final1.exists()){
    final1.createNewFile();
}
if(!final2.exists()){
    final2.createNewFile();
}

FileWriter fw1 = new FileWriter(final1);
PrintWriter pw1 = new PrintWriter(fw1);


String linha = "";
String linhaE = "";
String[] linha2 = null;
String[] linhaE2 = null;
String id = "";


while ((linhaE = br1.readLine()) != null){

    linhaE2 = linhaE.split(",");
    id = linhaE2[0];
    //System.out.println(id);
brEnt = new BufferedReader(new FileReader(entrada2 + "_temp.smi"));
    while((linha = brEnt.readLine()) != null){
        //System.out.println(linha);
        linha2 = linha.split(",");
```

```
        //System.out.println(id);
        //System.out.println(linha2.length);
        if (linha2[0].equalsIgnoreCase(id)){

            pw1.println(id + "," + linhaE2[1] + "," + linha2[1]);
            //System.out.println("ENCONTROU!");
        }

    }
    brEnt.close();

}

pw1.close();

FileWriter fw2 = new FileWriter(final2);
PrintWriter pw2 = new PrintWriter(fw2);



linha = "";
linhaE = "";
linha2 = null;
linhaE2 = null;
id = "";

while ((linhaE = br2.readLine()) != null){
    linhaE2 = linhaE.split(",");
    id = linhaE2[0];

brEnt = new BufferedReader(new FileReader(entrada2 + "_temp.smi"));
```

```
            while((linha = brEnt.readLine()) != null){
                //System.out.println(linha);

                linha2 = linha.split(",");
                //System.out.println(linha2.length);
                if (linha2[0].equalsIgnoreCase(id))
                    pw2.println(id + "," + linhaE2[1] + "," + linha2[1]);

            }
            brEnt.close();

        }

        pw2.close();


    }

}
```

5.7 DelRep

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
```

```
import java.util.Vector;


public class Delrep {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws FileNotFoundException,
     IOException {
        // TODO code application logic here

        delRep("CPDBAS.smi"); //rep
        //delRep("NCTRER.smi");
        //delRep("DBPCAN.smi");
        //delRep("EPAFHM.smi");



    }

    public static void delRep(String ent) throws FileNotFoundException,
     IOException{

        BufferedReader br = new BufferedReader(new FileReader(ent));

        File f = new File("SR" + ent);

        if(!f.exists()){
            f.createNewFile();
```

```
        }

        FileWriter fw = new FileWriter(f);
        PrintWriter pw = new PrintWriter(fw);

        String linha = "";
        String[] linhaSP = null;
        Vector<Integer> v = new Vector<Integer>();

        while ((linha = br.readLine()) != null){

            linhaSP = linha.split(",");

            if (!v.contains(Integer.parseInt(linhaSP[0]))){
                v.add(Integer.parseInt(linhaSP[0]));
                pw.println(linha);
                pw.flush();
                System.out.println(linha);
            }

        }


    }

}
```

5.8 Executar

```
/*
```

```
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */


public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException {

        //DBPCAN
        int count = 1;
     Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z DBPCAN.smi
         DBPCANres" + count + ".sub");
        count++;
        for (int i=1; i<11; i++){
        Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -m" + i +
             " DBPCAN.smi DBPCANres" + count + ".sub");
            count++;
        }

        for (int i=10; i<51; i++){
        Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -n" + i +
             " DBPCAN.smi DBPCANres" + count + ".sub");
            count++;
        }

        for (int i=5; i<21; i++){
```

```
Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -s" + i +
    " DBPCAN.smi DBPCANres" + count + ".sub");
   count++;
}

for (int i=1; i<16; i++){
Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -S" + i +
    " DBPCAN.smi DBPCANres" + count + ".sub");
   count++;
}


Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -G DBPCAN.smi
   DBPCANres" + count + ".sub");
  count++;
Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -C DBPCAN.smi
   DBPCANres" + count + ".sub");


  //CPDBAS
  /*
  count = 1;

Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z CPDBAS.smi
   CPDBASres" + count + ".sub");
  count++;
  for (int i=1; i<11; i++){
  Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -m" + i +
      " CPDBAS.smi CPDBASres" + count + ".sub");
     count++;
```

```
}

for (int i=10; i<51; i++){
Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -n" + i +
    " CPDBAS.smi CPDBASres" + count + ".sub");
   count++;
}

for (int i=5; i<21; i++){
Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -s" + i +
    " CPDBAS.smi CPDBASres" + count + ".sub");
   count++;
}

for (int i=1; i<16; i++){
Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -S" + i +
    " CPDBAS.smi CPDBASres" + count + ".sub");
   count++;
}


Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -G CPDBAS.smi
   CPDBASres" + count + ".sub");
  count++;
Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -C CPDBAS.smi
   CPDBASres" + count + ".sub");

  */
  //EPAFHM
  count = 1;
```

```
Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z EPAFHM.smi
  EPAFHMres" + count + ".sub");
  count++;
  for (int i=1; i<11; i++){
  Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -m" + i +
      " EPAFHM.smi EPAFHMres" + count + ".sub");
     count++;
  }

  for (int i=10; i<51; i++){
  Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -n" + i +
      " EPAFHM.smi EPAFHMres" + count + ".sub");
     count++;
  }

  for (int i=5; i<21; i++){
  Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -s" + i +
      " EPAFHM.smi EPAFHMres" + count + ".sub");
     count++;
  }

  for (int i=1; i<16; i++){
  Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -S" + i +
      " EPAFHM.smi EPAFHMres" + count + ".sub");
     count++;
  }


Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -G EPAFHM.smi
   EPAFHMres" + count + ".sub");
```

```
  count++;
Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -C EPAFHM.smi
   EPAFHMres" + count + ".sub");


  //NCTRER
  count = 1;


Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z NCTRER.smi
  NCTRERres" + count + ".sub");
  count++;
  for (int i=1; i<11; i++){
  Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -m" + i +
       " NCTRER.smi NCTRERres" + count + ".sub");
      count++;
  }

  for (int i=10; i<51; i++){
  Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -n" + i +
       " NCTRER.smi NCTRERres" + count + ".sub");
      count++;
  }

  for (int i=5; i<21; i++){
  Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -s" + i +
       " NCTRER.smi NCTRERres" + count + ".sub");
      count++;
  }
```

```
        for (int i=1; i<16; i++){
        Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -S" + i +
            " NCTRER.smi NCTRERres" + count + ".sub");
           count++;
        }


      Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -G NCTRER.smi
        NCTRERres" + count + ".sub");
        count++;
      Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -z -C NCTRER.smi
        NCTRERres" + count + ".sub");

    }

}
```

5.9 Read

```
import com.sun.corba.se.spi.ior.Writeable;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
```

```
import java.util.HashMap;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;

import java.io.File;
import java.util.Date;
import jxl.*;
import jxl.write.*;
import jxl.write.Number;



public class Read {



  public static void main(String[] args) throws FileNotFoundException, IOException,
    WriteException {
      // TODO code application logic here

    WritableWorkbook workbook = Workbook.createWorkbook(new File("output.xls"));



      try{
```

```
    WritableSheet sheet1 = workbook.createSheet("CPDBAS", 0);

Label labelD = new Label(0, 0, "Dez subestruturas com maior diferença");
    sheet1.addCell(labelD);
    Label labelS = new Label(1, 0, "Soma total:");
    sheet1.addCell(labelS);
    Label labelM = new Label(2, 0, "Media");
    sheet1.addCell(labelM);
    Label labelP = new Label(3, 0, "Desvio padrão:");
    sheet1.addCell(labelP);

    int count = 1;

    for (int i=0; i<86; i++){
       processaDados("CPDBASres" + i + ".sub", sheet1, count);
        count++;
    }



    count = 1;
   WritableSheet sheet2 = workbook.createSheet("DBPCAN", 1);

Label labelD2 = new Label(0, 0, "Dez subestruturas com maior diferença");
    sheet2.addCell(labelD2);
    Label labelS2 = new Label(1, 0, "Soma total:");
    sheet2.addCell(labelS2);
    Label labelM2 = new Label(2, 0, "Media");
    sheet2.addCell(labelM2);
    Label labelP2 = new Label(3, 0, "Desvio padrão:");
    sheet2.addCell(labelP2);
```

```
    for (int i=0; i<86; i++){
       processaDados("DBPCANres" + i + ".sub", sheet2, count);
        count++;
    }



    count = 1;
   WritableSheet sheet3 = workbook.createSheet("EPAFHM", 2);

Label labelD3 = new Label(0, 0, "Dez subestruturas com maior diferença");
    sheet3.addCell(labelD3);
    Label labelS3 = new Label(1, 0, "Soma total:");
    sheet3.addCell(labelS3);
    Label labelM3 = new Label(2, 0, "Media");
    sheet3.addCell(labelM3);
    Label labelP3 = new Label(3, 0, "Desvio padrão:");
    sheet3.addCell(labelP3);

    for (int i=0; i<86; i++){
       processaDados("EPAFHMres" + i + ".sub", sheet3, count);
        count++;
    }

    count = 1;
   WritableSheet sheet4 = workbook.createSheet("NCTRER", 3);

Label labelD4 = new Label(0, 0, "Dez subestruturas com maior diferença");
    sheet4.addCell(labelD4);
```

```
        Label labelS4 = new Label(1, 0, "Soma total:");
        sheet4.addCell(labelS4);
        Label labelM4 = new Label(2, 0, "Media");
        sheet4.addCell(labelM4);
        Label labelP4 = new Label(3, 0, "Desvio padrão:");
        sheet4.addCell(labelP4);

        for (int i=0; i<86; i++){
           processaDados("NCTRERres" + i + ".sub", sheet4, count);
            count++;
        }

    } catch (IOException e) {
   System.out.println("Falha ao processar dados: "+e.getMessage());
    }


    workbook.write();
    workbook.close();



}


public static void processaDados(String fic, WritableSheet sheet, int c)
 throws FileNotFoundException, IOException, WriteException{

    //abrir ficheiro
    // id,description,nodes,edges,s_abs,s_rel,c_abs,c_rel
```

```
    // [0], [1],        [2],   [3],   [4] , [5] , [6] , [7]
    // gravar dados dos ficheiros
    // fazer calculos
    // gravar para ficheiro excel

//WriteableWorkbok workbook = Workbok.createWorkbook(new File("output.xls"))
    /*String[] fic2 = fic.split(".sub");
    File finale = new File(fic2[0] + "info.txt");

    if(!finale.exists()){
        finale.createNewFile();
    }

    FileWriter fw = new FileWriter(finale);
    PrintWriter pw = new PrintWriter(fw);*/


    int soma = 0;
    double media = 0.0;
    double desvio = 0.0;

HashMap<Integer,Integer> tabela = new HashMap<Integer,Integer>();
    HashMap tabelaO = new LinkedHashMap<Integer, Integer>();
Hashtable<Integer, Integer> dez = new Hashtable<Integer, Integer>();


    BufferedReader br = new BufferedReader(new FileReader(fic));

    String linha = "";
    String[] linhaSP = null;
```

```
br.readLine();

while ((linha = br.readLine()) != null){

    linhaSP = linha.split(",");

tabela.put(Integer.parseInt(linhaSP[0]),(Integer.parseInt(linhaSP[4])
    -Integer.parseInt(linhaSP[6])));



}

tabelaO = sortHashMapByValues(tabela, false);

/*Iterator iterator = tabelaO.keySet().iterator();

    while (iterator.hasNext()) {
      Integer key = (Integer) iterator.next();
 System.out.println(" ID " + key + " Valor " + tabelaO.get(key));
    }*/

//os 10 primeiros

Set entries = tabelaO.entrySet();
Iterator it = entries.iterator();
int count = 0;
while (it.hasNext()) {
    Map.Entry entry = (Map.Entry) it.next();
```

```
    dez.put((Integer)entry.getKey(),(Integer)entry.getValue());
    count++;
    if (count == 10)
        break;
}

//System.out.println(dez);

entries = dez.entrySet();
it = entries.iterator();
while (it.hasNext()) {
    Map.Entry entry = (Map.Entry) it.next();
    soma = soma + (Integer)entry.getValue();
    media = (soma/10.0);
}

List lista = new ArrayList(dez.values());
double array[] = new double[10];
//System.out.println(lista.size());

int j =0;
it = lista.iterator();
while(it.hasNext()) {
    //System.out.println(it.next());
    array[j] = Double.parseDouble(it.next().toString());
    j++;
}
//qualquer coisa mal aqui ..
//System.out.println(" d" + array[6]);
```

```
        desvio = getDesvioPadrao(array);
        //System.out.println(soma);
        //System.out.println(media);
        //System.out.println("desvio " + desvio);

        /*pw.println(dez);
        pw.println("Soma total: " + soma);
        pw.println("Media: " + media);
        pw.println("Desvio padrão: " + desvio);

        pw.close();*/

        Label label = new Label(0, c, dez.toString());
        sheet.addCell(label);

        Number number = new Number(1, c, soma);
        sheet.addCell(number);

        Number number1 = new Number(2, c, media);
        sheet.addCell(number1);

        Number number2 = new Number(3, c, desvio);
        sheet.addCell(number2);


    }



public static LinkedHashMap sortHashMapByValues(HashMap passedMap, boolean
```

```
  List mapKeys = new ArrayList(passedMap.keySet());
List mapValues = new ArrayList(passedMap.values());
Collections.sort(mapValues);
Collections.sort(mapKeys);

if (!ascending)
Collections.reverse(mapValues);

LinkedHashMap someMap = new LinkedHashMap();
Iterator valueIt = mapValues.iterator();
while (valueIt.hasNext()) {
Object val = valueIt.next();
Iterator keyIt = mapKeys.iterator();
while (keyIt.hasNext()) {
Object key = keyIt.next();
if (passedMap.get(key).toString().equals(val.toString())) {
passedMap.remove(key);
mapKeys.remove(key);
someMap.put(key, val);
break;
}
}
}
return someMap;
}


/*public static double getDesvio(Hashtable<Integer, Integer> t, int soma, double media
```

```
    double desvio = 0.0;
    double variancia = 0.0;

    Set entries = t.entrySet();
    Iterator it = entries.iterator();
    while (it.hasNext()) {
        Map.Entry entry = (Map.Entry) it.next();
    variancia = variancia + Math.pow(((Integer)entry.getValue() - media), 2);
    }

    variancia = variancia*(1.0/10.0);

    System.out.println("var " + variancia);

    Math.sqrt(variancia);



    return desvio;
}*/

public static double getSomaDosElementos(double array[]) {

        double total = 0;

        for (int counter = 0; counter < array.length; counter++)

            total += array[counter];

        return total;
```

```
    }

  public static double getVariancia(double array[]) {

        double p1 = 1 / Double.valueOf(array.length - 1);

        double p2 = getSomaDosElementosAoQuadrado(array)

              - (Math.pow(getSomaDosElementos(array), 2) / Double

                    .valueOf(array.length));

        return p1 * p2;

    }

public static double getSomaDosElementosAoQuadrado(double array[]) {

        double total = 0;

        for (int counter = 0; counter < array.length; counter++)

            total += Math.pow(array[counter], 2);

        return total;

    }
```

```
    // Desvio Padrão Amostral

    public static double getDesvioPadrao(double array[]) {

        return Math.sqrt(getVariancia(array));

    }




}
```

5.10 processaDados

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Arrays;


public class processaDados {
```

```
public static void main(String[] args) throws IOException{

    for (int i=1; i<11; i++){
        processarDados("nctrer\\cv" + i, "nctrer\\nctrer");
        processarDados("cpdbas\\cv" + i, "cpdbas\\cpdbas");
        processarDados("dbpcan\\cv" + i, "dbpcan\\dbpcan");
        //processarDados("epafhm\\cv" + i, "epafhm\\epafhm");
    }



}

static void processarDados(String entrada, String entrada2) throws IOException{


   BufferedReader brEnt = new BufferedReader(new FileReader(entrada2.toUpperCase
      "_temp.smi"));

   BufferedReader br1 = new BufferedReader(new FileReader(entrada + ".f"));
   BufferedReader br2 = new BufferedReader(new FileReader(entrada + ".n"));

      // 20079,0

      File finale = new File(entrada + "_FINAL.smi");
      //File final1 = new File(entrada + "_toxicos.smi");
      //File final2 = new File(entrada + "_naotoxicos.smi");

      /*if(!final1.exists()){
          final1.createNewFile();
```

```
}
if(!final2.exists()){
    final2.createNewFile();
}*/

//FileWriter fw1 = new FileWriter(final1);
//PrintWriter pw1 = new PrintWriter(fw1);

FileWriter fw = new FileWriter(finale);
PrintWriter pw = new PrintWriter(fw);



String linha = "";
String linhaE = "";
String[] linha2 = null;
String[] linhaE2 = null;
String id = "";



while ((linhaE = br1.readLine()) != null){

    String newLinha = linhaE.replace("(", "");
    linhaE2 = newLinha.split("active");
    newLinha = newLinha = linhaE2[1].replace(")", "");
    id = newLinha.replace(".", "");
    //System.out.println(Arrays.toString(linhaE2));
    //System.out.println(id);
brEnt = new BufferedReader(new FileReader(entrada2.toUpperCase() +
     "_temp.smi"));
    while((linha = brEnt.readLine()) != null){
```

```
        //System.out.println(linha);
        linha2 = linha.split(",");
        //System.out.println(linha2[0]);
        //System.out.println(id);
        //System.out.println(linha2.length);
        if (linha2[0].equalsIgnoreCase(id)){

            pw.println(id + "," + "0" + "," + linha2[1]);
        }

    }
    brEnt.close();

}

//pw.close();

//FileWriter fw2 = new FileWriter(final2);
//PrintWriter pw2 = new PrintWriter(fw2);


linha = "";
linhaE = "";
linha2 = null;
linhaE2 = null;
id = "";

while ((linhaE = br2.readLine()) != null){

    String newLinha = linhaE.replace("(", "");
```

```
            linhaE2 = newLinha.split("active");
            newLinha = newLinha = linhaE2[1].replace(")", "");
            id = newLinha.replace(".", "");

        brEnt = new BufferedReader(new FileReader(entrada2.toUpperCase() +
            "_temp.smi"));
            while((linha = brEnt.readLine()) != null){
                //System.out.println(linha);

                linha2 = linha.split(",");
                //System.out.println(linha2.length);
                if (linha2[0].equalsIgnoreCase(id))
                    pw.println(id + "," + "1" + "," + linha2[1]);

            }
            brEnt.close();

        }

        pw.close();


    }

}
```

5.11 Executar2

```
import java.io.IOException;



public class Main {


    public static void main(String[] args) throws IOException {
        // TODO code application logic here

        for (int i=1; i<11; i++){
            // set do directorio
        Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -s8 -S4 dbpcan\\cv" + i
        Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -s7 -S6 cpdbas\\cv" + i
        Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -s14 nctrer\\cv" + i + "

Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -s8 -S4 dbpcan\\cv" + i + "_l
        Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -s7 -S6 cpdbas\\cv" + i
        Runtime.getRuntime().exec("java -cp moss.jar moss.Miner -s14 nctrer\\cv" + i + "

        }

    }


}
```

5.12 Processa

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import jxl.Workbook;
import jxl.write.Label;
import jxl.write.WritableSheet;
import jxl.write.WritableWorkbook;
import jxl.write.WriteException;
import jxl.write.Number;


public class Processa {

     static int count = 1;

    public static void main(String[] args) throws FileNotFoundException,
     IOException, WriteException {

       WritableWorkbook workbook = Workbook.createWorkbook(new File("frequenc


            WritableSheet sheet1 = workbook.createSheet("CPDBAS", 0);

              Label label1 = new Label(0, 0, "Subestrutura");
              sheet1.addCell(label1);
           Label label2 = new Label(1, 0, "Contagem no Resultado Total");
              sheet1.addCell(label2);
```

```
Label label3 = new Label(2, 0, "Contagem no Resultado");
sheet1.addCell(label3);
Label label4 = new Label(3, 0, "Diferença");
sheet1.addCell(label4);
Label label13 = new Label(4, 0, "DiferençaC");
sheet1.addCell(label13);

Label labelA = new Label(6, 0, "Diferença");
sheet1.addCell(labelA);
Label labelB = new Label(7, 0, "DiferençaC");
sheet1.addCell(labelB);


for (int i=1; i<11; i++){
   processaDados("cpdbas_R\\res" + i, sheet1);
   count++;
}
```

```
WritableSheet sheet2 = workbook.createSheet("DBPCAN", 1);

Label label5 = new Label(0, 0, "Subestrutura");
sheet2.addCell(label5);
Label label6 = new Label(1, 0, "Contagem no Resultado Total");
sheet2.addCell(label6);
Label label7 = new Label(2, 0, "Contagem no Resultado");
sheet2.addCell(label7);
Label label8 = new Label(3, 0, "Diferença");
```

```
sheet2.addCell(label8);
Label label14 = new Label(4, 0, "DiferençaC");
sheet2.addCell(label14);

Label labelC = new Label(6, 0, "rel_s1");
sheet1.addCell(labelC);
Label labelD = new Label(7, 0, "rel_s2");
sheet1.addCell(labelD);

Label labelE1 = new Label(6, 0, "rel_c1");
sheet1.addCell(labelE1);
Label labelF1 = new Label(7, 0, "rel_c2");
sheet1.addCell(labelF1);

count = 1;
for (int i=1; i<11; i++){
   processaDados("dbpcan_R\\res" + i, sheet2);
   count++;
}

WritableSheet sheet3 = workbook.createSheet("NCTRER", 2);

Label label9 = new Label(0, 0, "Subestrutura");
sheet3.addCell(label9);
Label label10 = new Label(1, 0, "Contagem no Resultado Total");
sheet3.addCell(label10);
Label label11 = new Label(2, 0, "Contagem no Resultado");
sheet3.addCell(label11);
Label label12 = new Label(3, 0, "Diferença");
sheet3.addCell(label12);
```

```
        Label label15 = new Label(4, 0, "DiferençaC");
        sheet3.addCell(label15);

        Label labelE = new Label(6, 0, "Diferença");
        sheet1.addCell(labelE);
        Label labelF = new Label(7, 0, "DiferençaC");
        sheet1.addCell(labelF);

        count = 1;
        for (int i=1; i<11; i++){
           processaDados("nctrer_R\\res" + i, sheet3);
           count++;
        }


        workbook.write();
        workbook.close();

}

public static void processaDados(String f, WritableSheet sheet)
throws FileNotFoundException, IOException, WriteException{

 BufferedReader br = new BufferedReader(new FileReader(f + "T.sub"));
 BufferedReader br2 = new BufferedReader(new FileReader(f + ".sub"));

    String linha = "";
    String[] linhaSP = null;

    String linha2 = "";
```

```
String[] linhaSP2 = null;

br.readLine();
br2.readLine();

// id,description,nodes,edges,s_abs,s_rel,c_abs,c_rel
// [0], [1],        [2],  [3],  [4] , [5] , [6] , [7]

while ((linha = br.readLine()) != null){
   linhaSP = linha.split(",");

   Label labelx = new Label(0, count, linhaSP[1]);
   sheet.addCell(labelx);

Number number = new Number(1, count, Integer.parseInt(linhaSP[4]));
   sheet.addCell(number);

   br2 = new BufferedReader(new FileReader(f + ".sub"));
   br2.readLine();
   while ((linha2 = br2.readLine()) != null){
      linhaSP2 = linha2.split(",");
     //System.out.println(linhaSP[1] + " ### " + linhaSP2[1]);
      if (linhaSP[1].equals(linhaSP2[1])){
          //System.out.println("Encontrei!!!!!!");
    Number number1 = new Number(2, count, Integer.parseInt(linhaSP2[4]));
          sheet.addCell(number1);

          Number number2 = new Number(3, count,
    (Integer.parseInt(linhaSP[4])-Integer.parseInt(linhaSP2[4])));
          sheet.addCell(number2);
```

```
                Number number3 = new Number(4, count,
            (Integer.parseInt(linhaSP[6])-Integer.parseInt(linhaSP2[6])));
                sheet.addCell(number3);



                Label number4 = new Label(6, count, linhaSP[5]);
                sheet.addCell(number4);
                Label number5 = new Label(7, count, linhaSP2[5]);
                sheet.addCell(number5);

                Label number6 = new Label(9, count, linhaSP[7]);
                sheet.addCell(number6);
                Label number7 = new Label(10, count, linhaSP2[7]);
                sheet.addCell(number7);



            }

        }
        br2.close();
        count++;

    }

}

}
```

5.13 Convertor

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;


public class Convertor {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException {
        // TODO code application logic here

        //processaDados("NCTRER.sdf");
        processaDados("CPDBAS.sdf");
        //processaDados("DBPCAN.sdf");
        //processaDados("EPAFHM.sdf");

    }


    static void processaDados(String entrada) throws IOException{
```

```
String[] ent = entrada.split(".sdf");
File saida = new File(ent[0] + ".ve");


if(!saida.exists()){
    saida.createNewFile();
}


FileWriter fw = new FileWriter(saida);
PrintWriter pw = new PrintWriter(fw);

pw.println("t # 0");

BufferedReader br = new BufferedReader(new FileReader(entrada));

String linha = "";
String[] linha2 = null;
int m = 0;
int nV = 0;
int nE = 0;
int cV = 0;
int c = -1;
int c2 = -1;

while((linha = br.readLine()) != null){
    linha2 = linha.split("\\s+");
    System.out.println(java.util.Arrays.toString(linha2));
```

```
//System.out.println(linha2[0]);

if (m == 4){
    //linha2 = linha.split(" ");
    //System.out.println("e " + Integer.parseInt(linha2[1]) +
     " " + Integer.parseInt(linha2[2]) + " " + linha2[3]);
    if (nE == 0){
        m=0;
        break;
    }
    pw.println("e " + Integer.parseInt(linha2[1]) + " " +
    Integer.parseInt(linha2[2]) + " " + linha2[3]);
    c2--;
    if (c2 ==0)
        m=0;
}


else if (m==3){
    //linha2 = linha.split(" ");
    System.out.println("v " + cV + " " + linha2[4]);
    pw.println("v " + cV + " " + linha2[4]);
    cV++;
    c--;
    if (c == 0){
        m=4;
        c2=nE;
    }

}
```

```
    else if (m == 2){
        //linha2 = linha.split(" ");
        nV = Integer.parseInt(linha2[1]);
        nE = Integer.parseInt(linha2[2]);
        c = nV;
        //System.out.println(nV + " " + nE);
        m=3;
        if (nV == 0)
            m = 0;
    }

    else if (m == 1)
        m = 2;



    if(linha2.length > 1){
        if (linha2[1].equalsIgnoreCase("Marvin")){
            m = 1;
            //System.out.println(linha2.length);
        }
        //System.out.println(linha2.length);
        //System.out.println(linha2[0]);

    }

}

pw.close();
```

```
        }


    }
```