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

Modelos Computacionais Realistas para Dependências entre Entidades de Software

Rodrigo Rocha Gomes e Souza

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

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Prof. Dr. Dalton Dario Serey Guerrero (Orientador)
Prof. Dr. Jorge César Abrantes de Figueiredo (Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRAFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

S729m Souza, Rodrigo Rocha Gomes e.
Modelos computacionais realistas para dependências entre entidades de software / Rodrigo Rocha Gomes e Souza. — Campina Grande, 2010.
67 f.: il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Orientadores: Prof. Dr. Dalton Dario Serey Guerrero, Prof. Dr. Jorge César Abrantes de Figueiredo.
Referências.

1. Manutenção de Software. 2. Engenharia Reversa. 3. Redes Complexas. 4. Modelagem Computacional. 5. Estudos Experimentais. 6. Dependências. 7. Análise Estática. I. Título.

CDU – 004.416(043)

"MODELOS COMPUTACIONAIS REALISTAS PARA DEPENDÊNCIAS ENTRE ENTIDADES DE SOFTWARE"

RODRIGO ROCHA GOMES E SOUZA

DISSERTAÇÃO APROVADA EM 31.08.2010

DALTON DARIO SEREY GUERRERO, D.Sc
Orientador(a)

JORGE CESAR ABRANTES DE FIGUEIREDO, D.Sc
Orientador(a)

RAQUEL VIGOLVINO LOPES, D.Sc
Examinador(a)

MARCO TÚLIO DE OLIVEIRA VALENTE, Dr.
Examinador(a)

CAMPINA GRANDE - PB

Resumo

A análise das dependências entre as entidades do código-fonte de um sistema de software é feita por diversas ferramentas de engenharia reversa com o propósito de revelar informações úteis para a manutenção do software. Existe, no entanto, uma carência de estudos experimentais projetados para avaliar tais ferramentas, em parte devido ao alto custo de se realizar experimentos na área.

Na área de redes e sistemas distribuídos, o alto custo de experimentação motiva o uso da simulação como meio para avaliar protocolos e algoritmos. Na engenharia reversa, no entanto, simulações são pouco exploradas — o que se explica parcialmente pela falta de modelos computacionais realistas para dependências entre entidades de código-fonte.

Neste trabalho são apresentados modelos computacionais que geram representações que podem ser interpretadas como dependências entre entidades de código-fonte. Um dos modelos computacionais, chamado BCR+, foi desenvolvido no contexto deste trabalho. Foi desenvolvido também um modelo de classificação que indica, com precisão de 96%, se uma representação de dependências é realista — isto é, se ela se assemelha a representações extraídas de sistemas reais.

Por fim, é apresentada uma prova de conceito, que demonstra a viabilidade do uso do modelo BCR+ na avaliação de algoritmos usados no contexto de recuperação de arquitetura de software, um ramo da engenharia reversa.

Abstract

The analysis of dependencies between source code entities of a software system is performed by several reverse engineering tools in order to reveal information that is useful for software maintenance. There is, however, a shortage of experimental studies designed to evaluate such tools, in part due to the high cost of conducting experiments in the area.

In the area of networks and distributed systems, the high cost of experimentation motivates the use of simulation as a means to evaluate protocols and algorithms. In reverse engineering, however, simulations are underexplored — which is partly explained by the lack of realistic computational models for dependencies between source code entities.

This paper presents computer models that generate representations which can be interpreted as dependencies between source code entities. One of the models, called BCR+, was developed in the context of this work. We have also developed a classification model that can tell, with accuracy of 96%, whether a representation of dependencies is realistic — that is, if it resembles representations extracted from real systems.

This work also presents a proof of concept, demonstrating the feasibility of using BCR+ to evaluate algorithms used in the context of software architecture recovery, a branch of reverse engineering.

Agradecimentos

Muito obrigado à minha família, que me ajudou de inúmeras maneiras em todos os momentos deste empreendimento acadêmico: meu pai, Renato, minha mãe, Ligia, e meu irmão, Henrique; minha esposa, Denise, meu sogro, Ângelo, e minha sogra, Fátima.

Obrigado à minha orientadora de graduação, Christina, que me apoiou antes, durante e depois do meu mestrado, de incontáveis formas.

Obrigado a Maria da Guia e Oscar, por nos acolherem (a mim e a Denise) sobretudo na nossa primeira visita a Campina Grande.

Obrigado aos meus orientadores, Dalton e Jorge, que me propuseram como tema de mestrado um desafio empolgante, e que sempre me motivaram a fazer o meu melhor.

Obrigado ao pessoal do Grupo de Métodos Formais (GMF) da UFCG, por servirem de inspiração para o meu trabalho, em especial a Roberto, a quem considero meu co-orientador. Obrigado aos colegas da UFCG, pela companhia, em especial a Dalton Cézane, Isabel, Stéfani e Rafael. Muito obrigado a Anne Caroline e a Bruno, pela grande amizade. Obrigado a Lilian do GMF e a Aninha da COPIN, pela atenção.

Obrigado a Lemuel e aos colegas do Coro em Canto, responsáveis por muitas das boas lembranças que tenho de Campina Grande. Obrigado a Marta pela amizade e ajuda.

Obrigado à CAPES, pelo apoio financeiro no primeiro ano, sob a forma de bolsa de mestrado. Obrigado à Fundação de Amparo à Pesquisa do Estado da Bahia (Fapesb) — em especial, Sandra (coordenadora de TI) e Dora (então diretora geral) —, pelo apoio financeiro no segundo ano do mestrado, através de um emprego flexível como analista de sistemas na Coordenação de TI. Obrigado aos colegas da Fapesb pela convivência agradável. Obrigado ao Doutorado Multi-Institucional em Ciência da Computação (DMCC) e à Fapesb, pelo apoio financeiro nos últimos seis meses, desta vez na forma de uma bolsa de doutorado. Obrigado à Apple, pelo apoio financeiro nos últimos seis meses, através da venda dos aplicativos da RoDen Apps para iOS (iPhone, iPod touch e iPad).

Obrigado a Fabíola e a Ítalo, que me permitiram rodar os experimentos do mestrado no *cluster* do Grupo de Algoritmos e Computação Distribuída (Gaudi) da UFBA.

Obrigado a todos os que acompanharam e opinaram sobre meu trabalho em pelo menos um momento: os professores do Laboratório de Engenharia de Software (LES) da UFBA;

Charles e Garcia, do grupo de Física Estatística e Sistemas Complexos (FESC) da UFBA; a professora Rose, do Departamento de Estatística da UFBA; Lancichinetti e Gail Murphy, pelo *feedback* via e-mail, e Nenad Medvidović, pelo *feedback* nos momentos finais; e Fubica, Eustáquio, Raquel e Marco Túlio, que participaram de bancas de avaliação ao longo do desenvolvimento deste trabalho e contribuíram com observações valiosas.

Obrigado a Deus por ter colocado tantas pessoas maravilhosas no meu caminho.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contexto

Para um sistema de software ser bem sucedido não basta ele ser rápido, funcional e isento de defeitos. Esses atributos dizem respeito à qualidade, da forma como é enxergada pelos seus usuários, de uma versão do sistema. Para atender a demandas emergentes de seus usuários e incorporar novidades tecnológicas, no entanto, um sistema de software deve apresentar certos atributos de qualidade interna, visíveis apenas para os seus desenvolvedores. Ele precisa ser fácil de compreender, fácil de modificar e fácil de testar [1].

Um bom indicador da qualidade interna de um sistema de software é a estrutura de dependências entre as entidades que compõem o seu código-fonte. Considera-se como dependência qualquer referência que uma entidade, em seu código-fonte, faça a outra entidade. É natural que existam interações entre diversas entidades em um sistema, mas dependências indesejadas adicionam complexidade ao software, tornando mais difíceis tarefas ligadas a sua manutenção, como compreender ou testar isoladamente as suas diversas partes [2].

Analisar uma a uma as dependências de um sistema de software com a finalidade de extrair informações que apoiem a sua manutenção é, no entanto, uma tarefa árdua [3]. Em um sistema com n entidades, existem cerca de n^2 potenciais dependências a se analisar. Em sistemas com mais do que algumas dezenas de entidades, a quantidade de dependências potenciais é da ordem de centenas ou milhares.

Face à dificuldade de se analisar manualmente as dependências de um sistema, surgiram ferramentas de engenharia reversa voltadas para a inferência automática — sem interferência

humana — de informações que apoiam tarefas específicas de manutenção. Existem, por exemplo, ferramentas que procuram identificar partes do sistema afetadas por uma mudança [4], código duplicado [5] ou módulos arquiteturais [6].

As tarefas que as ferramentas buscam automatizar, entretanto, envolvem alguma subjetividade. O conceito de módulo arquitetural, por exemplo, não é bem definido. Mesmo dois especialistas, ao analisar independentemente um sistema de software, dificilmente identificam o mesmo conjunto de módulos arquiteturais [7].

Dada a subjetividade das tarefas, a avaliação empírica de ferramentas de engenharia reversa usualmente requer a presença de programadores que tenham experiência nas tarefas que as ferramentas se propõem a cumprir, além de familiaridade com as ferramentas avaliadas. Tais requisitos elevam o custo envolvido na realização de experimentos.

Como resultado, a frequência com que as avaliações empíricas são realizadas está aquém do que se espera de uma área de pesquisa madura. Um estudo realizado com artigos sobre engenharia reversa publicados de 2002 a 2005 revelou que 25% dos artigos não apresentam qualquer forma de avaliação empírica e, dentre os demais artigos, apenas 30% recorrem a experimentos e estudos observacionais — os outros 70% recorrem a métodos menos rigorosos, como estudos de caso e relatos de experiência [3].

Uma abordagem empregada quando os experimentos controlados são caros é a simulação de modelos computacionais, por tratar-se de um meio para se realizar experimentos controlados a um custo baixo. Essa abordagem tem sido usada em diversas áreas, como física, economia, biologia, economia, engenharia, redes de computadores etc. Na área de redes e sistemas distribuídos, por exemplo, é frequente o uso de ambientes de simulação de redes, que modelam o funcionamento de uma rede de computadores, incluindo, às vezes, modelos de falhas de hardware e até modelos de comportamento de usuários [8].

Naturalmente, há o risco de os modelos usados na simulação não corresponderem à realidade nos aspectos relevantes para o experimento. Todavia, espera-se que esse risco seja reduzido à medida que os modelos são testados e refinados.

Na engenharia de software, a abordagem de modelagem e simulação é pouco explorada. Durante esta pesquisa, foram encontrados alguns trabalhos que aplicam a simulação no estudo de processos de desenvolvimento de software [9]. Não foi encontrado, no entanto, nenhum trabalho na área de engenharia reversa com foco na modelagem computacional de

dependências.

1.2 Problema

A avaliação empírica tradicional de ferramentas de engenharia reversa baseadas na análise de dependências requer a colaboração de programadores experientes e, por isso, é custosa. O custo elevado é um problema porque contribui para que o número de pesquisas experimentais na área seja reduzido.

Abordagens experimentais com custo mais baixo poderiam contribuir para aumentar o volume de pesquisas experimentais em engenharia reversa. Como resultado, os efeitos positivos e negativos de cada ferramenta seriam melhor compreendidos, o que colaboraria para o aprimoramento das ferramentas existentes.

1.3 Solução Proposta

A solução proposta nesta pesquisa para o problema do alto custo é usar modelos computacionais de dependências para avaliar ferramentas de engenharia reversa. Os modelos são algoritmos que recebem parâmetros e, como saída, geram grafos que representam dependências. Se protocolos de rede podem ser avaliados em experimentos simulados com modelos de redes de computadores, por que não avaliar ferramentas de engenharia reversa com base em dependências construídas a partir da simulação controlada de um modelo de dependências? A solução apresenta pelo menos duas vantagens em relação à abordagem tradicional de avaliação:

- o custo é mais baixo;
- o controle das variáveis independentes é maior, uma vez que modelos computacionais possuem parâmetros que podem ser ajustados.

Dito isso, o foco desta pesquisa não é a avaliação de ferramentas de engenharia reversa — embora uma avaliação do gênero seja apresentada como prova de conceito. O foco desta pesquisa é a avaliação de modelos computacionais que podem apoiar a avaliação de ferramentas de engenharia reversa.

Para simplificar as discussões, consideremos algumas definições. As dependências de um sistema de software são representados sob a forma de *rede* (para efeitos práticos, o mesmo que *grafo*). Assim, os modelos computacionais devem ser *modelos de redes*. Chamaremos de *rede de software* o conjunto de entidades e dependências entre entidades extraídos de um sistema de software. Redes que se assemelham a redes de software serão chamadas de redes *software-realistas*.

Com essas definições, a questão de pesquisa pode ser enunciada de forma sucinta: *existem modelos de redes software-realistas?*

1.4 Métodos

A abordagem empregada nesta pesquisa pode ser desdobrada nos seguintes passos:

1. descobrir modelos de redes, os quais podem ser interpretados como modelos de dependências entre entidades de código-fonte;

2. desenvolver e avaliar um modelo de classificação que determina se uma rede é ou não software-realista;

3. avaliar, através do modelo de classificação, se os modelos produzem redes software-realistas;

4. realizar uma prova de conceito para demonstrar a viabilidade da simulação de modelos de redes na avaliação de técnicas e ferramentas que se baseiam na análise de dependências.

Os métodos escolhidos para lidar com cada um dos passos são apresentados a seguir.

A teoria das redes complexas estuda métodos para analisar redes (ou grafos) encontradas nos mais diversos domínios, como redes sociais, redes de computadores e redes metabólicas. A aplicação de tais métodos levou à descoberta de propriedades comuns a um grande número de redes, bem como modelos de redes que incorporam essas propriedades.

Dado que as dependências em um sistema de software formam uma rede, optou-se por utilizar nesta pesquisa alguns dos métodos e descobertas da teoria das redes complexas. Em

particular, decidiu-se investigar modelos computacionais de redes propostos na literatura sobre redes complexas.

Para determinar se uma rede é ou não software-realista, optou-se por uma abordagem baseada na disciplina de aprendizagem de máquina, na qual o conceito de software-realismo é induzido a partir de exemplos. Para auxiliar tal abordagem, decidiu-se investigar métricas estudadas na teoria de redes complexas para comparação de redes.

Para a prova de conceito foi escolhido o problema de recuperação de arquitetura de software, devido à familiaridade do autor com o problema. Optou-se por aplicar algoritmos de agrupamento usados na atividade de recuperação de arquitetura a redes de dependências geradas por um dos modelos computacionais e então analisar os dados com ferramentas estatísticas.

1.5 Resultados

Foram encontrados na literatura diversos modelos de redes, os quais podem ser interpretados como modelos de dependências entre entidades de software. Além disso, um modelo de redes organizadas em módulos, o BCR+, foi desenvolvido no contexto desta pesquisa.

Foi desenvolvido um modelo de classificação que, dada uma rede, a classifica em software-realista ou não-software realista. O modelo de classificação foi avaliado com um conjunto composto de redes de software e de outros domínios e apresentou taxa de acerto de cerca de 98%.

O modelo de classificação foi aplicado a redes geradas com diversos modelos de redes e, como resultado, constatou-se que o modelo BCR+ e outros dois produzem tanto redes software-realistas quanto redes não software-realistas.

Como prova de conceito foram realizados, a partir da simulação do modelo BCR+, três experimentos com a finalidade de compreender e comparar algoritmos usados no contexto da atividade de recuperação de arquitetura. Os experimentos mostram que a abordagem de avaliação usando modelos é viável e complementa a abordagem tradicional.

Como resultado concreto de pesquisa, foi publicado um artigo na 14ª Conferência Europeia sobre Manutenção de Software e Reengenharia (CSMR 2010). O artigo foi intitulado "Modular Network Models for Class Dependencies in Software" e apresentou a avaliação de

três modelos de redes.

1.6 Estrutura do Documento

No Capítulo 2 são apresentados métodos, modelos e descobertas da teoria das redes complexas. Uma métrica de similaridade entre redes é exposta. Os modelos ER, BA, CGW e LFR são descritos.

No Capítulo 3 é detalhado o modelo BCR+, com descrição algorítmica e exemplos.

No Capítulo 4 é apresentado e avaliado um modelo de classificação que indica se uma dada rede é semelhante a redes de dependências extraídas de sistemas de software.

No Capítulo 5 é relatada a avaliação dos modelos de redes quanto ao software-realismo das redes que eles geram.

No Capítulo 6 é feita uma introdução ao problema de recuperação de arquitetura e, a seguir, são apresentados os métodos e resultados da prova de conceito envolvendo o problema de recuperação de arquitetura e o modelo BCR+.

No Capítulo 7 este trabalho é comparado a outros trabalhos.

No Capítulo 8 são apresentadas algumas conclusões, contribuições e limitações deste trabalho.

Capítulo 2

Redes Complexas

2.1 Introdução

A teoria das redes complexas estuda propriedades gerais de diversos tipos de redes, representadas como grafos, com o uso de ferramentas estatísticas. Estudos realizados na última década revelaram similaridades entre redes estudadas em diversos domínios. Exemplos incluem redes tecnológicas, como a Web e a rede de distribuição de energia elétrica dos Estados Unidos, redes biológicas, como cadeias alimentares e redes de ligações entre proteínas, e redes sociais, como as relações de amizade entre alunos de uma escola [10].

O termo "rede" em geral está associado a entidades reais, como pessoas e relacionamentos de amizade, enquanto o termo "grafo" designa uma abstração matemática conveniente para representar relacionamentos entre objetos. Na teoria das redes complexas, no entanto, os termos são frequentemente usados como sinônimos, e é desta forma que eles são usados neste trabalho.

2.2 Definições Básicas

Uma rede ou grafo é um conjunto de *vértices* e *arestas*, no qual as arestas relacionam dois vértices. Pode-se dizer também que as arestas ligam ou conectam dois vértices. Uma rede pode ser *orientada* ou *não-orientada*.

Nas redes orientadas, as arestas (chamadas de arestas orientadas) estabelecem uma relação assimétrica entre dois vértices, um dos quais é dito vértice de origem e o outro, vértice

de destino da aresta. Duas arestas são ditas *opostas* se ligam o mesmo par de vértices, porém com origem e destino invertidos.

Em uma rede orientada, o *grau de saída* de um vértice x, denotado $\mathrm{g_{out}}(x)$, é o número de arestas com origem no vértice x. O *grau de entrada* de um vértice x, denotado $\mathrm{g_{in}}(x)$, é o número de arestas com destino no vértice x. O *grau total* de um vértice x é igual à soma do seu grau de entrada com o seu grau de saída.

Nas redes não-orientadas, a relação entre pares de vértices é simétrica: cada aresta (chamada de aresta não-orientada) liga dois vértices, e não existe distinção entre origem e destino. Em uma rede não-orientada, o *grau* de um vértice x, denotado $\mathrm{g}(x)$, é o número de arestas que se relaciona com x. Para transformar uma rede não-orientada em uma rede orientada, basta transformar cada aresta não-orientada em duas arestas orientadas opostas que ligam o mesmo par de vértices que a aresta não-orientada.

Nas redes organizadas em módulos (que podem ser orientadas ou não-orientadas), o conjunto de vértices é particionado em subconjuntos denominados *módulos*. As arestas podem ser classificadas em *internas* (quando ligam vértices pertencentes ao mesmo módulo) ou *externas* (quando ligam vértices pertencentes a módulos distintos).

As redes podem ser representadas graficamente por diagramas com círculos e linhas. Cada vértice é representado por um círculo. Nas redes orientadas, as arestas são representadas por setas que ligam o vértice de origem ao vértice de destino. Duas arestas opostas podem ser representadas de forma simplificada por uma seta que aponta simultaneamente para dois vértices. Nas redes não-orientadas, as arestas são representadas por linhas que conectam dois vértices (sem seta). Os módulos, quando existem, são representados por retângulos ou formas livres que circundam os vértices que contêm.

2.3 Propriedades Estatísticas

Barabási e Albert [11] analisaram uma amostra da *World Wide Web*, modelada como um grafo não-orientado no qual os vértices representam páginas e as arestas representam *links* entre duas páginas. Eles observaram a distribuição dos graus dos vértices, isto é, o número de vértices conectados a outros k vértices ($N(k)$), para cada valor de $k > 0$, e encontraram uma lei de potência, isto é, acharam $N(k)$ proporcional a $k^{-\gamma}$, como mostra a Figura 2.1. Desde

então, leis de potência têm sido encontradas na distribuição de graus de redes estudadas em diversos domínios, inclusive no domínio de software, com γ variando tipicamente entre 2 e 3. Redes com esse padrão são chamadas de *redes livres de escala*.

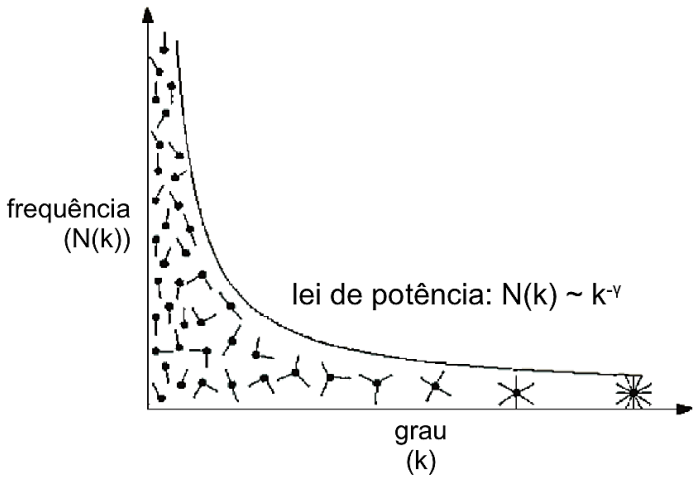

Figura 2.1: Distribuição do número de arestas por vértice do tipo lei de potência. Adaptado de [12].

Se diversas redes possuem um mesmo padrão de distribuição de graus, o que as diferencia? Milo e outros pesquisadores [13] estudaram a estrutura de redes orientadas de diversos domínios em busca da resposta. Para isso, eles listaram 13 possíveis configurações de arestas em redes com 3 vértices — as chamadas tríades —, mostradas na Figura 2.2. Contando o número de vezes que cada tríade aparece em uma rede, é possível formar um vetor, denominado *perfil de concentração de tríades* (PCT), que é característico de redes de um domínio.

O papel dos PCTs na caracterização de domínios de redes é ilustrado nas Figuras 2.3(a) e 2.3(b). Na Figura 2.3(a), são apresentados PCTs de redes de dois domínios distintos: uma rede de software e uma rede linguística. Na Figura 2.3(b), são apresentados PCTs de duas redes do mesmo domínio, o domínio de software. Uma análise informal dos gráficos revela que a similaridade entre os PCTs é maior no segundo caso, no qual as redes são do mesmo domínio. No primeiro caso, é notável a diferença nas concentrações das duas primeiras tríades (de cima para baixo).

A similaridade entre PCTs pode ser quantificada através do coeficiente de correlação de Pearson entre os PCTs [15]. O resultado é um valor entre -1,0 (menor similaridade) e 1,0 (maior similaridade). Na Figura 2.3(a), o coeficiente de correlação vale $0{,}68$; na Figura

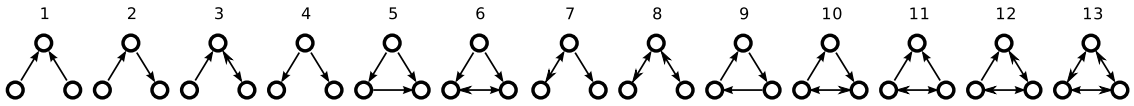

Figura 2.2: Tríades, ou grafos com três vértices, numeradas de 1 a 13.

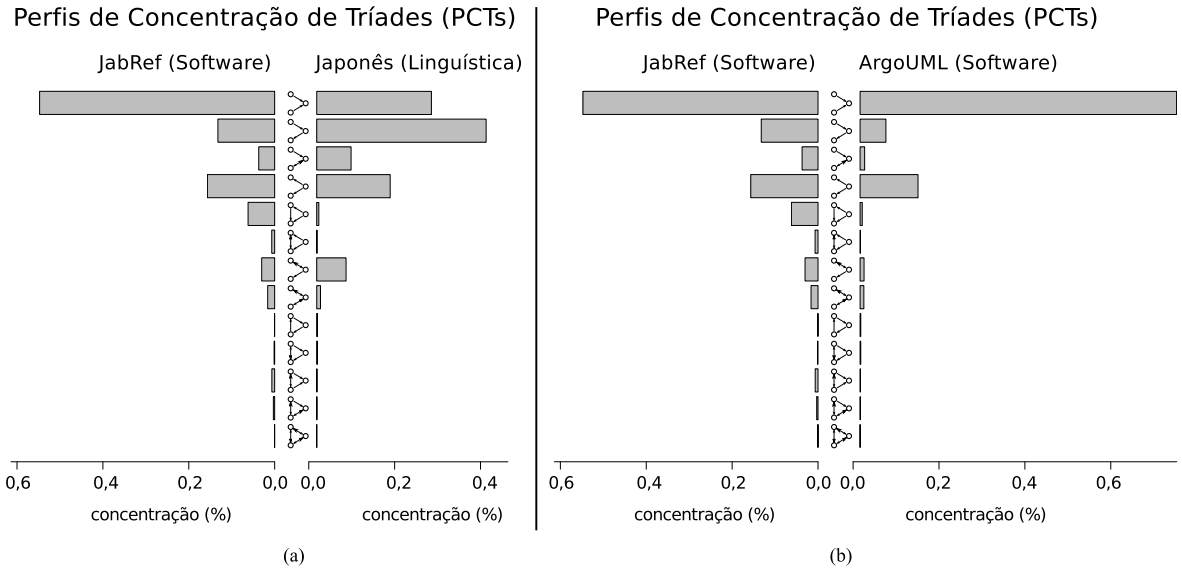

Figura 2.3: Comparação entre perfis de concentração de tríades de três redes distintas, extraídos pelo autor através da ferramenta igraph [14]. (a) À esquerda, rede de dependências entre as classes do programa JabRef, versão 2.5b2; à direita, rede de adjacências entre palavras da língua japonesa [15]. (b) À esquerda, a rede do programa JabRef, versão 2.5b2; à direita, a rede do programa ArgoUML, versão 0.28.

2.3(b), $0{,}98$. Os números confirmam a análise informal da Figura 2.3 e mostram que, no exemplo, a correlação é maior no caso em que as redes pertencem ao mesmo domínio.

Cabe aqui uma ressalva: os PCTs em geral não apresentam um conjunto de valores que seguem a distribuição normal, o que é um pré-requisito para se aplicar a correlação de Pearson. Curiosamente, não foi encontrado nenhum trabalho na literatura que chamasse atenção para esse ponto, tampouco que propusesse alternativas à correlação de Pearson.

2.4 Redes de Dependências no Domínio de Software

Redes são muito usadas no domínio de engenharia de software para representar as dependências entre entidades do código-fonte, tais como classes em linguagens orientadas a objetos. Estudos recentes têm aplicado a teoria das redes complexas a redes de software, isto é, redes de dependências entre entidades do código-fonte de sistemas de software. Um dos principais resultados é a constatação de que redes de software são livres de escala, isto é, as dependências entre as entidades da implementação dos sistemas de software se distribuem de acordo com uma lei de potência.

Valverde e Solé [16] estudaram redes não-orientadas formadas por relações de agregação de tipos em diagramas UML, programas em C e programas em C++. Myers [17] analisou redes de chamadas de função em programas em C e redes de agregação e herança em programas em C++. Em ambos os casos as redes foram identificadas como livres de escala.

Redes livres de escala também foram encontradas em programas escritos em Smalltalk [18, 19] e em Java [20–22], em dependências entre pacotes de software [23], em dependências entre bibliotecas dinâmicas [24] e até mesmo em referências entre objetos em tempo de execução [25].

2.5 Modelos de Geração de Redes

Para tentar explicar os mecanismos responsáveis pela formação de redes livres de escala, vários modelos de redes livres de escala foram propostos. Os modelos são algoritmos que geram vértices e arestas de forma probabilística porém de acordo com certas regras que garantem que, quando o número de vértices tende a infinito, a distribuição dos graus dos

vértices tende a uma lei de potência. Tais modelos, portanto, geram redes similares a redes de software, ao menos quanto à distribuição dos graus.

A seguir são apresentados quatro modelos de redes. Os dois primeiros, o de Erdős–Rényi (ER) [26] e o de Barabási-Albert (BA) [11], geram redes sem módulos. Eles foram selecionados devido a sua importância histórica. Os dois últimos, o modelo CGW [27] e o modelo LFR [28, 29], foram selecionados por gerarem redes organizadas em módulos. Acredita-se que tal característica os aproxima do processo de construção de sistemas de software, no qual o conceito de módulo desempenha um papel importante.

2.5.1 O modelo de Erdős–Rényi (ER)

O modelo de Erdős–Rényi (ER) [26] precedeu a teoria das redes complexas. Ele gera redes não-orientadas que, em geral, não são livres de escala. A distribuição esperada dos graus dos vértices é a distribuição de Poisson. O modelo ER recebe dois parâmetros:

- n, o número de vértices;
- m, o número de arestas.

Uma rede não-orientada com n vértices pode conter até $\frac{n(n-1)}{2}$ arestas. Nas redes geradas pelo modelo ER, m arestas são selecionadas aleatoriamente dentre as arestas potenciais. Cada aresta potencial tem a mesma probabilidade de ser selecionada.

Nada impede que o modelo seja usado para gerar redes orientadas. Na variedade orientada do modelo ER, considera-se que uma rede com n vértices pode conter até $n(n-1)$ arestas orientadas, e desse conjunto são selecionadas as m arestas.

Pela descrição do modelo, percebe-se que ele pode gerar qualquer rede possível, com qualquer distribuição de graus. A probabilidade de uma rede gerada pelo modelo ER ser livre de escala é, no entanto, baixíssima.

2.5.2 O modelo de Barabási-Albert (BA)

O modelo de Barabási-Albert (BA) [11] foi o primeiro modelo livre de escala da teoria das redes complexas. Ele gera redes não-orientadas, livres de escala e sem módulos através de dois mecanismos: crescimento e ligação preferencial. Crescimento significa que as redes são

construídas a partir da adição sucessiva de novos vértices. Ligação preferencial significa que os vértices com mais arestas apresentam maior probabilidade de receber novas arestas.

O modelo BA aceita os seguinte parâmetros:

- n, o número de vértices da rede final;
- m, o número de arestas adicionadas a cada passo.

A rede é inicializada com um número arbitrário de vértices e arestas de forma que cada vértice possua grau maior ou igual a 1. A cada passo, é adicionado um novo vértice, que é ligado através de m novas arestas a m vértices pré-existentes. Os m vértices são escolhidos de acordo com os seus graus, o que significa que a probabilidade de um vértice x ser escolhido, $\mathrm{P}(x)$, é proporcional ao grau de x: $\mathrm{P}(x) \sim \mathrm{g}(x)$. Como a soma dos probabilidades dos vértices deve ser igual a 1, a probabilidade de cada vértice é igual ao seu grau dividido pela soma dos graus de todos os vértices da rede:

$$\mathrm{P}(x) \;=\; \frac{\mathrm{g}(x)}{\sum_y \mathrm{g}(y)}.$$

Diz-se que os novos vértices se ligam *preferencialmente* a vértices com alto grau. Como consequência, alguns poucos vértices acumulam muitas arestas, enquanto a maioria dos vértices permanece com poucas arestas. O processo é repetido até que a rede atinja n vértices.

2.5.3 O modelo CGW

O modelo CGW [27] gera redes orientadas, livres de escala e organizadas em módulos. Baseado no modelo BA, ele utiliza os mecanismos de crescimento e ligação preferencial. Ele foi proposto como um modelo da evolução de sistemas de software. O modelo aceita 11 parâmetros:

- número de vértices, n;
- número de módulos, m;
- quatro probabilidades, p_1, p_2, p_3, p_4, com $p_1 + p_2 + p_3 + p_4 = 1$ e $p_1 > 0$;
- quatro números naturais, e_1, e_2, e_3, e_4;

- uma constante, α, com $\alpha \geq -1$.

Nesse modelo, a construção inicia-se com uma rede inicial arbitrária e então vai crescendo de acordo com determinadas regras de formação, até alcançar n vértices. Cada vértice é atribuído a exatamente um dos m módulos no momento em que é criado.

A rede inicial é alterada pela aplicação sucessiva de quatro regras em ordem aleatória:

- Regra 1: com probabilidade p_1, um novo vértice é adicionado a um módulo escolhido aleatoriamente, juntamente com e_1 arestas com origem no novo vértice. Os vértices de destino das e_1 arestas são escolhidos de acordo com a probabilidade preferencial baseada em módulos (PPBM), explicada mais à frente.

- Regra 2: com probabilidade p_2, são adicionadas e_2 arestas. Para cada aresta, o vértice de origem é escolhido aleatoriamente, enquanto o vértice de destino é escolhido de acordo com a PPBM.

- Regra 3: com probabilidade p_3, e_3 arestas são religadas. O procedimento de religamento de arestas é descrito a seguir:

  1. um vértice, v_1 é escolhido aleatoriamente;
  2. uma aresta, a_1, escolhida aleatoriamente dentre as arestas com origem em v_1, é removida da rede;
  3. é adicionada uma nova aresta cuja origem é v_1 e o vértice de destino é escolhido de acordo com a PPBM;

- Regra 4: com probabilidade p_4, e_4 arestas escolhidas aleatoriamente são removidas da rede.

Naturalmente, as probabilidades p_1, p_2, p_3 e p_4 devem somar 1. Além disso, p_1 deve ser maior que zero — do contrário o número de vértices na rede permanece constante. As quantidades e_1, e_2, e_3, e e_4 são inteiros maiores ou iguais a zero.

A probabilidade preferencial baseada em módulos, $\Pi(v_2|v_1)$, é uma função que indica a probabilidade de se escolher um vértice, v_2, como destino de uma aresta cujo vértice de origem, v_1, já foi determinado. O propósito da PPBM é controlar a proporção de arestas

externas na rede, privilegiando a escolha de um vértice de destino pertencente ao mesmo módulo do vértice de origem. Eis a definição da probabilidade preferencial baseada em módulos:

$$\Pi(v_2|v_1) \;=\; \begin{cases} \dfrac{1+\mathrm{g}(v_2)\cdot(1+\alpha)}{Q(v_1)}, & \text{se } v_2 \text{ está no mesmo módulo de } v_1; \\ \dfrac{1+\mathrm{g}(v_2)}{Q(v_1)}, & \text{caso contrário.} \end{cases}$$

A seguir é explicado o significado de α, g(v) e Q(v).

O parâmetro α controla a proporção de arestas externas na rede. Para $\alpha = -1$, a maioria das arestas serão externas. Para $\alpha > 0$, a maioria das arestas serão internas, e quanto maior o valor de α, maior a tendência. Quando $\alpha = 0$, arestas internas e externas são igualmente prováveis.

A expressão g(v) designa o grau de saída do vértice v. O termo Q é apenas uma constante de proporcionalidade cujo propósito é fazer a soma das probabilidades ser igual a 1, e é definido da seguinte forma:

$$Q(v_1) = \sum_{v \in m(v_1)} (1+\mathrm{g}(v)\cdot(1+\alpha)) \;+\; \sum_{v \notin m(v_1)} (1+\mathrm{g}(v))$$

A expressão m(v), neste contexto, designa o conjunto dos vértices que pertencem ao mesmo módulo de v.

2.5.4 O modelo LFR

O modelo LFR [28, 29] é um modelo flexível que pode gerar redes com arestas ponderadas e módulos sobrepostos, isto é, nas quais um vértice pode pertencer a mais de um módulo. Diferentemente do CGW, o LFR não é um modelo de crescimento: todos os vértices são gerados de uma vez e então são adicionadas as arestas.

Nesta pesquisa foi estudado um caso particular do modelo no qual todas as arestas têm o mesmo peso e os módulos não se sobrepõem (cada vértice pertence a exatamente um módulo). O modelo aceita os seguintes parâmetros:

- número de vértices, n;

- grau de entrada médio, k, com $k < n$;
- grau de entrada máximo, max_k, com $k \leq max_k < n$;
- parâmetro de mistura, μ, com $0 \leq \mu \leq 1$;
- expoente da distribuição de graus, $-\gamma$;
- expoente da distribuição de tamanho de módulos, $-\beta$;
- tamanho do menor módulo, min_m;

Os tamanhos dos módulos são selecionados de uma lei de potência com expoente $-\beta$. O parâmetro de mistura, μ, é a proporção de arestas externas na rede gerada. No modelo LFR, nem todas as combinações de parâmetros são factíveis. Por exemplo, se $n = 100$, então min_m não pode ser 60, caso contrário existiriam módulos menores do que min_m.

2.6 Conclusão

A teoria das redes complexas é uma área de pesquisa que oferece ferramentas teóricas as quais apoiam o estudo de redes do ponto de vista estatístico. A teoria tem sido aplicada no estudo de diversos domínios, como a sociologia, a biologia e a engenharia.

Redes livres de escala são redes que possuem uma determinada distribuição de graus. Redes de dependências entre entidades de software já foram identificadas como redes livres de escala em diversos estudos recentes.

Perfis de concentração de tríades (PCTs) caracterizam redes através de vetores de treze números. Dois PCTs podem ser comparados através do coeficiente de correlação de Pearson.

ER, BA, CGW e LFR são modelos de redes. O modelo ER gera redes sem módulos que, em geral, não são livres de escala. O modelo BA gera redes não-orientadas, livres de escala e sem módulos. Os modelos CGW e LFR geram redes orientadas, livres de escala e organizadas em módulos.

Capítulo 3

BCR+: Um Novo Modelo de Redes Organizadas em Módulos

3.1 Introdução

No capítulo anterior foram apresentados quatro modelos que geram redes. Três dos modelos geram redes que, tais quais redes de software, tendem a ser livres de escala. Destes, dois geram redes que, assim como sistemas de software, são organizadas em módulos. Falta a esses modelos, no entanto, um maior controle sobre as relações entre os módulos — algo que está presente no modelo proposto neste capítulo, o BCR+.

Durante o projeto de um sistema de software complexo, é comum estabelecer quais são os módulos do sistema e de que forma eles se relacionam. Em sistemas estritamente estruturados em camadas, por exemplo, cada módulo pode depender de, no máximo, um outro módulo. Diz-se que um módulo depende de outro se as entidades do primeiro dependem das entidades do segundo. Restringir as dependências permitidas entre módulos pode favorecer a portabilidade e o reuso de um sistema.

Nos modelos CGW e LFR, é possível controlar a proporção de arestas entre vértices pertencentes a módulos distintos, mas não é possível restringir as dependências entre módulos. Nas redes geradas por esse modelos, cada vértice pode se ligar a qualquer outro, independentemente dos módulos aos quais os vértices pertencem. As dependências entre os módulos não podem ser especificadas *a priori*.

Para suprir essa lacuna nos modelos apresentados, propomos, neste capítulo, um novo

modelo de redes organizadas em módulos, o BCR+. Assim como os modelo CGW o BCR+ gera — através dos mecanismo de crescimento e ligação preferencial — redes orientadas, livres de escala e organizadas em módulos. Diferentemente do que ocorre com o CGW, no entanto, com o novo modelo é possível controlar, a partir de um parâmetro, quais são as dependências permitidas entre módulos.

3.2 Descrição

O modelo BCR+ não foi criado do zero: ele é uma generalização de um modelo proposto por Bollobás e colegas [30], que gera redes orientadas, livres de escala e sem módulos. Optou-se neste trabalho por não descrever separadamente o modelo original. Em lugar disso, será descrito apenas o modelo BCR+, juntamente com indicações dos pontos em que os dois modelos diferem.

O modelo BCR+ aceita os seguintes parâmetros:

- número de vértices, n;
- três probabilidades, p_1, p_2 e p_3, com $p_1 + p_2 + p_3 = 1$;
- grau de entrada base, δ_{in};
- grau de saída base, δ_{out}.
- um grafo orientado de dependências entre módulos, G;
- uma constante μ, com $0{,}0 \leq \mu \leq 1{,}0$;

Os dois últimos parâmetros são exclusivos do BCR+ — eles não existem no modelo original.

O grafo G contém um vértice para cada módulo que será criado e define uma relação de dependência entre os módulos. Um módulo M_1 depende de um módulo M_2 se G contém uma aresta do vértice que representa M_1 para o vértice que representa M_2. Na rede gerada, uma aresta de um vértice $v_1 \in M_1$ para um vértice $v_2 \in M_2$ pode existir apenas se M_1 depende de M_2 no grafo G ou se M_1 e M_2 são o mesmo módulo.

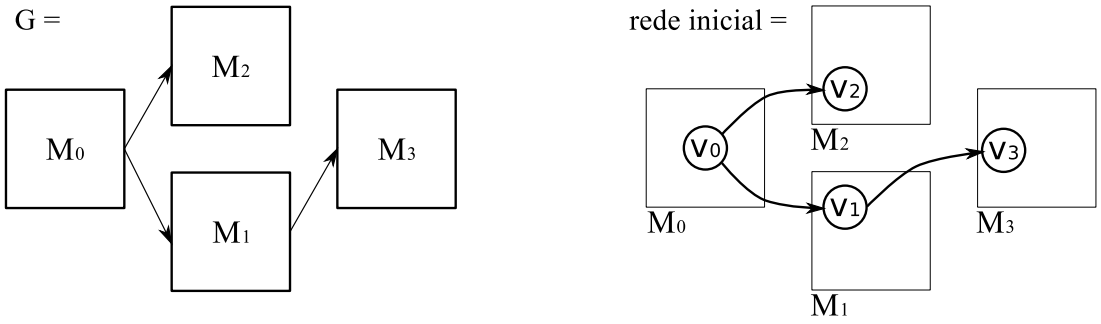

Figura 3.1: Rede inicial gerada a partir de um dado grafo G.

O parâmetro μ controla a proporção de arestas externas na rede — arestas que ligam vértices pertencentes a módulos distintos. Valores mais baixos levam a redes com menos arestas externas.

O modelo original de Bollobás e colegas [30] é um caso específico do modelo BCR+ quando $\mu = 0$ e G consiste de apenas um vértice, o qual representa um único módulo. Com essa informação é fácil derivar a descrição do modelo original a partir da descrição do modelo BCR+ apresentada a seguir.

Na rede inicial gerada pelo modelo BCR+, cada módulo contém exatamente um vértice e todas as arestas externas permitidas são adicionadas. A Figura 3.1 ilustra a geração de uma rede inicial a partir de um grafo G. Essa rede é então modificada de acordo com 3 regras de formação que são aplicadas sucessivamente, em ordem aleatória, até a rede alcançar o número de n vértices. A cada passo do algoritmo, a probabilidade de se aplicar a regra i é p_i.

3.3 Regras de Formação

Nas regras apresentadas a seguir, a expressão "escolher um vértice de acordo com f(x)", onde f(x) é uma função qualquer, significa que a probabilidade de escolher um vértice x é dada pela seguinte função de probabilidade:

$$\mathrm{P}(x) = \frac{\mathrm{f}(x)}{\sum_i \mathrm{f}(i)}$$

O denominador é apenas um fator de normalização, responsável por tornar a soma das probabilidades igual a 1.

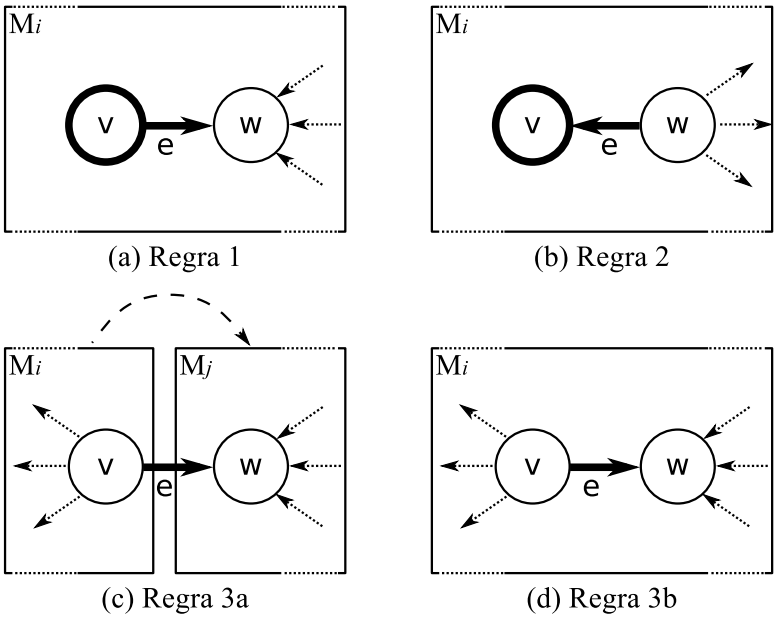

Figura 3.2: As regras de formação de redes do modelo BCR+. M_i e M_j são módulos, e M_i depende de M_j. Os vértices e arestas criados durante a execução de cada regra estão destacados com uma linha mais grossa.

As regras do modelo estão ilustradas na Figura 3.2 e são descritas a seguir:

1. *Adicionar um vértice com uma aresta de saída*. Um vértice existente, w, é escolhido de acordo com $\mathrm{f}(x) = \delta_{in} + \mathrm{g}_{in}(x)$ (parâmetro δ_{in} somado ao grau de entrada do vértice). Um novo vértice, v, é adicionado ao módulo que contém w, juntamente com uma aresta de v a w.

2. *Adicionar um vértice com uma aresta de entrada*. Um vértice existente, w, é escolhido de acordo com $\mathrm{f}(x) = \delta_{out} + \mathrm{g}_{out}(x)$. Um novo vértice, v, é adicionado ao módulo que contém w, juntamente com uma aresta de w a v.

3. *Adicionar uma aresta entre vértices pré-existentes*. Um vértice, v, é escolhido de acordo com $\mathrm{f}(x) = \delta_{out} + \mathrm{g}_{out}(x)$. Então é adicionada uma aresta do vértice v a um vértice, w, escolhido de acordo com $\mathrm{f}(x) = \delta_{in} + \mathrm{g}_{in}(x)$, respeitando um dos seguintes casos:

   (a) com probabilidade μ, w é escolhido dentre os vértices que estão em módulos dos quais o módulo de v depende, segundo o parâmetro G;

(b) com probabilidade $1 - \mu$, w é escolhido dentre os vértices que estão no mesmo módulo que v.

No modelo original proposto por Bollobás e colegas [30], não há módulos ou, equivalentemente, há apenas um módulo. Por esta razão, a distinção entre as regras 3(a) e 3(b) não existe no modelo original.

3.4 Exemplo

A Figura 3.3 mostra um passo da simulação do modelo BCR+, continuando a rede inicial mostrada na Figura 3.1. Suponhamos que o parâmetro δ_{out} vale 2 e que, neste passo da simulação, a regra 2 foi escolhida. Neste caso, é necessário escolher um vértice da rede, e essa escolha deve considerar, para cada vértice, x, o valor de $\mathrm{f}(x) = \delta_{out} + \mathrm{g}_{out}(x)$, ilustrado na rede inicial da figura através de números entre parênteses. A probabilidade de se escolher um vértice x, P(x) é, então, dada por $\mathrm{P}(x) = \frac{\mathrm{f}(x)}{\sum_i \mathrm{f}(i)}$, onde o denominador, no exemplo, é igual a 11, de forma que $\sum \mathrm{P}(x)$ = 1.

A Figura 3.3 ilustra, através de uma reta real, as probabilidades associadas à escolha cada vértice. Cada vértice ocupa, na reta, um segmento cujo tamanho é proporcional à probabilidade de ele ser escolhido. Assim, a escolha do vértice pode ser feita sorteando-se um número real entre 0 e 1 e marcando-o na reta. No exemplo, foi sorteado o número $\frac{5}{11}$, que corresponde a um ponto do segmento do vértice v_1 na reta. Dando seguimento à regra 2, é criado um novo vértice, v_4, que é atribuído ao módulo M_1 (o mesmo módulo ao qual pertence o vértice v_1), e ligado ao vértice v_1 através de uma aresta com origem em v_1.

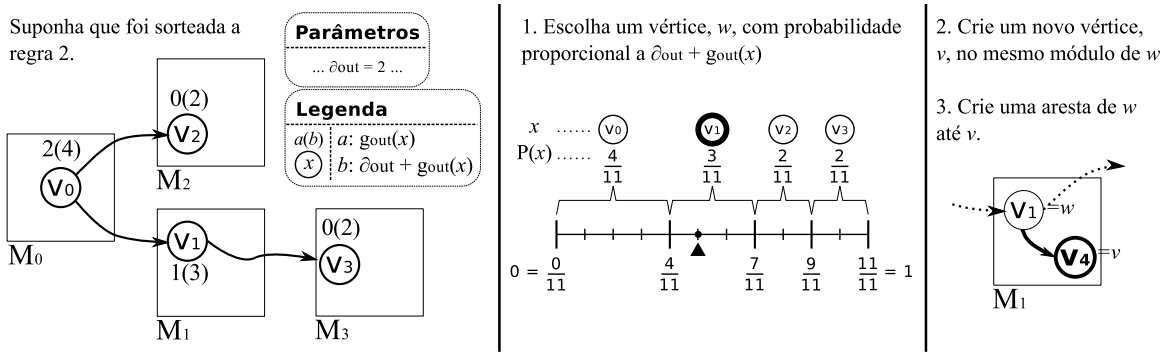

Figura 3.3: Um passo da simulação do modelo BCR+.

Nota-se que vértices com grau de saída alto têm mais chance de ser a origem de novas arestas. O parâmetro δ_{out} reduz essa tendência fornecendo um grau de entrada base que é aplicado a todos os vértices no cálculo das probabilidades. No exemplo da Figura 3.3, se δ_{out} fosse igual a zero, os vértices v_2 e v_3 teriam chance nula na escolha, uma vez que o grau de saída de ambos é igual a zero. O mesmo raciocínio se aplica a δ_{in} com relação ao grau de entrada.

3.5 Comparação com Outros Modelos

A principal diferença do BCR+ em relação aos modelos apresentados anteriormente é a possibilidade de restringir as dependências entre módulos. A Tabela 3.1 apresenta um sumário de características do BCR+ e dos demais modelos a fim de possibilitar uma comparação mais completa.

Tabela 3.1: Características do modelo BCR+ comparadas a características de outros modelos.

Característica vs. Modelo	**ER**	**BA**	**CGW**	**LFR**	**BCR+**
Número de parâmetros	2	2	11	7	8
Gera redes com módulos?	✗	✗	✔	✔	✔
Gera redes orientadas?	✗*	✗	✔	✔	✔
Gera redes livres de escala?	✗	✔	✔	✔	✔
Usa o mecanismo de crescimento?	✗	✔	✔	✗	✔
Restringe dependências entre módulos?	✗	✗	✗	✗	✔

* Existem variações do modelo ER que geram redes orientadas.

Dentre os modelos que geram redes orientadas, livres de escala e organizadas em módulos, o BCR+ é um dos modelos mais simples em relação ao número de parâmetros, possuindo 8 parâmetros. Nesse quesito ele perde apenas para o modelo LFR, que aceita 7 parâmetros. De acordo com a tabela, o modelo BCR+ é o mais completo, sendo o único que permite ao usuário restringir as dependências entre módulos.

3.6 Conclusão

O modelo BCR+ gera redes livres de escala orientadas e organizadas em módulos. Em relação a modelos similares, o modelo BCR+ tem a vantagem de oferecer controle sobre as dependências que podem ser geradas entre entidades que pertencem a módulos distintos. Tal possibilidade contribui para que o modelo BCR+ reproduza melhor o processo de construção de sistemas de software que se baseiam na implementação de uma arquitetura pré-definida.

Capítulo 4

Um Modelo de Classificação de Redes

4.1 Introdução

Redes software-realistas são redes que se assemelham a redes de software (redes de dependências extraídas de sistemas de software). Para avaliar se um modelo de redes gera redes software-realistas, é preciso encontrar um *modelo de classificação* que, dada uma rede, determine se ela é ou não software-realista. É importante, ainda, avaliar se o modelo de classificação apresenta alto desempenho, medido, por exemplo, pela taxa de acerto.

Este capítulo apresenta um modelo de classificação de redes com alto desempenho que foi construído de acordo com a abordagem de aprendizagem de máquina. Nessa abordagem, o modelo de classificação é induzido por um algoritmo — denominado *classificador* — a partir de um conjunto contendo redes software-realistas e redes não software-realistas.

Este capítulo está organizado da seguinte forma:

- na Seção 4.2 são introduzidos alguns conceitos pertinentes ao problema de classificação;
- na Seção 4.3 é introduzida uma métrica denominada grau de software-realismo;
- na Seção 4.4 é descrito um classificador que induz modelos de classificação de redes usando o grau de software-realismo das redes;
- na Seção 4.5 é apresentado um estudo com a finalidade de estimar o desempenho dos modelos induzidos pelo classificador;

- na Seção 4.6 é mostrado um modelo de classificação induzido pelo classificador;
- na Seção 4.7 os resultados mais importantes são recapitulados.

4.2 Conceitos de Classificação

De forma geral, o problema de classificação consiste em atribuir objetos a classes pré-determinadas a partir da análise de atributos dos objetos [31]. Um modelo de classificação nada mais é do que um conjunto de regras ou um algoritmo que, com base nos atributos de um objeto, procura acertar a classe à qual o objeto pertence.

Por exemplo, é possível dividir os animais em duas classes: mamíferos e não-mamíferos. Um possível modelo de classificação para animais seria a seguinte regra: se o animal não põe ovos, ele é mamífero; caso contrário, ele é não-mamífero. Tal modelo, embora apresente uma alta taxa de acerto, não é perfeito. O modelo erra quando classifica o animal ornitorrinco como não-mamífero. Apesar de pôr ovos, o ornitorrinco é classificado pela comunidade científica como um animal mamífero.

Na disciplina de aprendizagem de máquina, o problema de classificação é resolvido através da escolha de um *classificador* — um algoritmo que induz um modelo de classificação a partir de um conjunto que contém exemplos de objetos pertencentes a todas as classes relevantes, denominado *conjunto de treinamento*. O classificador busca criar um modelo de classificação que apresente alta taxa de acerto quando aplicada ao conjunto de treinamento.

Mais importante, no entanto, é que o modelo seja generalizável, isto é, apresente alta taxa de acerto ao tentar prever a classe de objetos que não foram usados em seu treinamento. Para avaliar se um modelo é generalizável, é selecionado um *conjunto de teste*, que contém objetos cujas classes são conhecidas e que não foram usados no treinamento. O modelo induzido a partir do conjunto de treinamento é então aplicado ao conjunto de teste, e a taxa de acerto é calculada comparando-se as classes reais dos objetos do conjunto de teste com as classes determinadas pelo modelo.

4.3 Grau de Software-Realismo

Nesta seção é apresentada uma métrica que indica, em uma escala contínua, o quanto uma rede se assemelha a redes de software.

Sejam a e b duas redes, PCT(x) o vetor com as concentrações das tríades na rede x e cor(x, y) o coeficiente de correlação de Pearson entre dois vetores. A similaridade entre as redes, sim(a, b), é dada por

$$\mathrm{sim}(a, b) \;=\; \mathrm{cor}(\mathrm{PCT}(a), \mathrm{PCT}(b)).$$

O grau de software-realismo de uma rede x com base em um conjunto de redes de referência, R, é dado por:

$$\mathrm{S}(x, R) \;=\; \frac{\sum_{s \in R} \mathrm{sim}(x, s)}{|R|}.$$

O conjunto R é um conjunto arbitrário contendo redes que são consideradas software-realistas. Redes de software (redes extraídas de sistemas de software reais) são boas candidatas a redes do conjunto R.

Cabem algumas ressalvadas à fórmula apresentada para o grau de software-realismo. Primeiro, os valores nos perfis de concentração de tríades não possuem uma distribuição normal, o que viola um pré-requisito da correlação de Pearson. Essa questão é relevante e vale a pena investigar em trabalhos futuros métricas que estejam de acordo com a teoria estatística.

Além disso, o leitor mais cético terá dificuldades em acreditar que os 13 valores dos perfis sejam suficientes para medir a similaridade entre redes com centenas ou milhares de vértices. Argumentamos que o mérito da métrica aqui apresentada pode ser julgado pelos resultados que ela produz. De fato, como veremos a seguir, a métrica fornece, em geral, valores altos para redes de software e valores baixos para redes de outros domínios, com poucas exceções. Com esses resultados, aumenta a confiança de que a métrica pode realmente distinguir entre redes de diferentes domínios.

4.4 Um Classificador para Modelos de Classificação de Redes

Nesta seção é descrito um classificador que induz modelos de classificação de redes. Primeiramente é definida a forma geral dos modelos induzidos pelo classificador. A seguir é descrito o classificador proposto.

4.4.1 Forma Geral dos Modelos de Classificação

Propomos que os modelos de classificação de redes sejam da forma $\mathrm{m}(x, R, S_0)$, onde x é a rede a ser classificada, R é um conjunto de redes consideradas software-realistas, e S_0 é um número real entre -1,0 e 1,0:

$$\mathrm{m}(x, R, S_0) \;=\; \begin{cases} \text{software-realista}, & \text{se } \mathrm{S}(x, R) \geq S_0; \\ \text{não software-realista}, & \text{caso contrário.} \end{cases}$$

A função $\mathrm{S}(x, R)$ representa o grau de software-realismo de uma rede, x, tomando como referência o conjunto R. O valor S_0 é chamado de limiar de software-realismo. Quando o grau de software-realismo da rede x supera o limiar, a rede é classificada como software-realista; nos demais casos, a rede é classificada como não software-realista.

Em suma, o modelo de classificação prevê que uma rede é software-realista somente se a similaridade média entre a rede e as redes de um conjunto pré-determinado de redes software-realistas é superior a um valor pré-estabelecido.

A escolha do limiar afeta diretamente a taxa de acerto do modelo de classificação. Quando o limiar é muito baixo, a taxa de acerto diminui, pois muitas redes passam a ser classificadas como software-realistas, inclusive redes que não o são. Quando o limiar é muito alto, a taxa de acerto também é baixa, pois neste caso muitas redes software-realistas são classificadas como não software-realistas. Assim, a escolha de um valor adequado para o limiar é fundamental para se obter um modelo de classificação com alta taxa de acerto.

4.4.2 Definição do Classificador

Um modelo de classificação da forma $\mathrm{m}(x, R, S_0)$ pode ser induzido por um classificador que recebe um conjunto de treinamento composto de dois subconjuntos: o conjunto T, contendo apenas redes consideradas software-realistas, e o conjunto $\bar{T}$, contendo apenas redes consideradas não software-realistas. A partir destes conjuntos, o classificador determina valores para os parâmetros R e S_0.

O processo de determinação de R e S_0 é detalhado no Algoritmo 1. Em poucas palavras, R é sempre determinado como sendo igual a T, e o valor de S_0 é selecionado dentre os graus de realismo das redes do conjunto de treinamento, $T \cup \bar{T}$. Em particular, é selecionado o valor que maximiza a taxa de acerto do modelo de classificação $\mathrm{m}(x, R, S_0)$.

Para exemplificar o funcionamento do classificador, considere um conjunto de treinamento composto de duas redes software-realistas (conjunto T) e duas redes não software-realistas (conjunto $\bar{T}$). O grau de software-realismo de cada rede é calculado tomando como referência o conjunto T. Considere que as redes do conjunto T possuem grau de software-realismo iguais a 0,9 e 0,7, e que as redes do conjunto $\bar{T}$ possuem grau de software-realismo iguais a -0,3 e 0,6.

O algoritmo de treinamento considera cada valor de grau de software-realismo como candidato a limiar, juntamente o número de acertos correspondentes. São considerados, portanto, valores de S_0 dentro do conjunto $\{-0{,}3; 0{,}6; 0{,}7; 0{,}9\}$. Se $S_0 = -0{,}3$, por exemplo, todas as redes do conjunto de treinamento são classificadas como software-realistas, resultando em 2 acertos (taxa de acerto de 50%). Se $S_0 = 0{,}6$ ou se $S_0 = 0{,}9$, o modelo apresenta 3 acertos (taxa de acerto de 75%). O valor escolhido para o limiar é, portanto, o valor $0{,}7$, que resulta em 4 acertos (taxa de acerto de 100%). O modelo de classificação induzido pelo conjunto de treinamento fica da seguinte forma:

$$\mathrm{m}(x) = \begin{cases} \text{software-realista}, & \text{se } \mathrm{S}(x, \mathrm{T}) \geq 0{,}7; \\ \text{não software-realista}, & \text{caso contrário.} \end{cases}$$

Algoritmo 1 Algoritmo que determina os parâmetros R e S_0 de um modelo de classificação da forma m(x, R, S_0) a partir dos conjuntos T e $\bar{T}$

$maxAcertos \leftarrow 0$

$melhorLimiar \leftarrow 0$

para cada $x \in (T \cup \bar{T})$ **faça**

$s \leftarrow \mathrm{S}(x, T)$

{Conta o número de acertos quando $S_0 = s$}

$acertos \leftarrow 0$

para cada $y \in (T \cup \bar{T})$ **e** $y \neq x$ **faça**

se $(S(y, T) \geq s$ **e** $y \in T)$ **ou** $(S(y, T) < s$ **e** $y \notin T)$ **então**

$acertos \leftarrow acertos + 1$

fim se

fim para

{Atualiza o melhor limiar encontrado até então}

se $acertos > maxAcertos$ **então**

$maxAcertos \leftarrow acertos$

$melhorLimiar \leftarrow s$

fim se

fim para

$R \leftarrow T$

$S_0 \leftarrow melhorLimiar$

retorna R, S_0

4.5 Avaliação do Classificador

Antes de usar o classificador em uma aplicação prática, é importante avaliar se, com um conjunto de treinamento real, ele induz modelos de classificação com bom desempenho. O desempenho de um modelo é medido a partir de sua aplicação a um conjunto de teste, disjunto do conjunto de treinamento. Três métricas são comumente usadas para avaliar o desempenho de um classificador: acurácia, precisão e cobertura.

A *acurácia*, ou taxa de acerto, é a proporção das redes do conjunto de teste que são classificadas corretamente pelo modelo. A *precisão* é a proporção das redes classificadas pelo modelo como software-realistas que de fato são software-realistas. A *cobertura* é a proporção das redes software-realistas do conjunto de teste que são corretamente classificadas pelo modelo.

Nesta seção, o desempenho do classificador apresentado na seção anterior é avaliado com base em um conjunto de redes reais. A avaliação é realizada em 3 etapas:

- *coleta de dados*: são coletados sistemas de software e redes estudadas em diversos domínios;
- *pré-processamento dos dados*: redes de dependência são extraídas dos sistemas de software coletados;
- *validação*: o conjunto de dados é repartido de diferentes formas entre conjunto de treinamento e conjunto de testes, induzindo assim diversos modelos de classificação para os quais as métricas de desempenho são calculadas.

4.5.1 Coleta de Dados: Redes e Sistemas

Foram coletadas 66 redes de domínios tão diversos quanto a biologia, a sociologia, a tecnologia e a linguística, com tamanho variando entre 32 e 18.163 vértices. As redes foram obtidas em *websites* de pesquisadores da área de redes complexas. A lista completa de redes, com referências, pode ser encontrada no Apêndice A. Apenas redes orientadas foram selecionadas para o estudo, uma vez que as redes de software são redes orientadas.

Também foram coletados 65 sistemas de software escritos em Java, contendo entre 63 e 6.433 classes ou interfaces cada um. Quase todos os sistemas foram selecionados a partir

da lista dos sistemas mais populares do repositório SourceForge.net, que abriga mais de 200 mil[1] projetos de código aberto; além destes, foi selecionado o sistema OurGrid, desenvolvido na Universidade Federal de Campina Grande.

4.5.2 Pré-Processamento de Dados: Extração de Redes de Software

Os sistemas de software selecionados continham diversos arquivos no formato JAR (*Java Archive*), que contêm código-objeto de cada classe do sistema. Alguns arquivos JAR correspondiam a bibliotecas que são comumente usadas por diversos sistemas; esses arquivos foram removidos da análise para não influenciar os resultados. Foram mantidos apenas os arquivos JAR que continham código específico de cada sistema.

Como os sistemas de software foram coletados sob a forma de código-objeto, foi necessário extrair de cada um deles a sua rede de dependências, ou rede de software. A extração foi realizada através da ferramenta gratuita Dependency Finder[2], que extrai dependências a partir de código-objeto Java. A escolha se deveu à facilidade de uso via linha de comando e à sua robustez.

Nas redes extraídas, as entidades são classes e interfaces Java. As dependências representam qualquer referência de uma classe ou interface no código de outra classe ou interface, incluindo relacionamentos de herança, chamadas de método, instanciação, leitura ou escrita de atributos e agregação.

A lista completa dos sistemas de software pode ser encontrada no Apêndice A, juntamente com o número de vértices e arestas de cada rede de dependências correspondente.

A seguir foi calculado o grau de software-realismo de cada rede de software, comparando cada uma das 65 redes com as outras 64 redes. O valor mediano alcançado foi 0,91, e a diferença entre quartis (IQR, de *inter-quartile range*, uma medida de dispersão estatística) foi de apenas 0,04. O alto valor mediano e a baixa dispersão indicam que o grau de software-realismo caracteriza bem o conjunto de redes de software.

Quatro redes de software, no entanto, apresentaram grau de software-realismo inferior ao valor mediano menos $3 \times$ IQR, isto é, inferior a 0,79. Essas redes (com grau de software-realismo denotado entre parênteses) são as seguintes: battlefieldjava-0.1 (0,63), ec2-api-

[1] `http://sourceforge.net/about`

[2] `http://depfind.sf.net/`

tools-1.3-36506 (0,28), Hl7Comm.1.0.1 (0,09) e ourgrid-4.1.5 (0,72). Essas redes foram removidas do estudo, por não serem, do ponto de vista estrutural, representativas de redes de software.

4.5.3 Validação

A validação do classificador consiste em induzir modelos a partir de conjuntos de treinamento, aplicá-los a conjuntos de teste e então computar as métricas de desempenho. Os conjuntos de treinamento e de teste são subconjuntos do conjunto de redes que foram coletadas ou extraídas nas etapas anteriores, totalizando 127 redes.

Dado o número pequeno de redes disponíveis para teste e treinamento, optou-se por realizar a validação cruzada, na qual cada rede é usada exatamente uma vez para teste e um número fixo de vezes para treinamento. A validação cruzada permite que se obtenham estimativas confiáveis do desempenho do classificador mesmo que o conjunto de dados seja pequeno [32].

Na validação cruzada, o conjunto de redes é dividido em um número fixo, k, de partições (ou dobras) com tamanho aproximadamente igual. A atribuição de redes a partições é aleatória. Inicialmente, a primeira partição é escolhida como conjunto de teste, enquanto as demais formam o conjunto de treinamento. Então um modelo de classificação é induzido a partir do conjunto de treinamento e aplicado ao conjunto de teste. Nesse momento são calculadas as métricas de desempenho. O procedimento é executado k vezes, de forma que, a cada iteração, uma partição é escolhida para teste e as demais, para treinamento. Ao final, uma estimativa do valor das métricas de desempenho pode ser obtida através da média aritmética dos valores calculados a cada iteração.

Dois recursos podem ser empregados para obter uma estimativa mais confiável do desempenho (taxa de acerto) de um modelo de classificação: a estratificação e a repetição [32]. Na validação cruzada estratificada, a proporção das categorias nas partições reflete a proporção das categorias no conjunto completo. Por exemplo, se o conjunto de dados possui 40% de objetos de uma categoria e 60% de outra, procura-se manter essa proporção nas partições. A repetição consiste em repetir a validação diversas vezes (tipicamente dez vezes [32]), de forma a reduzir os efeitos aleatórios do particionamento do conjunto de redes em dobras.

Neste estudo foi utilizada a validação cruzada de 10 dobras estratificadas (*stratified 10-*

fold cross-validation) com 10 repetições, o que equivale a 100 modelos induzidos e, portanto, 100 medidas de acurácia, precisão e cobertura. Calculando a média, chegou-se às seguintes estimativas:

- acurácia: $98{,}0\%$
- precisão: $96{,}2\%$
- cobertura: $100{,}0\%$

Para todas as métricas foi encontrado um valor médio superior a 95%. Em particular, a cobertura foi de 100% em todos os testes, o que significa que todas as redes software-realistas foram corretamente classificadas.

Os resultados indicam que a métrica grau de software-realismo, na qual se baseia o classificador, é capaz de diferenciar redes de software e redes de outros domínios com alta precisão e alta cobertura.

4.6 Modelo de Classificação de Redes Induzido

Na seção anterior, foi estimado o desempenho de modelos induzidos a partir de conjuntos de treinamento com 90% das 127 redes selecionadas para o estudo (10% das redes foram separadas para teste). Usando-se 100% do conjunto de dados para treinamento, obtém-se um modelo de classificação cujo desempenho é no mínimo tão bom quanto o desempenho dos modelos induzidos na validação.

O modelo induzido a partir do conjunto de 127 redes apresenta $R = T$ e $S_0 = 0{,}799$. O modelo fica da seguinte forma:

$$\mathrm{m}(x) = \begin{cases} \text{software-realista}, & \text{se } \mathrm{S}(x, \mathrm{T}) \geq 0{,}799; \\ \text{não software-realista}, & \text{caso contrário}, \end{cases}$$

onde T é o conjunto de redes extraídas dos 61 sistemas de software coletados.

4.7 Conclusão

Neste capítulo foi apresentado um modelo de classificação que classifica redes em software-realistas — que se assemelham a redes de software — ou não software-realistas. O modelo foi induzido por um classificador a partir de um conjunto de treinamento formado por 61 redes de software — consideradas software-realistas — e 66 redes de outros domínios — consideradas não software-realistas. Uma avaliação usando o conjunto de 127 redes permitiu estimar que o modelo de classificação apresenta alta acurácia (cerca de 98%), alta precisão (cerca de 96%) e cobertura perfeita (100%).

Capítulo 5

Avaliação de Modelos de Redes

5.1 Introdução

Modelos de redes geram redes que, a princípio, podem ser usadas na avaliação de ferramentas de engenharia reversa. Antes de adotar um modelo de redes, no entanto, é importante avaliar se ele é capaz de gerar redes software-realistas — que se assemelham a redes de software. Em caso afirmativo, é conveniente entender como ajustar os parâmetros dos modelos para que a maioria das redes geradas sejam software-realistas.

Neste capítulo é descrito um experimento que mostra que três dos cinco modelos de redes apresentados anteriormente — BCR+, CGW e LFR — geram tanto redes software-realistas quanto redes não-software-realistas.

O experimento é dividido nas seguintes etapas:

- *seleção de parâmetros*: para cada modelo, são definidos centenas ou milhares de configurações de parâmetros, de forma a explorar a variedade de redes que podem ser geradas por cada modelo;
- *geração de redes*: cada configuração de parâmetros é usada para gerar pelo menos uma rede;
- *classificação das redes*: cada rede gerada é classificada pelo modelo de classificação como software-realista ou não-software-realista;

5.2 Seleção de Parâmetros

Nesta seção os valores selecionados para cada modelo são explicitados. A fim de gerar uma grande diversidade de redes, para cada parâmetro de cada modelo foram selecionados valores cobrindo, se possível, toda a extensão do domínio do parâmetro. Para os parâmetros cuja faixa de valores é ilimitada, foram escolhidos valores arbitrários, baseados na literatura ou em observações sobre redes de software.

Para o número de vértices foi fixado o valor arbitrário 1.000, que tem sido usado em outros estudos [33]. Fixar o número de vértices parece uma restrição razoável, uma vez que tal parâmetro afeta apenas o tamanho da rede gerada, e não o padrão de ligações entre vértices.

A arbitrariedade dos valores seria uma limitação séria do estudo se o seu objetivo fosse relacionar parâmetros a características das redes geradas pelos modelos. Este estudo, no entanto, tem caráter exploratório. O objetivo é variar valores de parâmetros e aferir se alguma configuração de parâmetros leva a uma rede software-realista. A arbitrariedade na escolha dos valores é, portanto, uma limitação menor, dado que, para cada modelo, o número de configurações selecionadas para os parâmetros é grande.

A seguir são descritas as configurações de parâmetros selecionadas para cada modelo. No total, quase 50.000 configurações foram selecionadas.

O Modelo ER

Para o modelo ER, foram selecionados para m (número de arestas) os valores de 2.000 a 10.000, com incrementos de 100, isto é, $m \in \{2.000, 2.100, 2.200, \ldots, 10.000\}$. Essa faixa de valores foi escolhida com base na análise das redes de software do Apêndice A. Nessas redes, o número de arestas é sempre de 2 a 10 vezes maior do que o número de vértices. Assim, consideramos valores de m que são de 2 a 10 vezes maior do que o valor de n (número de vértices). Além disso, consideramos duas variedades do modelo ER: o modelo original, que gera redes não-orientadas, e uma adaptação que gera redes orientadas. São 81 configurações de parâmetros por variedade, ou 162 configurações no total.

O Modelo BA

Para o modelo BA, foram selecionados valores de m no conjunto $2, 3, 4, 5, 6, 7, 8, 9, 10$, usando a mesma justificativa do parâmetro m do modelo ER. Isso representa 9 configurações de parâmetros.

O Modelo BCR+

Para o modelo BCR+, foram escolhidos grafos de módulos (parâmetro G) extraídos a partir de dependências entre arquivos JAR de 5 sistemas de software: GEF (2 módulos), iBATIS (4 módulos), MegaMek (8 módulos), findbugs (16 módulos) e zk (32 módulos). Como muitos dos arquivos JAR foram concebidos para serem reusados em projetos distintos, eles são uma boa aproximação do conceito de módulo. A escolha de grafos de módulos extraídos de sistemas de software reais foi guiada pela busca configurações de parâmetros que, intuitivamente, levassem a redes software-realistas. Essa escolha não teve como objetivo exaurir as possibilidades do parâmetro G.

Para os demais parâmetros, os seguintes valores foram escolhidos:

- $p_1, p_2, p_3 \in \{0{,}0; 0{,}2; 0{,}4; 0{,}6; 0{,}8; 1{,}0\}$, com $p_1 + p_2 + p_3 = 1$ e $p_1 + p_2 > 0$ (do contrário a rede jamais alcançaria 1.000 vértices);
- $\delta_{in}, \delta_{out} \in \{0, 1, 2, 3, 4\}$;
- $\mu \in \{0{,}0; 0{,}2; 0{,}4; 0{,}6\}$ (valores altos foram evitados a fim de ignorar redes com módulos fortemente acoplados).

Combinando-se todas as possíveis atribuições de valores a parâmetros dentro dos domínios escolhidos chega-se a 9.500 configurações possíveis.

O Modelo CGW

Para o modelo CGW, os seguintes valores de parâmetros foram escolhidos:

- $p_1, p_2, p_3, p_4 \in \{0{,}0; 0{,}2; 0{,}4; 0{,}6; 0{,}8; 1{,}0\}$, com $p_1 + p_2 + p_3 + p_4 = 1$ e $p_1 > 0$ (do contrário a rede jamais alcançaria 1.000 vértices);

- $e_1, e_2, e_3, e_4 \in \{1, 2, 4, 8\}$ (com a restrição de que e_i não varia quando $p_i = 0$, o que não faria sentido);
- $\alpha \in \{-1, 0, 1, 10, 100, 1000\}$
- $m \in \{2, 4, 8, 16, 32\}$.

O total de combinações, neste caso, é 38.790. O número elevado se deve à grande quantidade de parâmetros a serem combinados.

Além disso, considerou-se que a rede inicial é formada por dois vértices contidos em um mesmo módulo, juntamente com duas arestas opostas que ligam os vértices.

O Modelo LFR

Para chegar aos valores para os parâmetros do modelo LFR, foram analisadas métricas das redes de software contendo de 500 a 2.000 vértices. Para cada métrica foram identificados os valores mínimo, mediano e máximo; esses foram os valores usados nos parâmetros correspondentes. No caso das métricas grau médio, grau máximo e tamanho do menor módulo, os valores foram normalizadas de acordo com o número de vértices da rede analisada. Explicitamente, estes foram os valores escolhidos:

- parâmetro de mistura: $\mu \in \{0{,}0; 0{,}2; 0{,}4; 0{,}6\}$;
- expoente da distribuição de graus: $\gamma \in \{2{,}18; 2{,}70; 3{,}35\}$;
- expoente da distribuição de tamanhos de módulos: $\beta \in \{0{,}76; 0{,}99; 1{,}58\}$;
- grau médio: $k \in \{5, 10, 15, 25\}$;
- grau máximo: $max_k \in \{58, 157, 482\}$;
- tamanho do menor módulo: $min_m \in \{1, 10, 273\}$.

O total de combinações, neste caso, é 1.296.

Tabela 5.1: Resultados da classificação das redes geradas pelos modelos.

Modelo	**Proporção de Redes Software-Realistas**
ER	zero
BA	zero
BCR+	32,48%
CGW	63,57%
LFR	46,68%

5.3 Geração e Classificação de Redes

Para cada configuração de valores de parâmetros dos modelos BCR+, CGW e LFR, uma rede foi gerada. Para os modelos ER e BA foram geradas 20 redes por configuração, dado o número pequeno de configurações por modelo.

No caso do modelo LFR, foi usada a implementação original dos autores[1]. No caso dos modelos ER e BA foram usadas implementações disponíveis no pacote igraph [14].

Após a geração das redes, cada rede foi classificada como software-realista ou não software-realista, de acordo com o modelo de classificação apresentado no capítulo anterior. Os resultados estão condensados na Tabela 5.1.

Os modelos ER e BA geraram apenas redes não software-realistas, independentemente da configuração de parâmetros usada. Nos demais modelos, a proporção de redes software-realistas foi superior a 30%. A proporção exata de redes software-realistas para cada modelo não deve ser interpretada como medida de qualidade do modelo, uma vez que é diretamente influenciada pela seleção dos valores dos parâmetros.

5.4 Conclusão

Determinou-se, através de uma avaliação empírica, que os modelos CGW, LFR e BCR+ podem gerar, a depender dos valores atribuídos a seus parâmetros, redes software-realistas, isto é, redes que se assemelham a redes de software.

[1]Disponível em `http://santo.fortunato.googlepages.com/inthepress2`

Capítulo 6

Prova de Conceito: Estudo sobre Algoritmos de Agrupamento

6.1 Introdução

No capítulo anterior foi mostrado que pelo menos três modelos geram redes organizadas em módulos que se assemelham a redes de software. Neste capítulo, o modelo BCR+ é usado em três estudos experimentais realizados com o propósito de entender e comparar algoritmos de agrupamento que são estudados no contexto de recuperação de arquitetura de software. Os estudos são uma prova de conceito: eles demonstram a viabilidade do uso do modelo na avaliação de ferramentas de engenharia reversa.

Primeiramente, é feita uma breve introdução ao tema de algoritmos de agrupamento no contexto de recuperação de arquitetura. A seguir, são delineados os três estudos sobre algoritmos de agrupamento. Por fim, são apresentados os resultados e as conclusões dos estudos.

6.2 Agrupamento de Software

Agrupamento é o processo de organizar entidades em módulos (ou grupos) de maneira que as entidades de um mesmo módulo sejam similares entre si segundo algum critério. (O termo agrupamento designa também o resultado do processo, que é a atribuição de entidades a módulos.) Um exemplo clássico é o agrupamento de pontos em um plano (entidades) de acordo com o critério da distância, como mostra a Figura 6.1. O processo de agrupamento pode ser

automatizado por meio de algoritmos, os quais têm sido objeto de estudo de áreas diversas como inteligência artificial, mineração de dados, análise de redes sociais, entre outras.

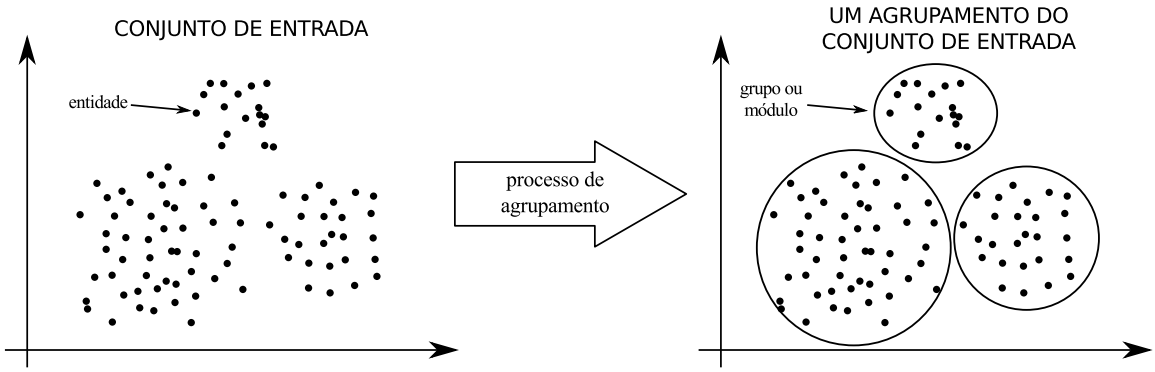

Figura 6.1: O agrupamento de um conjunto de pontos no plano.

Algoritmos de agrupamento têm sido usados no contexto da engenharia de software para apoiar a atividade de recuperação de arquitetura, e nesse caso são chamados de *algoritmos de agrupamento de software*. No início da atividade de recuperação de arquitetura, um algoritmo de agrupamento é aplicado a um sistema existente — descrito, por exemplo, como uma rede de dependências entre classes, no qual as classes representam as entidades a serem agrupadas. O agrupamento encontrado serve como ponto de partida para a identificação manual dos módulos da arquitetura do sistema e das entidades que compõem cada módulo.

O conjunto de módulos de uma arquitetura e entidades associadas constituem um agrupamento, denominado agrupamento de referência. O objetivo de algoritmos de agrupamento de software, neste contexto, é encontrar agrupamentos similares aos agrupamentos de referência dos sistemas analisados [7].

O custo de um experimento para avaliar algoritmos de agrupamento de software é elevado, uma vez que é necessário obter um agrupamento de referência para cada sistema analisado. Em uma avaliação feita por Girard e Koschke, quatro engenheiros de software levaram de 20 a 35 horas para obter agrupamentos de referência para programas na linguagem C com aproximadamente 30 mil linhas de código e apenas 40 entidades (tipos abstratos de dados) cada um [34]. O alto custo de experimentos com algoritmos de agrupamento faz desse tipo de estudo um bom candidato à técnica de avaliação através da simulação de modelos de redes.

6.3 Delineamento dos Experimentos

A fim de comparar algoritmos de agrupamento e de entender os fatores que influenciam o seu desempenho, três experimentos foram realizados. O primeiro experimento teve como propósito comparar o desempenho dos algoritmos de agrupamento sob o critério da semelhança dos agrupamentos encontrados pelos algoritmos com agrupamentos de referência. O segundo experimento teve como propósito entender como o desempenho dos algoritmos é afetado por parâmetros que descrevem cada rede. Por fim, o terceiro experimento teve como propósito entender como os algoritmos se comportam com diferentes configurações de dependências entre módulos.

Os métodos e materiais dos três experimentos são semelhantes e, por isso, são discutidos conjuntamente nesta seção. Os experimentos consistem nas seguintes etapas:

1. gerar redes usando diversas combinações de valores para os parâmetros do modelo BCR+;

2. aplicar algoritmos de agrupamento a cada uma das redes;

3. aferir o desempenho dos algoritmos de agrupamento em cada rede, isto é, medir a similaridade entre os agrupamentos encontrados pelos algoritmos e os agrupamentos de referência. O agrupamento de referência de uma rede, neste caso, é determinado pelos próprios módulos gerados pelo modelo BCR+;

4. interpretar os resultados.

A escolha do modelo BCR+ se deve a este ser o modelo mais completo, como foi visto na Tabela 3.1. Os estudos tiveram como ponto de partida as 9.500 configurações de parâmetros descritas no experimento de avaliação de modelos de redes, na Seção 5.2. Com a finalidade de mitigar os efeitos aleatórios na geração de redes, para cada configuração foram geradas três redes, totalizando 28,5 mil redes.

Para este estudo foram escolhidos algoritmos de agrupamento que possuem implementação disponível na Web e que já foram estudados por mais de um autor em pesquisas sobre recuperação de arquitetura. De acordo com esse critério, foram escolhidos os algoritmos ACDC, Bunch e algoritmos hierárquicos aglomerativos (ligação simples, SL, e ligação completa, CL), descritos no Apêndice B. Para cada um dos algoritmos aglomerativos, foram

estudadas duas alturas de corte[1], $0{,}75$ e $0{,}90$. No total são 4 configurações de algoritmos aglomerativos, que serão referenciadas como SL75, SL90, CL75 e CL90. Para os algoritmos Bunch e ACDC foram escolhidas as configurações originais das implementações dos autores dos algoritmos. As 6 configurações escolhidas para este estudo são idênticas àquelas estudadas por Wu, Hassan e Holt [35] (maiores informações no Capítulo 7).

Para medir a similaridade entre agrupamentos, foi escolhida a métrica MoJoSim [36, 37], que tem sido usada em estudos sobre agrupamento de software para recuperação de arquitetura. Em poucas palavras, a métrica MoJoSim mede, em uma escala contínua de 0 a 1, o esforço necessário para se transformar no agrupamento de referência um agrupamento dado. O valor 1 indica que os dois agrupamentos comparados são idênticos (e o esforço é, portanto, nulo). A métrica MoJoSim é apresentada com mais detalhes no Apêndice C.

Na métrica MoJoSim, considera-se que o esforço envolvido em unir dois módulos é pequeno se comparado ao esforço de dividir um módulo em dois. A justificativa é que, para dividir um módulo em dois, é preciso analisar cada entidade e determinar para qual módulo ela deve ser movida.

A partir dos dados coletados foram realizadas análises gráficas e testes de hipóteses. Uma análise preliminar revelou que os dados, em geral, não obedecem a uma distribuição normal, e por isso foram usados testes de hipótese não-paramétricos, os quais não assumem nenhuma distribuição particular dos dados. A variável dependente é, em ambos os experimentos, o desempenho de um determinado algoritmo em uma determinada rede, medido pela métrica MoJoSim. As variáveis independentes são os parâmetros do modelo BCR+ e a escolha do algoritmo de agrupamento.

6.4 Experimento 1: Comparação entre Algoritmos

O primeiro experimento teve como objetivo comparar o desempenho dos algoritmos de agrupamento com relação à similaridade entre agrupamentos encontrados pelos algoritmos e os agrupamentos de referência correspondentes.

[1] Em um algoritmo aglomerativo, a altura de corte influencia o número de módulos identificados: quanto maior a altura de corte, menor o número de módulos (e, consequentemente, maior o tamanho médio dos módulos). Para maior detalhes, consulte o Apêndice B.

Cada algoritmo foi aplicado a cada uma das redes sintéticas, resultando em agrupamentos cujo desempenho foi medido pela métrica MoJoSim. Apenas redes software-realistas, totalizando quase 6 mil redes, foram usadas neste estudo, uma vez que o objetivo é comparar o desempenho de algoritmos quando aplicadas a redes de software.

A Figura 6.2 mostra um *boxplot* dos valores de MoJoSim alcançados por cada algoritmo. No *boxplot*, o retângulo vai do quartil inferior (Q1) até o quartil superior (Q3), com a mediana desenhada como uma linha horizontal dentro do retângulo. Q1 representa o valor que é maior do que 25% dos valores e Q3 representa o valor que é maior do que 75% dos valores. Acima e abaixo do retângulo estão linhas horizontais que indicam o valor mínimo e o valor máximo do conjunto de dados.

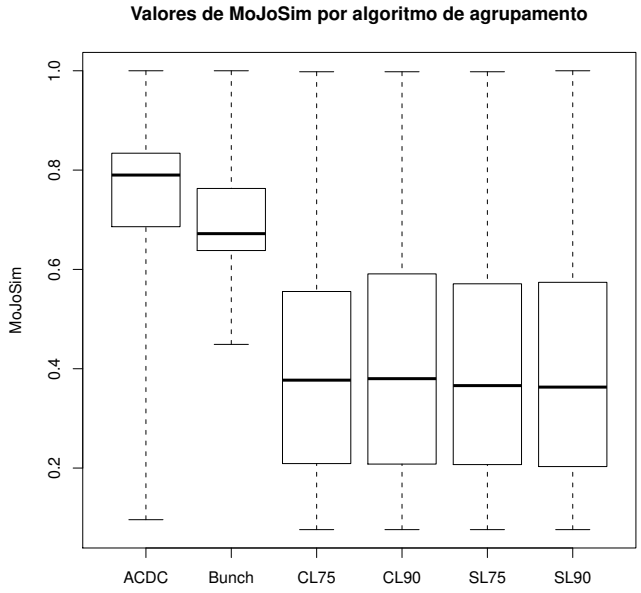

Figura 6.2: Resumo estatístico dos valores de MoJoSim de cada algoritmo de agrupamento.

Comparando pelo gráfico os MoJoSims medianos de cada algoritmo, nota-se que o algoritmo ACDC apresenta o melhor desempenho, seguido do Bunch. A seguir vêm os algoritmos aglomerativos, com pequenas diferenças entre si. A fim de verificar se as diferenças observadas são estatisticamente significativas, foi aplicado o teste de Wilcoxon pareado para as medianas de cada par de algoritmos, com nível de significância igual a 5%. Para mitigar o efeito que a aplicação de múltiplos testes exerce sobre a significância estatística, foi usada a correção de Bonferroni, que consiste em dividir o nível de significância pelo número de tes-

tes realizados, reduzindo a probabilidade de se obter falsos positivos. Os testes confirmaram as conclusões feitas a partir do gráfico, mas não foi encontrada nenhuma evidência de que algum algoritmo aglomerativo se destaque sobre os demais.

Outro aspecto a se observar é a dispersão dos valores. O algoritmo Bunch é o que apresenta a menor dispersão dentre os algoritmos analisados, com mais de 50% dos valores de MoJoSim na faixa de $0{,}60$ a $0{,}80$, e menor MoJoSim igual a $0{,}45$. No caso do ACDC, 50% dos valores de MoJoSim estão entre $0{,}65$ e $0{,}85$, porém o valor mínimo é $0{,}01$. Esta observação sugere que o algoritmo Bunch, embora apresente desempenho inferior ao ACDC na maioria dos casos, pode ser uma escolha interessante pelo fato de ter um desempenho mais previsível.

O resultado encontrado diverge das conclusões de Wu, Hassan e Holt [35]. Eles concluíram que os algoritmos, ordenados do melhor desempenho para o pior, são CL90, CL75, Bunch, ACDC, SL75, SL90. As divergências provavelmente se explicam pelos critérios empregados para definir o agrupamento de referência. No estudo deles, o agrupamento de referência foi extraído da estrutura de diretórios do código-fonte dos sistemas estudados; neste estudo, o agrupamento de referência é definido *a priori*, e as redes são geradas de forma a reduzir as dependências entre módulos.

Vale ressaltar que, apesar de uma amostra grande ter sido usada no estudo, as conclusões não são definitivas. Segundo o modelo de classificação de redes, todas as redes da amostra são software-realistas, mas não é possível afirmar que as redes software-realistas estejam bem representadas na amostra. Possivelmente o modelo BCR+ não é capaz de gerar certos tipos de redes software-realistas, o que potencialmente introduz um viés no experimento que pode beneficiar um algoritmo ou outro. Ainda assim, esta análise complementa estudos de caso realizados por outros autores [35, 38], que são enviesados pelo fato de analisarem uma amostra pequena de sistemas de software.

6.5 Experimento 2: Estudo de Parâmetros

O segundo experimento teve como objetivo estudar como o desempenho dos algoritmos é afetado pela variação de parâmetros do modelo.

O experimento seguiu uma configuração fatorial completa: considerando que cada parâ-

metro do modelo BCR+ pode assumir os valores discretos determinados na Seção 5.2, foram estudadas as 28,5 mil redes geradas a partir de todas as combinações de valores discretos para cada parâmetro do modelo. Tal configuração experimental permite estudar isoladamente o efeito de cada parâmetro sobre o desempenho de um algoritmo, algo que não é possível em estudos de caso.

Vale destacar que, neste caso, foram estudadas tanto redes software-realistas quanto redes não software-realistas, pois apenas desta forma é possível isolar cada parâmetro. Considere duas amostras das redes do estudo: a primeira composta de todas as redes com $\mu = 0{,}2$ e a segunda composta de todas as redes com $\mu = 0{,}4$. Exceto pelo valor de μ, cada combinação de valores de parâmetros na primeira amostra está presente também na segunda amostra e vice-versa. Assim, qualquer diferença entre os MoJoSim médios das amostras pode ser atribuído exclusivamente à variação no valor de μ e a efeitos aleatórios. Tal conclusão não seria válida se apenas redes software-realistas fossem analisadas. Nesse caso, não haveria uma correspondência completa entre as redes das duas amostras, e essa diferença entre as amostras também afetaria o valor de MoJoSim. Não seria possível, portanto, isolar a contribuição do parâmetro μ no resultado.

Tampouco seria viável realizar esse tipo de estudo usando a abordagem tradicional de avaliação, na qual os algoritmos de agrupamento são aplicados a redes extraídas de sistemas reais. Na abordagem tradicional, não há controle sobre as características das redes. Dificilmente duas redes extraídas de sistemas reais diferem apenas, por exemplo, no número de módulos: é mais provável que elas sejam diferentes também no número de vértices, no número de arestas etc. A abordagem proposta neste trabalho supre, portanto, essa deficiência da abordagem tradicional.

Um ponto que chamou a atenção foi a relação entre o desempenho dos algoritmos e o número de módulos do agrupamento de referência de cada rede. No caso dos algoritmos aglomerativos, um aumento no número de módulos provoca uma degradação no desempenho; nos demais algoritmos, não há uma variação significativa. Este fenômeno pode ser observado no gráfico da Figura 6.3, e foi confirmado através do teste de Wilcoxon pareado com 5% de significância. Esse é um comportamento, portanto, que diferencia os algoritmos aglomerativos dos demais.

Uma possível explicação está na distribuição dos tamanhos dos módulos encontrados pe-

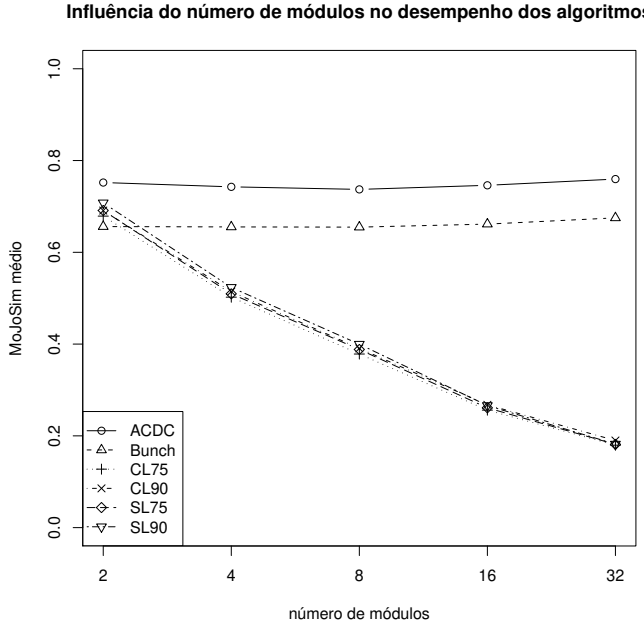

Figura 6.3: Influência do número de módulos do agrupamento de referência no desempenho de cada algoritmo de agrupamento.

los algoritmos aglomerativos. Nestes, é comum serem encontrados módulos muito grandes, às vezes contendo mais da metade da rede [35]. Quando o agrupamento de referência possui muitos módulos, os módulos grandes encontrados pelo algoritmo precisam ser divididos em diversos módulos menores, uma operação que é bastante penalizada na métrica MoJoSim.

6.6 Experimento 3: Influência do Tipo de Dependência entre Módulos

O terceiro experimento teve como objetivo medir a influência do tipo de dependência entre módulos (simples ou dupla) no desempenho de cada algoritmo de agrupamento. A variável independente deste experimento é o parâmetro G do modelo BCR+: o grafo de dependências entre módulos. Foram consideradas duas configurações para o parâmetro G, ambas contendo dois módulos, M_1 e M_2:

1. configuração *dupla*: vértices de M_1 podem se ligar a vértices de M_2 e vice-versa (diz-se que há uma dependência dupla entre M_1 e M_2);

2. configuração *simples*: vértices de M_1 podem se ligar a vértices de M_2, mas o contrário não é verdadeiro (diz-se que há uma dependência simples entre M_1 e M_2).

Vale notar que esse experimento não poderia ser realizado com os outros modelos de redes apresentados (CGW e LFR), pois eles permitem controlar apenas o número de módulos da rede. Somente o BCR+ possibilita restringir as dependências permitidas entre módulos.

Para os demais parâmetros do BCR+, foram considerados os valores apresentados na Seção 5. Como nos outros estudos, foram geradas três redes para cada configuração, totalizando 11.400 redes. A seguir, cada algoritmo de agrupamento foi aplicado a cada rede. Por fim, foi calculado o desempenho, medido pela métrica MoJoSim, de cada agrupamento.

A Figura 6.4 mostra, em um *boxplot*, o desempenho (valores de MoJoSim) alcançado por cada algoritmo nas duas configurações de G. Todos os algoritmos obtiveram desempenho ligeiramente superior nas redes geradas a partir da configuração *simples*. As diferenças entre desempenhos são estatisticamente significativas, exceto para o algoritmo CL75 (foi aplicado o teste de Wilcoxon pareado com nível de significância = 5%).

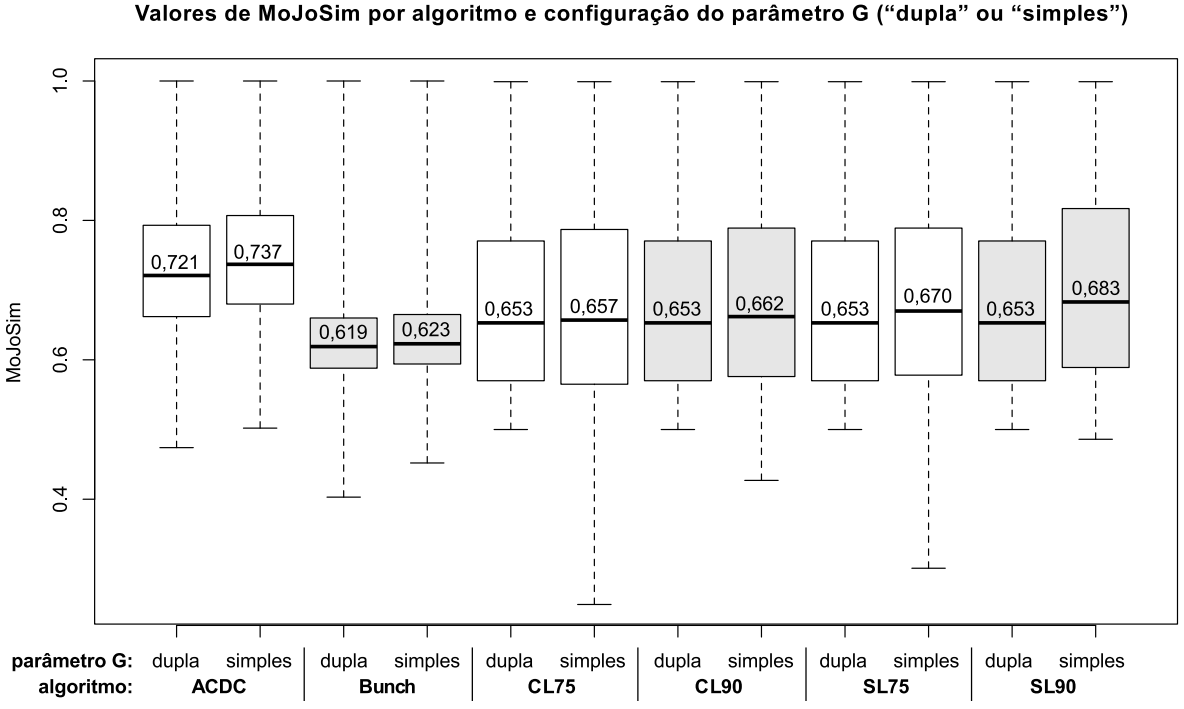

Figura 6.4: Influência do tipo de dependência entre módulos (simples ou dupla) no desempenho de cada algoritmo de agrupamento.

A Tabela 6.1 mostra, para cada algoritmo, a diferença de desempenho entre as duas configurações do parâmetro G. É reportado um intervalo de confiança (nível de confiança igual a 95%) para essa diferença. Pela tabela, conclui-se que que o ACDC e o SL75 são os algoritmos mais influenciados pela escolha entre a configuração *dupla* e a configuração *simples* do parâmetro G, e que a diferença entre os desempenhos, na maioria dos casos, não ultrapassa 0,03.

ACDC	**Bunch**	**CL75**	**CL90**	**SL75**	**SL90**
[0,016; 0,023]	[0,004; 0,008]	[-0,004; 0,006]	[0,006; 0,015]	[0,007; 0,017]	[0,021; 0,031]

Tabela 6.1: Intervalo de confiança de 95% para a diferença de desempenho de cada algoritmo, medido em MoJoSim, entre a configuração *simples* e a configuração *dupla* do parâmetro G.

Conclui-se que, em geral, a existência de módulos que dependem mutuamente um do outro influencia negativamente o desempenho dos algoritmos de agrupamento apresentados. No estudo realizado a partir de redes com dois módulos, essa diferença foi pequena, não ultrapassando 0,03 na maioria dos casos. Em redes com mais módulos e mais dependências entre os módulos, no entanto, é possível que a diferença seja maior, embora sejam necessários outros estudos para confirmar ou rejeitar tal hipótese.

6.7 Conclusão

Neste capítulo foram apresentados três estudos experimentais envolvendo a aplicação de algoritmos de agrupamento a redes geradas pelo modelo BCR+. Ao aplicar a abordagem proposta neste trabalho, os estudos demonstram que a abordagem é viável e complementa a abordagem tradicional.

Dois dos estudos apresentados, que avaliam a influência de determinadas características das redes no desempenho dos algoritmos, seriam difíceis de se realizar usando a abordagem tradicional, na qual os algoritmos são aplicados a redes extraídas de sistemas de software reais. O último estudo, em particular, mostra uma possibilidade de investigação que só pode ser explorada com auxílio do modelo BCR+.

Capítulo 7

Trabalhos Relacionados

O modelo CGW [27], assim como o modelo BCR+, foi proposto para modelar o crescimento, vértice a vértice, de redes de dependências extraídas de sistemas de software. Diferentemente do BCR+, o modelo CGW incorpora operações de remoção e religamento de arestas, que espelham a atividade de refatoração de programas. O modelo BCR+, em contrapartida, pode simular a evolução de um sistema de software sujeito a restrições na interação entre módulos, como ocorre na implementação de sistemas a partir de um *design* pré-estabelecido.

Stopford e Counsell propuseram um arcabouço para simular a evolução estrutural de sistemas de software [9]. O arcabouço incorpora conceitos como requisitos, desenvolvedores, métricas de código e base de código. Em simulações realizadas com o arcabouço, a exemplo do que ocorre em sistemas reais, o tamanho do código cresce linearmente ao longo do tempo. Não há qualquer discussão, no entanto, sobre a distribuição estatística das dependências entre entidades nas bases de código geradas pelo arcabouço.

Gunqun, Lin e Li investigaram a estrutura de 138 redes de software extraídas de programas escritos em Java [39]. Ao analisar a similaridade entre as redes, medida pela correlação entre os perfis de concentração de tríades (PCTs), eles concluíram que as redes podiam ser divididas em três grupos, de acordo com a similaridade. Redes grandes, com mais de 400 entidades, apresentaram grande similaridade entre si, e foram todas classificadas no mesmo grupo. Os autores não mediram, no entanto, a similaridade entre redes de software e redes de outros domínios, como foi feito no Capítulo 4 deste documento.

Com o propósito de avaliar algoritmos de agrupamento no contexto de redes sociais, Lancichinetti e Fortunato aplicaram 12 algoritmos de agrupamento a redes geradas pelo mo-

delo LFR [33]. Em particular, foi estudada a relação entre a proporção de arestas externas — determinado pelo coeficiente de mistura — e o desempenho dos algoritmos. Os demais parâmetros do modelo foram fixados em valores arbitrários, que não estão associados a nenhum domínio em particular (redes sociais, redes de software etc.). Como espera-se que os resultados variem de acordo com os parâmetros utilizados, as conclusões do estudo não são imediatamente aplicáveis a nenhum domínio específico, e nem são generalizáveis para todos os domínios. A arbitrariedade dos valores dos parâmetros pode ser reduzida através de um estudo como o que foi realizado no Capítulo 5.

O algoritmo de agrupamento Bunch foi avaliado por seus criadores através de sua aplicação a redes geradas aleatoriamente, variando o número de arestas [40]. Os autores do estudo encontraram uma grande variação no número de módulos encontrados pelo algoritmo para as redes mais densas, o que não ocorreu em redes esparsas. A julgar pelo fato de que o modelo de geração de redes aleatórias não foi discutido pelos autores, no entanto, é provável que não tenha sido escolhido um modelo livre de escala, o que compromete a generalização das conclusões para o domínio de sistemas de software.

No nosso conhecimento, este é o primeiro trabalho sobre geração automática de dependências de software realistas.

Capítulo 8

Conclusão

Neste trabalho foram apresentados modelos de redes que podem ser interpretadas como dependências entre entidades de código-fonte com a finalidade de apoiar a abordagem de simulação na avaliação de ferramentas de engenharia reversa. Mostrou-se que três dos modelos são capazes de gerar redes que se assemelham a redes de dependências estáticas entre classes em programas orientados a objetos. A partir deste resultado, foi feito um estudo sobre algoritmos de agrupamento de software, como prova de conceito, usando um dos modelos.

Este trabalho não diminui a importância de se avaliar ferramentas de engenharia reversa a partir da análise por especialistas de dependências extraídas de sistemas de software reais. Ainda assim, o uso de redes de software sintéticas de forma complementar pode ser vantajoso no estudo dos algoritmos. Primeiramente, o uso de modelos permite a criação de conjuntos de teste extensos. Além disso, as redes são criadas de maneira controlada, de acordo com parâmetros pré-definidos, o que torna possível estudar o comportamento dos algoritmos com diversos valores de parâmetros.

As contribuições desta pesquisa para o estado da arte podem ser assim resumidas:

- desenvolvimento de um modelo de redes organizadas em módulos, o BCR+;
- desenvolvimento e avaliação de um modelo de classificação que identifica redes que se assemelham a redes de dependências extraídas de sistemas de software;
- fornecimento de evidências empíricas de que os modelos de redes BCR+, CGW e LFR são capazes de gerar redes software-realistas;

- prova de conceito da viabilidade do uso dos modelos de redes para avaliar ferramentas de engenharia reversa, em particular, algoritmos de agrupamento aplicados no contexto de recuperação de arquiteturas de software.

As três primeiras contribuições desta pesquisa foram documentadas de forma preliminar no artigo "Modular Network Models for Class Dependencies in Software", apresentado na 14ª Conferência Europeia sobre Manutenção de Software e Reengenharia (CSMR 2010).

Naturalmente, este trabalho possui algumas limitações que poderão ser endereçadas em trabalhos futuros. A representação usada para sistemas de software — grafo orientado — é bastante simples e abstrai aspectos de sistemas de software que podem ser importantes, como a diferenciação de tipos de entidades (classes, atributos, métodos, arquivos fonte etc.). Além disso, o grafo orientado não diferencia diversos tipos de relacionamento entre entidades, como chamada de método, leitura de atributo etc. Em vez disso, todos os relacionamentos são abstraídos sob a forma do relacionamento de dependência.

Uma possibilidade interessante do modelo BCR+, que não foi explorada nesta pesquisa, é a geração de redes organizadas hierarquicamente em vários níveis. Por exemplo, seria possível gerar uma rede para representar um sistema de software em três níveis hierárquicos: métodos, classes e módulos (métodos agrupados em classes, classes agrupadas em módulos). Para isso bastaria executar o modelo diversas vezes, de modo que cada execução usasse como parâmetro G (grafo de dependências entre módulos) a rede produzida na execução anterior.

Referências Bibliográficas

[1] PARNAS, D. L. Software aging. In: *Proceedings of the 16th International Conference on Software Engineering ICSE '94*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994. p. 279–287. ISBN 0-8186-5855-X.

[2] CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, IEEE Press, v. 20, n. 6, p. 476–493, 1994. ISSN 0098-5589.

[3] TONELLA, P. et al. Empirical studies in reverse engineering: state of the art and future trends. *Empirical Software Engineering*, Kluwer Academic Publishers, v. 12, n. 5, p. 551–571, 2007. ISSN 1382-3256.

[4] ARNOLD, R. S.; BOHNER, S. A. Impact analysis - towards a framework for comparison. In: *Proceedings of the Conference on Software Maintenance ICSM '93*. Washington, DC, USA: IEEE Computer Society, 1993. p. 292–301. ISBN 0-8186-4600-4.

[5] ROY, C. K.; CORDY, J. R. A survey on software clone detection research. *School of Computing TR 2007-541, Queen's University*, v. 115, 2007.

[6] MAQBOOL, O.; BABRI, H. A. Hierarchical clustering for software architecture recovery. *IEEE Transactions on Software Engineering*, v. 33, n. 11, p. 759–780, 2007. ISSN 0098-5589.

[7] KOSCHKE, R.; EISENBARTH, T. A framework for experimental evaluation of clustering techniques. In: *Proceedings of the 8th International Workshop on Program Comprehension IWPC 2000*. [S.l.: s.n.], 2000. p. 201–210.

[8] WHITE, B. et al. An integrated experimental environment for distributed systems and networks. *SIGOPS Operating Systems Review*, ACM, v. 36, n. SI, p. 255–270, 2002. ISSN 0163-5980.

[9] STOPFORD, B.; COUNSELL, S. A framework for the simulation of structural software evolution. *ACM Transactions on Modeling and Computer Simulation*, ACM, v. 18, n. 4, p. 1–36, 2008. ISSN 1049-3301.

[10] NEWMAN, M. E. J. The structure and function of complex networks. *SIAM Review*, v. 45, p. 167–256, 2003.

[11] BARABASI, A.-L.; ALBERT, R. Emergence of scaling in random networks. *Science*, v. 286, p. 509, 1999.

[12] BARABASI, A.-L. The architecture of complexity: From network structure to human dynamics. *Control Systems Magazine, IEEE*, v. 27, p. 33–42, ago. 2007.

[13] MILO, R. et al. Network motifs: simple building blocks of complex networks. *Science*, v. 298, n. 5594, p. 824–827, out. 2002. ISSN 1095-9203.

[14] CSARDI, G.; NEPUSZ, T. The igraph software package for complex network research. *InterJournal*, Complex Systems, p. 1695, 2006. Disponível em: <http://igraph.sf.net>.

[15] MILO, R. et al. Superfamilies of evolved and designed networks. *Science*, v. 303, n. 5663, p. 1538–1542, mar. 2004. ISSN 1095-9203.

[16] VALVERDE, S.; SOLÉ, R. V. Hierarchical small worlds in software architecture. *Submitted to IEEE Transactions on Software Engineering*, 2003.

[17] MYERS, C. R. Software systems as complex networks: structure, function, and evolvability of software collaboration graphs. *Physical Review E*, v. 68, n. 4 Pt 2, p. 046116, out. 2003.

[18] MARCHESI, M. et al. Power laws in Smalltalk. In: *ESUG Conference*. Kothen, Germany: [s.n.], 2004.

[19] CONCAS, G. et al. Power-laws in a large object-oriented software system. *IEEE Transactions on Software Engineering*, v. 33, n. 10, p. 687–708, 2007. ISSN 0098-5589.

[20] HYLAND-WOOD, D.; CARRINGTON, D.; KAPLAN, S. Scale-free nature of Java software package, class and method collaboration graphs. In: *Proceedings of the 5th International Symposium on Empirical Software Engineering*. [S.l.]: ACM, 2006.

[21] BAXTER, G. et al. Understanding the shape of Java software. *SIGPLAN Notices*, ACM, v. 41, n. 10, p. 397–412, 2006. ISSN 0362-1340.

[22] ICHII, M.; MATSUSHITA, M.; INOUE, K. An exploration of power-law in use-relation of Java software systems. In: *Proceedings of the 19th Australian Conference on Software Engineering ASWEC 2008*. [S.l.: s.n.], 2008. p. 422–431. ISSN 1530-0803.

[23] LABELLE, N.; WALLINGFORD, E. Inter-package dependency networks in open-source software. *Computing Research Repository*, cs.SE/0411096, 2004.

[24] LOURIDAS, P.; SPINELLIS, D.; VLACHOS, V. Power laws in software. *ACM Transactions on Software Engineering and Methodology*, ACM, v. 18, n. 1, p. 1–26, 2008. ISSN 1049-331X.

[25] POTANIN, A. et al. Scale-free geometry in OO programs. *Communications of the ACM*, ACM, v. 48, n. 5, p. 99–103, 2005. ISSN 0001-0782.

[26] ERDÖS, P.; RÉNYI, A. On random graphs, i. *Publicationes Mathematicae (Debrecen)*, v. 6, p. 290–297, 1959.

[27] CHEN, T. et al. Module-based large-scale software evolution based on complex networks. *Proceedings of the 8th IEEE International Conference on Computer and Information Technology*, p. 798—803, 2008.

[28] LANCICHINETTI, A.; FORTUNATO, S.; RADICCHI, F. Benchmark graphs for testing community detection algorithms. *Physical Review E*, APS, v. 78, n. 4, 2008.

[29] LANCICHINETTI, A.; FORTUNATO, S. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E*, American Physical Society, v. 80, n. 1, p. 016118, jul. 2009.

[30] BOLLOBÁS, B. et al. Directed scale-free graphs. In: *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms SODA '03*. [S.l.]: Society for Industrial and Applied Mathematics, 2003. p. 132–139. ISBN 0-89871-538-5.

[31] TAN, P.-N.; STEINBACH, M.; KUMAR, V. *Introduction to Data Mining*. 2nd. ed. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2005. ISBN 0321321367.

[32] WITTEN, I. H.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd. ed. [S.l.]: Morgan Kaufmann, 2005. ISBN 978-0-12-088407-0.

[33] LANCICHINETTI, A.; FORTUNATO, S. Community detection algorithms: A comparative analysis. *Physical Review E*, v. 80, n. 5, p. 056117–+, nov. 2009.

[34] GIRARD, J.-F.; KOSCHKE, R. A comparison of abstract data types and objects recovery techniques. *Science of Computer Programming*, Elsevier North-Holland, Inc., v. 36, n. 2-3, p. 149–181, 2000. ISSN 0167-6423.

[35] WU, J.; HASSAN, A. E.; HOLT, R. C. Comparison of clustering algorithms in the context of software evolution. In: *Proceedings of the 21st IEEE International Conference on Software Maintenance ICSM '05*. [S.l.: s.n.], 2005. p. 525–535. ISSN 1063-6773.

[36] TZERPOS, V.; HOLT, R. C. MoJo: a distance metric for software clusterings. In: *Proceedings of the 6th Working Conference on Reverse Engineering WCRE '99*. [S.l.: s.n.], 1999. p. 187–193.

[37] BITTENCOURT, R. A.; GUERRERO, D. D. S. Comparison of graph clustering algorithms for recovering software architecture module views. In: *Proceedings of the 13th European Conference on Software Maintenance and Reengineering CSMR '09*. [S.l.]: IEEE Computer Society, 2009. p. 251–254. ISBN 978-0-7695-3589-0.

[38] ANDRITSOS, P.; TZERPOS, V. Information-theoretic software clustering. *IEEE Transactions on Software Engineering*, v. 31, n. 2, p. 150–165, 2005. ISSN 0098-5589.

[39] GUNQUN, Q.; LIN, Z.; LI, Z. Applying complex network method to software clustering. In: *Proceedings of the 1st International Conference on Computer Science and Software Engineering CSSE '08*. [S.l.]: IEEE Computer Society, 2008. p. 310–316. ISBN 978-0-7695-3336-0.

[40] MITCHELL, B. S.; MANCORIDIS, S. On the evaluation of the Bunch search-based software modularization algorithm. *Soft Computing*, Springer-Verlag, v. 12, n. 1, p. 77–93, 2007. ISSN 1432-7643.

[41] TRAUD, A. L. et al. Community Structure in Online Collegiate Social Networks. *ArXiv e-prints*, set. 2008.

[42] JEONG, H. et al. The large-scale organization of metabolic networks. *Nature*, v. 407, n. 6804, p. 651–654, out. 2000. ISSN 0028-0836.

[43] JEONG, H. et al. Lethality and centrality in protein networks. *Nature*, v. 411, p. 41, 2001.

[44] ADAMIC, L. A.; GLANCE, N. The political blogosphere and the 2004 U.S. election: divided they blog. In: *Proceedings of the 3rd international workshop on Link discovery LinkKDD '05*. [S.l.]: ACM Press, 2005. p. 36–43. ISBN 1595932151.

[45] WATTS, D. J.; STROGATZ, S. H. Collective dynamics of 'small-world' networks. *Nature*, v. 393, n. 6684, p. 440–442, jun. 1998. ISSN 0028-0836.

[46] ANQUETIL, N.; LETHBRIDGE, T. C. Experiments with clustering as a software remodularization method. In: *Proceedings of the 6th Working Conference on Reverse Engineering WCRE '99*. [S.l.]: IEEE Computer Society, 1999. p. 235–255. ISBN 0-7695-0303-9.

[47] MANCORIDIS, S. et al. Using automatic clustering to produce high-level system organizations of source code. In: *Proceedings of the 6th International Workshop on Program Comprehension IWPC '98*. [S.l.: s.n.], 1998. p. 45–52.

[48] TZERPOS, V.; HOLT, R. C. ACDC: an algorithm for comprehension-driven clustering. In: *Proceedings of the 7th Working Conference on Reverse Engineering WCRE '00*. [S.l.]: IEEE Computer Society, 2000. v. 0, p. 258–267. ISSN 1095-1350.

Apêndice A

Redes Empregadas na Avaliação dos Modelos

Tabela A.1: 65 redes de software (conjunto T).

Sistema	**S(x, T)**	**Vértices**	**Arestas**
abaguibuilder-1.8	0,94	875	3695
alfresco-labs-deployment-3Stable	0,82	134	271
argouml-0.28	0,93	2436	12505
ArtOfIllusion272	0,96	1041	8597
battlefieldjava-0.1	0,63	98	320
checkstyle-5.0	0,93	349	1084
dbwrench	0,97	1061	6732
dom4j-1.6.1	0,96	190	903
ec2-api-tools-1.3-36506	0,28	321	1023
findbugs-1.3.8	0,96	1581	8857
flyingsaucer-R8	0,92	536	2540
fm	0,95	508	1889
freetts-1.2.2-bin	0,89	113	399
ganttproject-2.0.9	0,96	1050	5177
gdata	0,94	1194	6926

GEF-0.13-bin	0,95	336	1387
geotools-2.5.5	0,95	6433	40651
GFP	0,85	418	889
groovy-all-1.5.5	0,95	1537	7733
guice-2.0	0,96	639	3123
gwt-windows-1.6.4	0,97	6076	32801
hibernate-distribution-3.3.1.GA	0,96	1711	10157
Hl7Comm.1.0.1	0,09	63	123
hsqldb	0,94	309	1763
iBATIS_DBL-2.1.5.582	0,94	279	1267
iReport-nb-3.5.1	0,94	2475	7068
jacorb	0,92	4346	38844
jai-1_1_4-pre-dr-b03-lib-linux-i586-08_Jun_2009	0,94	936	5086
jailer	0,87	328	1142
jalopy-1.5rc3	0,96	647	3460
jasperreports-3.5.2	0,97	1522	9080
jaxb	0,96	2280	14298
jboss-5.1.0.GA	0,97	4284	18360
jfreechart-1.0.13	0,95	1010	7663
jGnash	0,96	765	3513
jgraphpad-5.10.0.2	0,95	519	2257
jmsn-0.9.9b2	0,84	232	777
jruby	0,92	4714	33891
juel-2.1.2	0,92	111	437
juxy-0.8	0,87	77	198
jxplorer	0,96	394	1511
makagiga-3.4	0,96	1981	10873
MegaMek-v0.34.3	0,96	1787	11245
mondrian-3.1.1.12687	0,94	1716	12035

myjgui_0.6.6	0,9	127	419
oddjob-0.26.0	0,95	1208	5966
openide	0,95	1421	6545
ourgrid-4.1.5	0,72	855	3218
pdfsam-1.1.3-out	0,96	205	770
pentaho-reporting-engine-classic-0.8.9.11	0,96	1219	5727
pjirc_2_2_1_bin	0,83	180	687
pmd-4.2.5	0,92	723	3828
proguard4.3	0,96	604	4923
rapidminer-4.4	0,93	2648	15096
richfaces-ui-3.3.0.GA	0,95	865	1432
smc_6_0_0	0,95	148	1249
squirrel-sql-3.0.1-base	0,96	1171	4262
stendhal-0.74	0,95	468	1513
subethasmtp-3.1	0,85	85	217
thinkui_sqlclient-1.1.2	0,96	944	7563
tvbrowser-2.7.3	0,96	3335	14133
VilloNanny	0,96	100	585
weka	0,92	2165	12834
xalan	0,94	1439	10413
zk-bin-3.6.1	0,95	1753	8360

Tabela A.2: 66 redes de domínios diversos (conjunto $\bar{T}$).

Rede	**S(x, T)**	**Vértices**	**Arestas**
5 redes de amizade do site Facebook [41]	$-0{,}130 \pm 0{,}007$	768 a 18162	33312 a 1533600
3 redes de circuitos eletrônicos [15]	$0{,}433 \pm 0{,}005$	121 a 511	189 a 819

4 redes de adjacências entre palavras [15]	$0{,}573 \pm 0{,}064$	2703 a 11585	8300 a 46281
3 redes de estrutura proteica [15]	$0{,}540 \pm 0{,}046$	52 a 96	123 a 213
2 redes sociais de sentimento positivo [15]	$0{,}555 \pm 0{,}248$	31 a 66	96 a 182
43 redes metabólicas [42]	$0{,}509 \pm 0{,}015$	408 a 2360	792 a 5959
Rede de interação entre proteínas da levedura [43]	$0{,}22$	687	1079
Links entre blogs políticos [44]	$0{,}96$	1489	19090
Rede neural do verme C Elegans [45]	$0{,}87$	296	2359
Rede “beta3sreduced” (fonte desconhecida)	$0{,}05$	1286	33813
Rede “czech” (fonte desconhecida)	$0{,}78$	5225	51687
Rede “ecoli-metabolic” (fonte desconhecida)	$0{,}69$	895	964

Apêndice B

Algoritmos de Agrupamento

A seguir são descritos brevemente três tipos de algoritmos de agrupamento de software que operam sobre redes de dependências entre entidades.

B.1 Algoritmos Hierárquicos Aglomerativos

Algoritmos hierárquicos aglomerativos têm sido aplicados a diversos problemas, incluindo o agrupamento de software [6, 46]. Essa família de algoritmos funciona com qualquer descrição de entidade, desde que seja fornecida uma métrica de similaridade entre pares de entidades. A similaridade entre duas entidades é medida em uma escala contínua de 0 a 1. O algoritmo é simples: inicia-se com um módulo para cada entidade e então mesclam-se sucessivamente os dois módulos mais similares até restar apenas um módulo com todas as entidades.

No contexto de recuperação de arquitetura de software, é comum descrever as entidades de código-fonte como um grafo de dependências entre entidades e usar a métrica de Jaccard para medir a similaridade entre duas entidades [46]. Sejam X e Y duas entidades, e seja d(A) o conjunto de entidades ligadas à entidade A no grafo de dependências. A similaridade entre duas entidades, X e Y, é dada pela expressão a seguir:

$$\mathrm{sim}(X, Y) \;=\; \frac{|\mathrm{d}(X) \cap \mathrm{d}(Y)|}{|\mathrm{d}(X) \cup \mathrm{d}(Y)|}$$

O que diferencia os algoritmos hierárquicos aglomerativos entre si é o critério usado para medir a similaridade entre dois módulos. Os dois critérios mais estudados no contexto

de agrupamento são chamados de ligação simples (SL, do inglês *single linkage*) e ligação completa (CL, do inglês *complete linkage*). Na ligação simples, a similaridade entre dois módulos é computada como a maior similaridade entre pares de entidades tiradas uma de cada módulo. Na ligação completa é considerada a menor similaridade.

Naturalmente, um agrupamento que consiste de apenas um módulo com todas as entidades analisadas é de pouco valor. Por essa razão deve existir um critério de parada que interrompa o algoritmo antes de todos os módulos terem sido mesclados em um grande módulo. No contexto de agrupamento de software, o critério mais comumente usado é a altura de corte, h, que varia entre 0 e 1. Sob esse critério, o algoritmo interrompe sua execução quando a maior similaridade entre dois módulos é menor ou igual a $1 - h$. Assim, quanto maior a altura de corte h, menor o número de módulos do agrupamento resultante.

B.2 Bunch

Com a finalidade de auxiliar a compreensão de programas, o algoritmo Bunch [47] agrupa o conjunto de entidades de um programa de acordo com os relacionamentos existentes entre elas, representados como um grafo orientado. O agrupamento é tratado como um problema de otimização no qual o objetivo é maximizar uma função denominada qualidade de modularização (QM), que recompensa agrupamentos com muitas arestas internas (que ligam entidades de um mesmo módulo) e poucas arestas externas (que ligam entidades de módulos distintos).

Encontrar um agrupamento que possui QM ótima é um problema intratável; por isso o algoritmo Bunch usa heurísticas como algoritmos genéticos e *hill-climbing* para obter resultados quase ótimos.

B.3 ACDC

O algoritmo ACDC [48] foi projetado com o intuito de encontrar agrupamentos que facilitam a compreensão de programas. Assim como o Bunch, ele opera sobre grafos orientados. Sua principal característica é a identificação de conjuntos dominantes de vértices no grafo. Um conjunto dominante é um conjunto de vértices, $v_0, v_1, \ldots, v_n$, onde v_0 é o vértice dominante,

que satisfaz a duas condições: (i) existe um caminho entre v_0 e qualquer v_i; (ii) qualquer caminho de um vértice que não pertence ao conjunto até um vértice do conjunto, v_i, passa por v_0. O algoritmo ACDC identifica os conjuntos dominantes, em ordem crescente de número de vértices, e considera-os módulos do agrupamento.

Apêndice C

Métrica MoJoSim

A métrica MoJoSim [37], baseada na métrica MoJo [36], mede a similaridade entre dois agrupamentos, um dos quais é considerado agrupamento de referência. Neste capítulo a métrica MoJo é explicada através de um exemplo e, a seguir, a métrica MoJoSim é definida.

A Figura C.1 ilustra dois agrupamentos de um mesmo sistema de software hipotético, representado como um grafo orientado que descreve dependências estáticas entre classes do sistema. É fácil perceber que os agrupamentos não são idênticos: um deles possui quatro módulos, enquanto o outro possui três. Ainda assim, os agrupamentos são bastante semelhantes. O módulo M_4 corresponde exatamente ao módulo M_C; O módulo M_3 é quase igual ao módulo M_B; e os módulos M_1 e M_2, unidos, são parecidos com o módulo M_A.

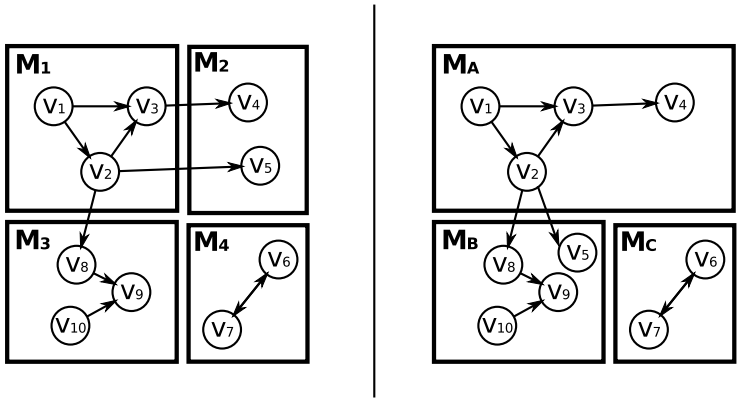

Figura C.1: Dois agrupamentos de um mesmo sistema de software hipotético, descrito como um grafo que representa dependências entre classes.

A ideia por trás da métrica MoJo é que um agrupamento dado é muito similar a um

agrupamento de referência se é possível transformar o primeiro agrupamento no segundo usando poucas operações do tipo mover e mesclar. A operação mover consiste de mover uma entidade de um módulo para outro módulo distinto. A operação mesclar consiste de mesclar dois módulos. O MoJo entre dois agrupamentos é igual ao número de operações de mover e mesclar que são necessárias para transformar o primeiro agrupamento no segundo. Quanto menor o MoJo, maior a similaridade entre dois agrupamentos; agrupamentos idênticos têm MoJo igual a zero.

Na Figura C.1, considerando que o agrupamento de referência é o segundo (o da direita), o MoJo entre os dois agrupamentos vale 2. Para transformar o primeiro agrupamento no segundo são necessárias, portanto, duas operações: mesclar os módulos M_1 e M_2, e mover o vértice v_5 para o módulo M_3. Após a realização dessas operações, os agrupamentos se tornam idênticos, considerando as correspondências $(M_1 \cup M_2) = M_A$, $M_3 = M_B$ e $M_4 = M_C$.

Observando que o valor MoJo entre dois agrupamentos com o mesmo número de entidades varia entre 0 e o número de entidades em cada agrupamento, n, é possível definir uma métrica, chamada MoJoSim, que mapeia o valor MoJo em uma escala de 0 a 1 [37]. O valor 0 representa a menor similaridade e o valor 1, a maior similaridade. O valor de MoJoSim entre dois agrupamentos é definido de acordo com a seguinte equação:

$$\mathrm{MoJoSim}(X, Y) \;=\; 1 - \frac{\mathrm{MoJo}(X, Y)}{n}$$

O valor de MoJoSim na Figura C.1 é, portanto, igual a $1 - \frac{2}{10} = 0{,}8$.