CRISTIAN SIMONS

PRIORIZAÇÃO DE CASOS DE TESTES DE REGRESSÃO USANDO AMOSTRAGEM POR PERSEGUIÇÃO DE DEFEITOS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito para a obtenção do título de Mestre em Informática.

CURITIBA

2010

CRISTIAN SIMONS

PRIORIZAÇÃO DE CASOS DE TESTES DE REGRESSÃO USANDO AMOSTRAGEM POR PERSEGUIÇÃO DE DEFEITOS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito para a obtenção do título de Mestre em Informática.

Área de Concentração: *Ciência da Computação*.

Orientador: Prof. Dr. Emerson Cabrera Paraiso

CURITIBA

2010

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

APD	*Amostragem por Perseguição de Defeitos*
CT	*Caso(s) de Teste(s)*
FAG	*Fase de Agrupamento*
FAM	*Fase de Amostragem*
FPA	*Filtragem por Agrupamentos*
GT	*Grupos de Testes ("Test Suits")*
MD	*Métrica de Dissimilaridade*
MPDD	*Média da Percentagem de Detecção de Defeitos ("APFD")*
PCT	*Priorização de Casos de Testes*
RGT	*Redução de Grupo de Testes*
RTT	*Refazer Todos os Testes ("Retest-all")*
STR	*Seleção de Testes de Regressão*
TBC	*Técnicas Baseadas em Cobertura ("Coverage-based techniques")*
TBD	*Técnicas Baseadas em Distribuição ("Distribution-based techniques")*
TF	*Testes Funcionais*
TR	*Testes de Regressão*
TRO	*Testes de Regressão Otimizados ("Cost-efficient Regression Tests")*
TS	*Testes de Sistemas*

Resumo

A necessidade de diminuir o custo de execução de testes de sistemas é ponto consensual na comunidade de produção de software. O presente estudo demonstra a otimização de teste de regressão, através da adaptação de uma técnica de priorização de casos de testes denominada Amostragem por Perseguição de Defeitos - já usada e comprovada para testes em geral – na execução exclusiva de testes de regressão. Para tal, foram adaptadas as fases de agrupamento e amostragem do método original, para que estas sejam capazes de incluir informações provenientes de testes realizados em versões anteriores dos programas, e possam usar estas informações importadas para direcionar a eficiência do método desenvolvido para os testes das versões atuais. Usando-se gráficos de média de percentagem de detecção de defeitos, demonstrou-se com os resultados obtidos um ganho significativo na eficiência de priorização de testes que revelam algum defeito - executando-os todos em torno de 60% da lista total de testes – quando comparados com resultados conseguidos com o uso da sequência original dos testes, ou de uma ordenação aleatória dos mesmos.

Palavras-Chave: (Testes de Regressão, Priorização de Casos de Testes, Amostragem por Perseguição de Defeitos, Análise de Agrupamentos).

Abstract

The necessity of lessening the cost of execution of system tests is a consensual point in the software production community. The present empirical study presents an optimization of the regression tests' activity, by adapting a test case prioritization technique called Failure Pursuit Sampling – already used and validated for general testing – for a regression tests exclusive execution. With this purpose, the clustering and sampling phases of the original method were adapted, so that they are capable of receive information coming from tests made on previous versions of the programs, and also capable of using the imported information to direct de efficiency of the proposed method for tests made on the present versions. By using Average of the Percentage of Faults Detected charts, the obtained results present a significant gain of efficiency on the fault revealing tests prioritization – by executing them all around 60% of the whole test list – when compared with the results achieved by using the test list original sequence, or a random list ordination.

Keywords: *(Regression Tests, Test Case Prioritization, Failure Pursuit Sampling, Cluster Analysis).*

Capítulo 1

Introdução

A fim de atender à crescente demanda da sociedade por acesso cada vez mais rápido, a um número cada vez maior e mais complexo de informação, sobrecarrega-se substancialmente o mercado de produção de software. Proporcional ao aumento da complexidade e da magnitude dos software desenvolvidos é também o aumento da probabilidade de defeitos existentes em cada conjunto destes. Consequentemente, se faz necessário que os testes que pretendem encontrar estes defeitos, acompanhem a evolução tornando-se cada vez mais rápidos, robustos e eficientes.

Para aperfeiçoar o desenvolvimento de software, inúmeros estudos ressaltam a importância de um processo de teste cada vez mais efetivo (DUSTIN et. al., 1999); (DUSTIN, 2002); (FEWSTER e GRAHAM, 1999); (MYERS, 2004); (PRESSMAN, 2009). Desta forma, quanto mais eficiente e mais eficaz forem os testes, menor será o custo dos reparos, e maior será a qualidade do produto final (XIE e NOTKIN, 2002).

De todos os casos de testes que precisam ser produzidos e executados - a uma taxa compatível à de produção dos software - existe uma classe que se torna cada vez mais crítica e vital para a manutenção do problema supracitado: os chamados Testes de Regressão.

Se as pessoas chaves do ambiente de produção de software (usuários, programadores, analistas, etc.) já vêem problema em encontrar um defeito em um software recém-desenvolvido, problema ainda maior se instaura ao tentarem encontrar um defeito em um software que se supunha estar funcionando a contento (ELFRIEDE, 2003). Os testes de regressão são então executados para que este segundo problema seja minimizado: introduzir defeitos onde estes não existem, ao se alterar um software com o intuito de melhorá-lo, corrigir defeitos, ou adicionar funcionalidades (RAKITIN, 2001).

Teste de regressão basicamente é todo e qualquer tipo de teste que, uma vez criado para assegurar o bom funcionamento do software, foi armazenado para ser reutilizado em uma versão posterior deste para - no mínimo - garantir que nada que já funcionava de acordo tenha sido comprometido (GRAVES et. al., 2001).

Realizadas estas considerações, percebe-se de imediato que o conjunto de testes de regressão se mantém em constante crescimento. Independentemente dos recursos (equipamentos, tempo e humano) disponíveis para a execução dos testes de regressão (sendo estes irremediavelmente finitos), em algum momento faltarão recursos para executar todos os testes de regressão disponíveis. Porém, se adicionarmos a esse processo as pressões exercidas pelas competições do mercado, as constantes contenções de recursos e as súbitas ocorrências de emergências, o problema ainda é bastante agravado.

1.1. Motivação para esta Pesquisa

É consenso na comunidade de desenvolvimento de software que a fase de teste de um produto de software é uma das fases de maior custo para os projetos. Existem empresas que investem de 50% a 70% (KANELLOPOULOS et. al., 2006) de seus recursos disponíveis nestas fases e qualquer iniciativa de diminuí-los é extremamente bem-vinda.

Também é fato, que quanto mais tardiamente um defeito é encontrado em um sistema sendo desenvolvido ou testado, maior é o custo de sua correção (MYERS, 2004), e que de nada adianta entregar-se novas e inovadoras funcionalidades a clientes, se estas alterações afetam a qualidade do sistema já utilizado (ELFRIEDE, 2003).

Estas afirmações criam uma motivação significativa para se otimizar qualquer fase do processo de teste de software.

1.2. Hipótese de Trabalho

Com o desejo de diminuir a necessidade de recursos em relação à fase de teste de regressão de um processo de teste de software, idealizou-se uma estratégia para adaptar uma técnica de priorização de casos de testes já conhecida e de eficiência comprovada para qualquer tipo de casos de teste – chamado Amostragem por Perseguição de Defeitos (LEON e PODGURSKI, 2003) - para o uso exclusivo na priorização de casos de teste de regressão.

1.3. Objetivos

Almejou-se com a realização deste estudo, validar a proposta de uma forma de priorizar casos de teste de regressão, para que - independentemente do número de casos de teste que se possa executar por limitação de recursos, ou até mesmo em casos de interrupção abrupta da sequência de execução de teste - tenha-se a convicção que o período disponível foi aproveitado da melhor forma possível.

1.3.1. Objetivos Gerais

De modo geral, pretende-se validar a proposta de uma técnica de priorização de casos de testes - adaptando a técnica já existente denominada Amostragem por Perseguição de Defeitos (usada para priorizar testes em geral) - para o uso específico em testes de regressão.

1.3.2. Objetivos Específicos

- Determinar quais são as características (atributos) dos casos de testes que possibilitem evidenciar as áreas de código sobre as quais os mesmos atuam;
- Selecionar e adaptar uma técnica de agrupamento capaz de formar agrupamentos de casos de testes com ênfase nas áreas de código que os mesmos supervisionem, dessa forma, aumentando a chance de que áreas de código alteradas sejam testadas com maior prioridade;
- Definir uma métrica de dissimilaridade, capaz de enfatizar a área de código que os testes percorrem;
- Escolher e adaptar uma estratégia de amostragem que possibilite a priorização completa do conjunto de testes de regressão, sem que a ênfase desejada seja comprometida;
- Calibração de um número de agrupamentos que produza o equilíbrio necessário para que casos de testes capazes de evidenciarem defeitos não sejam diluídos em meio aos demais, porém apresentando uma priorização eficiente.

1.4. Contribuição Científica

A contribuição científica mais importante deste estudo é a oportunidade de se usar e estudar técnicas da área de Inteligência Artificial – mais especificamente a geração de

agrupamentos – na atividade de priorização de casos de testes, bastante útil para a comunidade de desenvolvimento de software.

Podem-se citar também as contribuições de novos dados em relação a uma técnica de priorização de casos de testes já enunciada pela comunidade de otimização de testes, acrescentando informação de interesse a um tema parcialmente explorado e criando novas perguntas e oportunidades para trabalhos futuros.

1.5. Principais Resultados Obtidos

O principal resultado obtido - em termos gerais - é a confirmação de que a técnica de Amostragem por Perseguição de Defeitos (após as adaptações necessárias) – torna-se uma ferramenta eficiente na priorização de casos de testes de regressão.

Antes mesmo de se realizar qualquer ajuste fino do método desenvolvido para o caso específico sendo testado, pode-se observar uma melhora significativa na velocidade em que os testes que indicam defeitos em uma versão de código são executados, em detrimento dos testes que executam código sem indicarem defeitos.

Em algumas versões específicas do programa utilizado no estudo, tem-se uma melhora na localização de testes prioritários, que chega a executá-los todos quando a execução completa dos testes ainda se encontra nas marcas de 58% e 62% do todo – garantindo que se revele todos os defeitos que poderiam ser revelados muito antes de os testes terminarem.

1.6. Organização

Este documento está organizado de forma que todo o embasamento teórico necessário para o seu entendimento está apresentado nos dois primeiros capítulos. O capítulo 2 – Os Testes de Sistemas e os Testes de Regressão - está organizado de forma hierárquica descrevendo os temas, do mais genérico (Testes de Sistemas) ao mais específico (Amostragem por Perseguição de Defeitos), mantendo a linha de pensamento que os liga através dos Testes de Regressão. Alguns assuntos adjacentes a cada tema são explanados com o intuito de embasar as decisões tomadas a seguir no documento.

No capitulo 3 – A Tarefa de Agrupamento – descreve um tema chave para a construção do método desenvolvido. O tema foi rapidamente identificado assim que tomou-se a decisão de utilizar uma técnica de otimização baseada em distribuição de Casos de Testes em um espaço multidimensional. O capítulo 3, de modo análogo ao seu predecessor, é

organizado de forma hierárquica descrevendo os temas referentes à Tarefa de Agrupamento até os Algoritmos Hierárquicos Aglomerativos e assuntos adjacentes.

Já o capítulo 4 – Método Proposto - enuncia e descreve a metodologia usada para a construção do método de Priorização de Casos de Testes de Regressão usando Amostragem por Perseguição de Defeitos e explica o método propriamente dito. O capítulo baseia-se nos conceitos apresentados nos dois capítulos anteriores.

O capítulo 5 contém os passos realizados para executar o método desenvolvido empiricamente (evidenciando os passos genéricos que poderão ser reutilizados exatamente como estão para qualquer outro cenário de testes de regressão – e os passos específicos – que precisarão ser retrabalhados devido a qualquer mudança de contexto) e os resultados específicos obtidos, dependentes dos dados alvos selecionados para o estudo. O capítulo está organizado de forma cronológica, para facilitar o entendimento das interpretações e ajustes feitos durante a aplicação do método desenvolvido, e da geração dos resultados.

Finalizando o documento resultante do estudo, o capítulo 6 traz as conclusões gerais do trabalho.

Capítulo 2

Os Testes de Sistemas e os Testes de Regressão

Este capítulo tem por objetivo fornecer o embasamento teórico necessário para o entendimento dos testes de sistemas de forma geral, entrando em detalhes mais específicos no que diz respeito aos testes de regressão - um dos assuntos principais deste estudo. Explanações pertinentes à tarefa de agrupamento estão descritas no capítulo 3, assim como os detalhes do método desenvolvido podem ser encontrados no capítulo 4.

O presente capítulo está formatado basicamente como uma "árvore" de assuntos onde os assuntos seguintes apresentam detalhes de uma área específica do assunto anterior, mais geral. Para cada assunto específico também são apresentados alguns temas "adjacentes", para que seja possível o entendimento das decisões feitas durante a geração do método desenvolvido.

Este capítulo se inicia com o assunto mais geral de testes de sistemas, explora com mais profundidade o que são os testes de regressão e suas diferentes metodologias, segue então para as técnicas de priorização de casos de testes (a categoria na qual se encaixa o método desenvolvido), até chegar às técnicas baseadas em distribuição e, especificamente, a

Amostragem por Perseguição de Defeitos – técnica com a qual se pretende abordar o problema específico: A priorização de casos de testes para testes de regressão.

Este capítulo ainda discorre - ao fim - sobre a Medida de Percentagem de Detecção de Defeitos (MPDD), medida usada para demonstrar a eficiência do método desenvolvido.

2.1. Testes de Sistemas de Software

É consenso na comunidade de produção de software que software complexos, por melhor que tenham sido projetados e executados, contêm defeitos (SUNDMARK et. al., 2005). Portanto, é correto concluir que qualquer software de alguma complexidade precise ser constantemente verificado, validado e corrigido para que estes defeitos, provavelmente presentes, sejam em sua maioria encontrados e remediados, reduzindo-se o seu número ao mínimo possível. No entanto, deve-se ressaltar que esta atividade de manutenção de sistemas de software é bastante custosa em termos de recursos (financeiros, humanos e temporais) (BROOKS, 1995), sendo vista como uma tarefa necessária que precisa encontrar um equilíbrio entre o seu custo e o seu benefício.

Neste capítulo é apresentado um estudo teórico para embasar o que entendemos por Testes de Sistemas para software, incluindo explanações mais detalhadas sobre conceitos centrais deste projeto como: Testes de Regressão, Metodologias de Otimização de Testes de Regressão e Técnicas de Priorização de Casos de Testes.

2.1.1. Testes de Sistemas

Os Testes de Software têm por objetivo encontrar defeitos no(s) artefato(s) de software a ser usado (PRESSMAN, 2009), ou seja, assegurar que o mesmo esteja alinhado com os requisitos iniciais propostos, desempenhando as funcionalidades que lhe são esperadas, e livre de defeitos conhecidos (RAJA e TRETTER, 2007), (XIE e NOTKIN, 2002), (WEBER et. al., 1990).

As etapas de teste podem adquirir bastante complexidade e formalidade (SOMMERVILLE, 2003), podendo variar de simples testes de observação do sistema (beta-testes) até ambientes automatizados que maximizem a cobertura de código e minimizem a interação humana (DICKINSON et. al., 2001a). Todavia, quanto mais complexo e extenso for o software desenvolvido, mais formal e organizado seu ambiente de testes de sistemas tende a ser para que haja uma garantia concreta de qualidade.

Normalmente os testes iniciais derivam diretamente dos requisitos de um sistema e são chamados de Testes Funcionais (MYERS, 2004), cujo objetivo é encontrar os defeitos que minam o correto comportamento do sistema em relação ao que foi proposto de início. Qualquer discrepância ou comportamento indesejado deve ser considerado um defeito que precisa ser corrigido. Assim como qualquer conformidade ou comportamento esperado torna-se um registro que embasará a garantia de qualidade do artefato de software.

Os casos de teste comumente se apresentam em forma de uma lista de testes cujo conteúdo pode variar um pouco, mas basicamente contém uma tríade {Identificador, Entradas, Saídas} (ROTHERMEL e HARROLD, 1997) de informação para cada teste: i) um identificador para o teste; ii) o estado das entradas do sistema para o teste em questão; iii) o resultado esperado das saídas do sistema . Dada a entrada, analisa-se a saída. Caso se confirme à correspondência o identificador é associado a um acerto, caso contrário, há um defeito.

Lembra-se aqui que para sistemas complexos esta lista de testes pode vir a se tornar bastante grande, chegando à ordem de milhares de testes bem como apresentar uma complexidade proporcional a do sistema em si, pelas dependências sequenciais entre os mesmos. Devido a estes fatores é bastante comum se adotar o agrupamento de testes em sequências lógicas comumente denominadas de Casos de Testes (CT), que por sua vez são agrupados em Grupos de Testes (GT). Entretanto, os critérios de como se organizar os CT e os GT são apresentados em inúmeras fontes literárias e artigos científicos que apresentam inclusive alguns conselhos contraditórios (ROTHERMEL et. al., 2004).

É importante notar que esta organização toda não seria necessária se estes testes fossem executados apenas uma vez. Em uma situação ideal os artefatos de software seriam escritos, o teste seria realizado e o produto finalizado. Porém, todo esse esforço de formalidade e organização se deve ao fato de que nenhum software escrito é perfeito. Faz-se necessário, portanto, que sejam seguidamente aperfeiçoados, exigindo também que todos os seus testes sejam armazenados para que possam ser reutilizados posteriormente como Testes de Regressão (SUNDMARK et. al., 2005).

2.1.2. Testes de Regressão

O Teste de Regressão (TR) representam uma etapa mais específica. Têm por finalidade garantir que todo o esforço de testes outrora já realizado em relação ao artefato de

software continue oferecendo a garantia de qualidade após qualquer alteração do mesmo (SOMMERVILLE, 2003).

O desenvolvimento de software complexos é longo e contínuo, muitas vezes transcendendo o seu ciclo de vida inicialmente proposto, e normalmente sofrendo impactos de mudanças drásticas de escopo durante ou após o seu desenvolvimento (KANELLOPOULOS et. al., 2006). Seja pela correção de defeitos ou não-conformidades encontrados, seja pela introdução e desenvolvimento de novas funcionalidades ou até mesmo por adaptações exigidas pelas alterações dos requisitos e escopo propostos inicialmente, um artefato de software já escrito costuma ser alterado seguida e constantemente. Soma-se a isso o agravante destas alterações, muitas vezes, serem realizadas por mais de um desenvolvedor e até mesmo ocorrerem de modo simultâneo.

Sendo a estrutura de software complexos uma verdadeira "teia" de artefatos de software e processos sendo executados de maneiras simultâneas e por vezes não sincronizados, torna-se bastante difícil prever todos os impactos resultantes de uma determinada modificação de código em um ponto específico do sistema (DUNOP et. al., 2001).

Casos de teste de regressão devem ser reutilizados na nova situação do sistema para garantir que qualquer alteração realizada não afete a já alcançada estabilidade de funcionamento (RAKITIN, 2001). De nada adianta promover inovações e atender a exigência de clientes para novas funcionalidades se a introdução desses fatores no sistema também afetarem negativamente as funcionalidades que já vinham sendo executadas de maneira satisfatória (ELFRIEDE, 2003).

Como exemplo, considerar-se-á um programa *P* que atenda os requisitos de uma série de funcionalidades *F* (enumeradas de *F1* a *Fn*) e que - após ter sido testado por um grupo de testes *G* (que não evidenciou defeitos) - *P* é liberado para uso. Tem-se então que durante a execução de *P* um defeito *D* em *Fx* é encontrado. Este cenário, ilustrado na Figura 2.1, desencadearia as seguintes ações:

- Um caso de teste *T* adicional deve ser escrito (ou um caso de testes existente, adaptado), pois fica claro que o grupo de testes *G* usado durante os testes de *P* não foi capaz de encontrar *E*. *T* será necessário para que, quando *D* for corrigido, possa-se garantir que o mesmo não ocorre mais.
- Uma correção *C* é feita para *D* gerando uma versão *P'* de *P*.

- Existe a possibilidade de que, devido a alguma dependência erroneamente desconsiderada entre *F-Fn*, ao se gerar *C* possa-se simultaneamente gerar *D'* em *Fy*.

Definimos como TF a execução de *T* sobre *P'* para garantir que *C* tenha corrigido *D*.

Definimos como TR a execução de *G* sobre *P'* para encontrar qualquer *D'* que possa ter sido gerado por *C*.

Definimos como TS a execução de TF e TR.

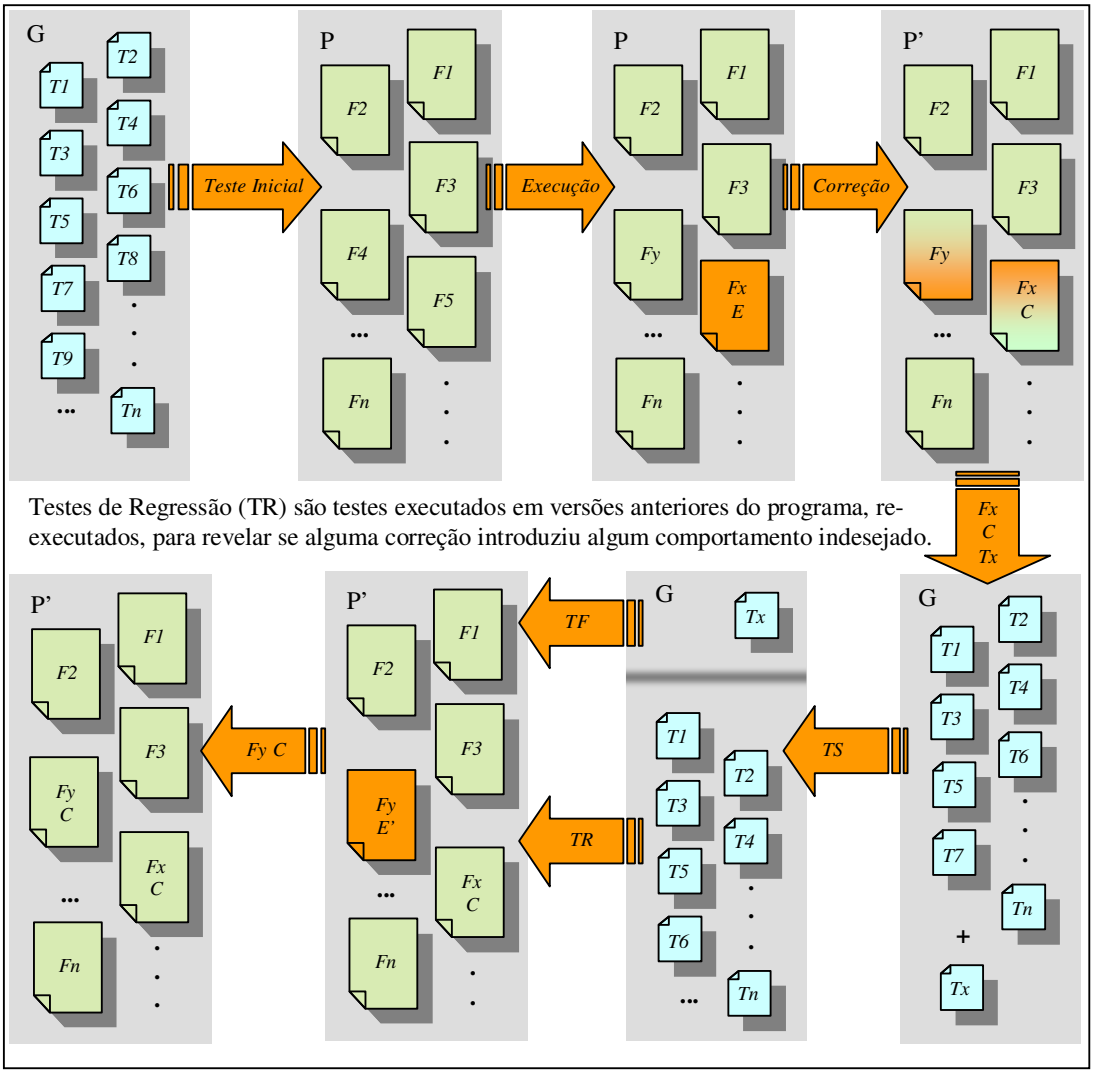


Figura 2.1: Fluxo dos Testes de Regressão

Atualmente, a maneira mais simples e garantida de se executar os TR ainda é através da técnica de "Refazer Todos os Testes" (RTT). Está técnica dita que, com a exceção de testes

obsoletos que já não traduzem mais a execução atual do sistema dada às alterações, devem-se refazer todos os testes outrora feitos para a versão anterior. Este procedimento visa a garantir que a versão atual mantenha o comportamento adequado e não apresente a introdução de nenhum efeito colateral indesejado. Por outro lado, recitando o conceito de equilíbrio entre o custo e o benefício levantado anteriormente, o processo de RTT pode vir a se tornar bastante custoso. Há registros de TR que precisam de sete semanas para serem executados em RTT (ROTHERMEL et. al., 2001) e, devido ao processo incremental dos mesmos, é correto afirmar que este problema tende apenas a se agravar.

Recentemente, os avanços mais significativos na área de TR concentram-se no estudo de várias técnicas para otimizá-los, para que os mesmos possam ser executados de maneira mais eficiente, gastando menos tempo e esforço e mantendo uma qualidade próxima da conseguida com a RTT.

2.1.3. Metodologias de Otimização de Testes de Regressão

Testes de Regressão Otimizados (TRO) implicam em realizar os TR em um período menor mantendo uma capacidade satisfatória de evidenciar defeitos de regressão (qualquer efeito colateral introduzido indesejadamente por uma alteração de código). Varias técnicas de TRO vem sendo sugeridas, estudadas e analisadas nos últimos anos (ROTHERMEL et. al., 2004), porém todas acabam por se enquadrar em uma das estratégias descritas abaixo. Para explicá-las definiremos que *P* é um programa, *P'* uma versão modificada deste programa e *G* um GT desenvolvido para *P*. Deste modo, os TR podem ser definidos como sendo testes com o propósito de verificar se a modificação que gerou *P'* não produziu nenhum efeito colateral indesejado no restante do código.

A primeira e já citada metodologia é a RTT. Quando *P* é alterado, criando *P'*, são usados todos os testes não obsoletos em *G* para testar *P'*. Apesar de não ser considerada uma metodologia de TRO é usada a título de comparação toda vez que se pretende averiguar a capacidade de evidenciar defeitos de regressão de qualquer *G'*.

Seleção de Testes de Regressão (STR): São técnicas de minimização, fluxo de dados, aleatórias (GRAVES et. al., 2001) e baseadas em segurança (ROTHERMEL et. al., 2004), (ROTHERMEL et. al., 2001), que usam informações de *P*, *P'* e *G* para selecionar um subgrupo de *G* que será usado para testar *P'*. As características que são trabalhadas durante a formação deste subgrupo *G'* são segurança e eficiência. As técnicas de STR que enfatizam a

segurança, chamadas de seguras, procuram garantir que, sob condições específicas, os CT não selecionados não seriam capazes de evidenciar defeitos em *P'*. No entanto, atingir essa segurança normalmente significa incluir um número maior de CT, que pode vir a comprometer a eficiência (tempo de execução). Já as técnicas inseguras sacrificam a segurança pela eficiência, selecionando CT que de alguma forma são mais úteis que os excluídos. Um caso especial dentre as técnicas inseguras, por exemplo, envolve técnicas que tentam minimizar o GT selecionado em relação a critérios de cobertura sobre as alterações em *P'*, procurando o GT com menor tempo de execução possível que cubra o trecho de código modificado.

Redução de Grupo de Testes (RGT): Com a evolução de *P* novos CT precisam ser adicionados a *G* para validar novas funcionalidades. Técnicas de RGT (ROTHERMEL et. al., 2004) procuram usar informações em *P* e *G* para permanentemente remover CT redundantes em *G*, deixando subsequentes utilizações de *G* mais eficientes. RGT diferem de STR no sentido de que a última não remove nenhum CT de *G* de modo permanente, apenas os "separa" para utilizá-los em uma versão específica *P'* de *P*, mantendo CT não utilizados para usos futuros. As RGT são também comumente utilizadas independentemente de *P'* (diferentemente das STR). Porém, as metodologias RGT oferecem uma desvantagem em potencial, pois existe sempre a possibilidade da remoção permanente de CT de um GT afetar a sua capacidade de detecção de defeitos.

Priorização de Casos de Testes (PCT): Esta metodologia engloba técnicas que ordenam os CT de acordo com determinados critérios, de modo que os mais prioritários sejam executados primeiro durante a execução dos TR (ELBAUM et. al., 2002), (ELBAUM et. al., 2004). Por exemplo, os CT podem estar ordenados para atingir uma completa cobertura de código o mais rápido possível, ou fazer uso das funcionalidades de acordo com a expectativa de frequência de uso das mesmas, ou aumentar a possibilidade de detecção de defeitos no início dos testes.

Uma vez evidenciada a intenção de se executar algum tipo de processamento prévio antes de se executar TR com o intuito de otimizá-lo, é preciso deixar bastante claro que este processamento, antes de mais nada, gerará esforço e portanto é necessário um meio de se modelar o "custo" gasto por uma TR e sua TRO. Segundo Leung e White (LEUNG e WHITE, 1990) o Custo de Regressão pode ser definido por $A + E(G')$, onde A é o custo em tempo do processamento da otimização do TR e $E(G')$ o custo de execução do TRO

propriamente dito. Segundo esta definição, o custo de regressão de uma RTT seria definido apenas por *E(G)*, ou seja, o custo em tempo de se executar todos os CT disponíveis. A idéia é que qualquer TRO para ser considerada, precisa obedecer à regra *A+E(G') < E(G)*. O tempo de execução da técnica de TRO somado ao tempo de pré-processamento para otimizá-la não pode de maneira alguma ultrapassar o temo de execução de RTT (GRAVES et. al., 2001).

2.2. Técnicas de Priorização de Casos de Testes

O problema de PCT pode ser definido como (ELBAUM et. al., 2004): sendo *G* um grupo de testes; *PG*, um grupo de permutações de *G*; e *f*, uma função de PG para um número real, encontrar *G'*.

$$G' \in PG \rightarrow (\forall G'')(G'' \in PG)(G'' \neq G')[f(G') \geq f(G'')] \qquad (2.1)$$

Na fórmula 2.1 *PG* representa o conjunto de todas as permutações possíveis (diferentes ordens) de *G*, enquanto f é a função que aplicada a qualquer uma destas ordens lhe fornece um valor de premiação.

Entrando em maiores detalhes sobre a metodologia PCT, um dos focos deste projeto, é interessante citar que a mesma ainda é encontrada subdividida em diferentes categorias na literatura atual como: Técnicas Baseadas em Cobertura (TBC) e Técnicas Baseadas em Distribuição (TBD) (LEON e PODGURSKI, 2003). As TBC baseiam-se em uma idéia bastante tradicional na área de testes de software e estão apoiadas na execução de um GT sobre a estrutura completa de um sistema. Independente do tipo de elemento de software usado (ex. classe, função, comando, etc.), deve haver ao menos um CT exercitando cada elemento. A priorização normalmente se dá dos CT mais genéricos (que mais cobrem diferentes elementos de software), para os mais específicos (que menos cobrem elementos de software).

Enquanto as TBC priorizam CT baseados na contribuição incremental para maximizar a cobertura de código, as TBD priorizam CT baseados em suas próprias definições dentro do Espaço Multidimensional de Definições regida por uma Métrica de Dissimilaridade (MD), como veremos a seguir.

2.2.1. Técnicas Baseadas em Distribuição

Como mencionado, as TBD são técnicas que se baseiam no uso de uma função que, para cada par de definições resulta em um número real representando o grau de dissimilaridade entre os mesmos, ou MD. Um exemplo de um Espaço Multidimensional de Definições poderia ser o espaço n-dimensional definido pela aplicação da Medida de Distância Euclidiana sobre definições com *n* características (DICKINSON et. al., 2001b). Adicionalmente, ainda é possível usar diferentes pesos para cada uma das características conforme se queira evidenciá-las (aumentar-lhes a relevância) ou diluí-las (diminuir-lhes a influência).

Uma vez tendo-se a MD entre cada CT retirado da similaridade entre suas definições, podem-se construir agrupamentos de CT relativamente similares capazes de indicar possíveis CT redundantes para o cenário em questão. É possível até mesmo representá-los graficamente, onde CT bastante isolados podem indicar testes inusitados que têm maiores chances de revelarem problemas desconhecidos ou usar áreas de pouca densidade para indicar cenários ou situações com deficiência de CT para cobri-los (DICKINSON et. al., 2001b).

Dentre as técnicas de TBD já estudadas é interessante citar os exemplos de Filtragem por Agrupamentos (FPA) e Amostragem por Perseguição de Defeitos (APD) (LEON e PODGURSKI, 2003). A FPA é baseada na possibilidade de se conduzir uma análise automática dos dados para formar agrupamentos em uma população de elementos. Estes agrupamentos são gerados de acordo com a MD entre os mesmos, indicando certa similaridade. É preciso selecionar uma técnica ou algoritmo de agrupamento para se conseguir o resultado desejado. Uma vez formados os agrupamentos é necessário também escolher um método de amostragem como, por exemplo, selecionar um elemento por agrupamento de maneira aleatória.

A técnica seguinte, APD, trata-se de uma extensão da FPA e explora a idéia de que CT que evidenciam defeitos normalmente são reunidos nos mesmos agrupamentos. A perseguição de defeitos representa o fato de que uma vez que se encontre na FPA um teste que evidencie um defeito, *k* "vizinhos" deste CT são selecionados para fazer parte do GT. Caso algum desses *k* CT também acabe por evidenciar algum defeito, novamente *k* "vizinhos" deste último CT são selecionados e assim por diante, até que não haja mais nenhum CT para ser selecionado, ou os CT selecionados não evidenciem mais defeitos.

2.2.2. Amostragem por Perseguição de Defeitos

De modo análogo a seção anterior onde uma formalidade foi usada para exemplificar TRs, define-se agora um grupo finito de casos de testes de *T1* a *Tn* denominado *G*. Pode-se afirmar que o problema de PCT é formar um grupo de testes *G'* onde os casos de testes *T1* a *Tn* estejam ordenados de forma que o primeiro caso de teste *T1'* a ser executado é aquele com a maior probabilidade de evidenciar um defeito *D'* no programa *P'*, o segundo caso de teste T2' é conseguido da mesma forma, porém considerando agora o grupo de testes *G'-T1'* e assim por diante.

O funcionamento da técnica de priorização de casos de testes APD, por sua vez, pode ser exemplificado através dos seguintes passos (e é apresentado na Figura 2.2):

1. Separa-se *G* em um número *m* de agrupamentos onde todos os CTs de *T1* a *Tn* vão para agrupamentos que evidenciem suas similaridades quanto ao código que testam;
2. Com os agrupamentos formados, coleta-se aleatoriamente *um* CT de cada agrupamento, que são então designados como testes da lista ordenada *G'*;
3. Executam-se os CTs recém adicionados a *G'* sobre *P'*;
4. Caso haja algum *Tx* que evidencie algum defeito em *P'*, *k* "vizinhos" de *Tx* são coletados de seu antigo agrupamento (os *k* vizinhos coletados são os que possuem a maior similaridade com *Tx*) e o Passo 3 é repetido. O Passo 4 é repetido com qualquer CT que evidencie algum defeito até que nenhum defeito seja mais evidenciado. Caso contrário, segue-se ao Passo 5;
5. Repete-se toda a sequência do Passo 2 em diante até que todos os CT tenham sido coletados formando *G'*.

2.2.3. Média da Percentagem de Detecção de Defeitos

Como visto no início deste capítulo, pode haver muitos objetivos para os diferentes resultados de priorização. Estes objetivos podem ser descritos de maneira informal como melhorar a taxa de detecção de defeitos de um GT, ou de maneira formal em forma quantitativa, para que se tenha um método de comparação entre diferentes PCT. Em Elbaum (ELBAUM et. al., 2004) é demonstrada uma métrica criada por Rothermel como *APFD – Average of the Percentage of Faults Detected*, mas que aqui denominaremos de Média da Percentagem de Detecção de Defeitos (MPDD). A MPDD varia de 0 a 100, onde valores maiores indicam taxas de detecção de defeitos mais rápidas (melhores).

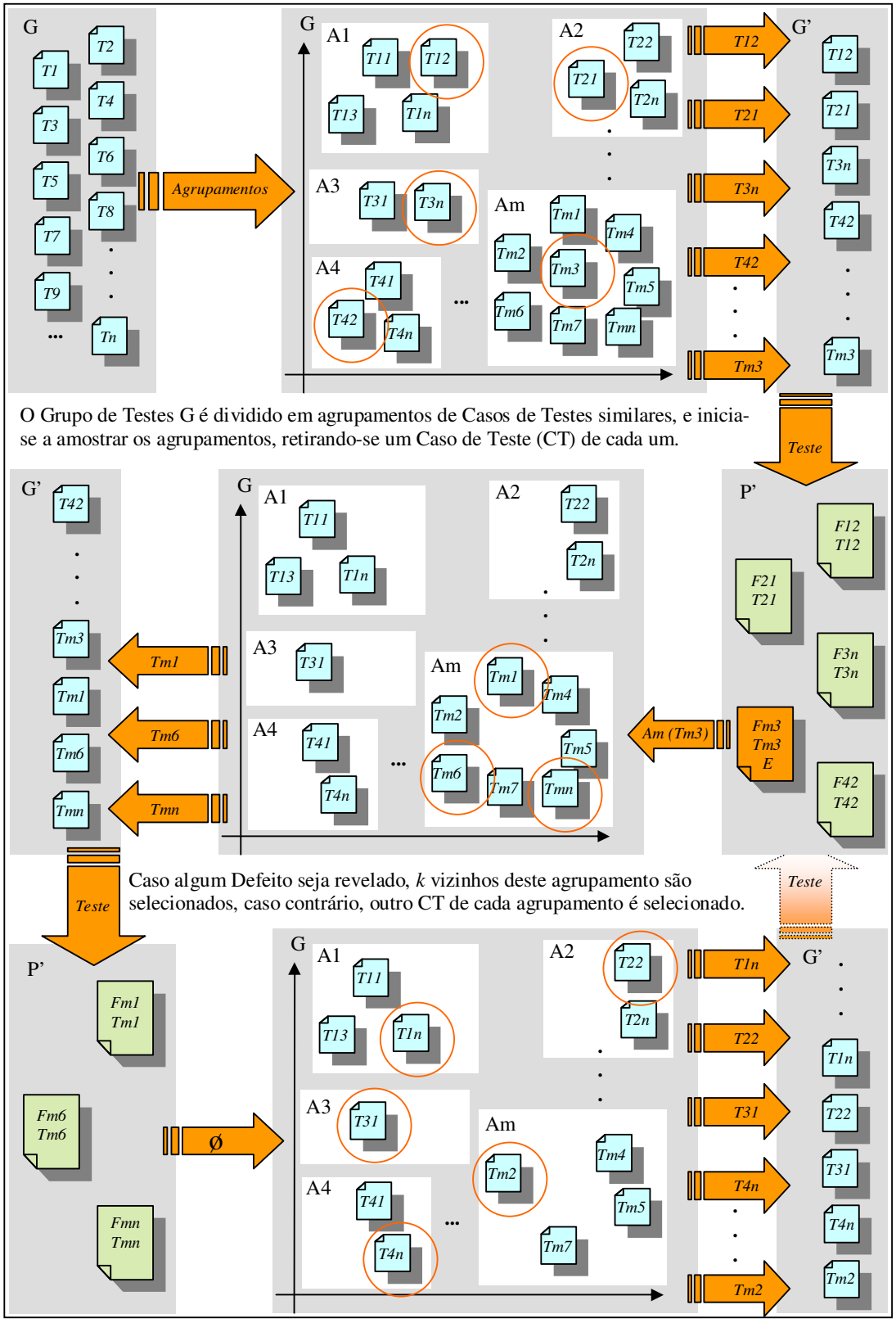


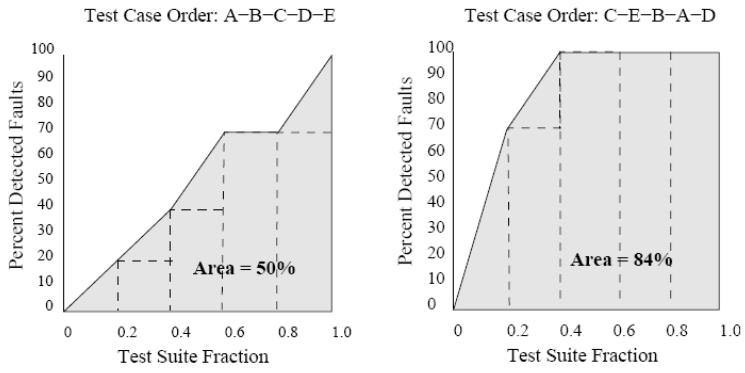
Figura 2.2: Fluxo do Método de Amostragem por Perseguição de Defeitos.

Definindo *G* como um grupo de testes contendo *n* casos de testes e *F* como um conjunto de *m* defeitos revelados por *G*, podemos definir GF_i como sendo o primeiro caso de teste ao ordenarmos *G'* de *G* que revela o defeito *i*. A MPDD para *G'* é dada pela fórmula 2.2.

$$MPDD = 1 - \frac{GF_1 + GF_2 + \ldots + GF_m}{nm} + \frac{1}{2n} \qquad (2.2)$$

Para ilustrar melhor este procedimento, considere um programa com um GT de cinco CT, de *A* a *E*. Este programa contém oito defeitos detectados por estes CT, como mostrado na Figura 2.3 (A). Considere duas ordens distintas para os CT, ordem *T1*: *A-B-C-D-E* e ordem *T2: C-E-B-A-D*. A Figura 2.3 (B&C) mostra a percentagem da detecção de defeitos em relação à fração do GT usado, para ambas as ordens respectivamente. As áreas dentro dos retângulos pontilhados representam os pesos em percentagem de defeitos detectados na fração correspondente do GT, enquanto as linhas sólidas conectando os vértices dos retângulos interpolam o ganho na percentagem de detecção de defeitos. A área total debaixo da curva, no entanto, representa o peso da MPDD durante o ciclo de vida do GT. A ordem de *T1* – Figura 2.3 (B) – produziu um MPDD de 50,0% e a ordem de *T2* – Figura 2.3 (C) – apresentou uma ordem de CT muito "mais eficiente" que *T1* (e de fato é a ordem ótima), com um MPDD de 84,0% .

test	fault									
	1	2	3	4	5	6	7	8	9	10
A	x				x					
B	x				x	x	x			
C	x	x	x	x	x	x	x			
D					x					
E								x	x	x



A. Test suite and faults exposed B. APFD for prioritized test suite T1 C. APFD for prioritized test suite

Figura 2.3: Medida MPDD (*APFD*). Extraído de (ELBAUM et. al., 2004)

2.3. Considerações Finais

O capítulo 2 visou construir a base de conhecimento necessária para o entendimento do problema a ser resolvido. Os assuntos foram apresentados de forma hierárquica iniciando pelo tema mais geral e comumente conhecido (Testes de Sistemas) pra culminar no tema mais específico (Técnicas Baseadas em Distribuição – Amostragem por Perseguição de Defeitos). Este último tema foi selecionado como base para o método proposta no Capítulo 4, que é ainda formado pela Tarefa de Agrupamento, descrita no capítulo a seguir.

O capítulo em questão, apesar de discorrer sobre um assunto bastante vasto como os Testes de Regressão (e por vezes até polêmicos), procurou apresentar os assuntos pertinentes à otimização de testes de regressão de forma objetiva e detalhada, assim como também oferecer embasamento para as tomadas de decisões da linha de pensamento apresentada.

Capítulo 3

Tarefa de Agrupamento

Uma vez apresentado todo o panorama geral de testes de sistemas e testes de regressão, que pode ser considerado a parte central do problema a ser resolvido – assunto do capítulo 2 – explora-se a seguir a parte central do método desenvolvido. Partindo-se do princípio que o método desenvolvido está baseado fortemente em uma técnica baseada em distribuição denominada amostragem por perseguição de defeitos, que por sua vez, é fortemente baseada em agrupamentos de Casos de Testes (CTs), este capítulo apresenta a Tarefa de Agrupamento.

De modo análogo ao capítulo 2, o presente capítulo encontra-se no formato de uma "árvore", apresentando uma sequência lógica de assuntos e sub-assuntos assim como seus contextos adjacentes, para que seja possível o entendimento das decisões feitas durante a geração do método desenvolvido.

O capítulo se inicia com a introdução à tarefa de agrupamento e, em seguida, detalha suas diferentes características. O capítulo ainda explana sobre os algoritmos Hierárquicos

Aglomerativos e descreve - em detalhes - o Agrupamento por Média Aritmética Ponderada (algoritmo usado pelo método desenvolvido).

Este capítulo ainda entra no assunto específico de Métricas de Dissimilaridade (MD), usadas pelas tarefas de agrupamento, e de maneira mais específica, a da Distância de Hamming que é a MD usada pelo método desenvolvido.

3.1. Introdução a Tarefa de Agrupamento

Uma tarefa de agrupamento consiste na ação de se formar grupos de elementos com base nas informações que os descrevem (STEINBACH et. al., 2005); (WITTEN e FRANK, 2005). Uma tarefa de agrupamento específica, portanto, pode ser definida como uma técnica de análise multi-variável realizada para encontrar agrupamentos de objetos similares dentro de uma população de objetos definidos cada um por um conjunto de variáveis. O objetivo da tarefa de agrupamento é dividir a população de objetos em grupos, de modo que elementos com representações multi-variáveis similares sejam organizados para um mesmo grupo, enquanto representações diferentes sejam organizadas em grupos distintos.

Na Figura 3.1 pode-se observar um exemplo de diferentes modos de agrupar-se um mesmo conjunto de elementos (Figura 3.1 (a)) que, no caso, podem ser vistos como dois agrupamentos de acordo com suas posições (Figura 3.1 (b)), quatro agrupamentos de acordo com o seu preenchimento (Figura 3.1 (c)), ou seis agrupamentos de acordo com suas formas geométricas (Figura 3.1 (d)). Como agrupá-los, depende dos critérios utilizados.

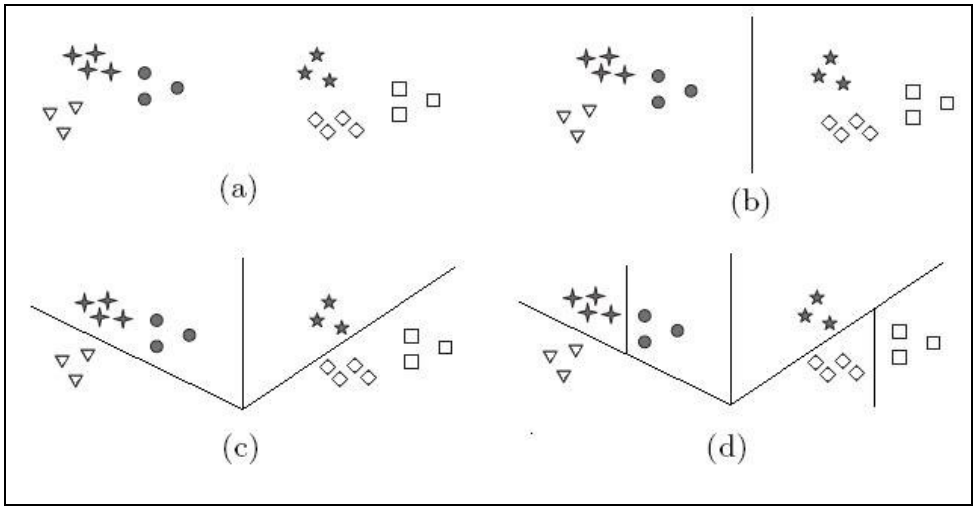


Figura 3.1: Diferentes modos de agrupar um mesmo conjunto de elementos.

Tipicamente uma tarefa de agrupamento é formada pelos seguintes passos (JAIN et. al., 1999):

1. Um padrão de representação que de alguma forma descreva cada objeto e que fornece a subsídio para a diferenciação dos mesmos (que pode – ou não – incluir a ação de extração de características e/ou seleção das mesmas);
2. A definição de uma Métrica de Dissimilaridade (MD) apropriada para o padrão de representação dos objetos. Esta será usada para determinar um valor quantitativo de quão similares os objetos do grupo são entre si;
3. A ação de agrupar os dados em seus determinados grupos;
4. A eventual necessidade de se abstrair os dados resultantes para que possam ser interpretados para o fim que lhes cabe;
5. E por fim, a possível ação de avaliar o resultado do agrupamento em termos de um "bom resultado" ou um "resultado ruim";

Os passos supracitados podem ser resolvidos de tantas formas diferentes que a ocorrência mais comum é de que se tenha uma técnica de agrupamento diferente para cada tarefa de agrupamento específica a ser implementada. A literatura inclusive apresenta uma coleção tão vasta de propostas - K-means, PAM, CLARANS, DBSCAN, CURE, ROCK, CHAMELEON (KARYPIS et. al., 1999) - que a escolha de uma solução específica para um determinado problema se torna uma decisão significativamente complexa (JAIN et. al., 1999).

3.1.1. Características dos Tipos de Agrupamentos

Antes mesmo de poder-se comentar sobre os algoritmos de agrupamento em si, faz-se necessário descrever as diferentes características dos conjuntos de objetos a serem agrupados. Na Figura 3.2 estão representados alguns dos tipos mais encontrados de conjuntos de objetos agrupados na prática. Os dados estão apresentados como pontos em um espaço bidimensional, porém, as definições são igualmente válidas para qualquer dimensão multi-variável mais complexa.

Agrupamentos Bem Espalhados: Representado na Figura 3.2(a) são agrupamentos formados por objetos que se encontram mais próximos (ou mais similares) aos outros objetos no mesmo agrupamento do que de qualquer outro objeto no conjunto. Adicionalmente, é possível que se use um limiar predeterminado para determinar o limite de proximidade entre os objetos.

Agrupamentos Baseados em Protótipos: São agrupamentos nos quais cada objeto está mais próximo (ou é mais similar) a um protótipo que define o agrupamento em questão. Muitas vezes o protótipo acaba por se tornar o ponto central do agrupamento, comumente denominado de Agrupamento Baseado no Centro com representado na Figura 3.2(b).

Agrupamentos Baseados em Continuidade: Os agrupamentos são definidos por objetos que estão conectados uns aos outros. Normalmente a conexão entre dois objetos é definida por uma distância específica entre eles, o que implica que num agrupamento baseado em continuidade cada objeto está mais próximo (ou mais similar) a algum outro objeto do agrupamento do que qualquer outro objeto do conjunto como mostrado na Figura 3.2(c). Esta definição é bastante útil ao se trabalhar com agrupamentos irregulares e intricados, porém, tornando a presença de ruído bastante crítica. Como pode ser observado também na Figura 3.2(c) é necessário que haja apenas uma estreita ponte entre dois agrupamentos para que estes sejam considerados apenas um.

Agrupamentos Baseados em Densidade: Um grupo de objetos é considerado um agrupamento se o mesmo estiver situado em uma região de maior densidade, cercado de uma área de menor densidade. Aplicando-se ruído proposital e homogêneo a imagem na Figura 3.2(c), forma-se uma nova imagem na Figura 3.2(d) onde o ruído supracitado torna-se irrelevante, baseando a análise não na sua contiguidade, porém, em sua densidade. Embora a curva observada na Figura 3.2(c) tenha se mesclado ao ruído e deixado de existir.

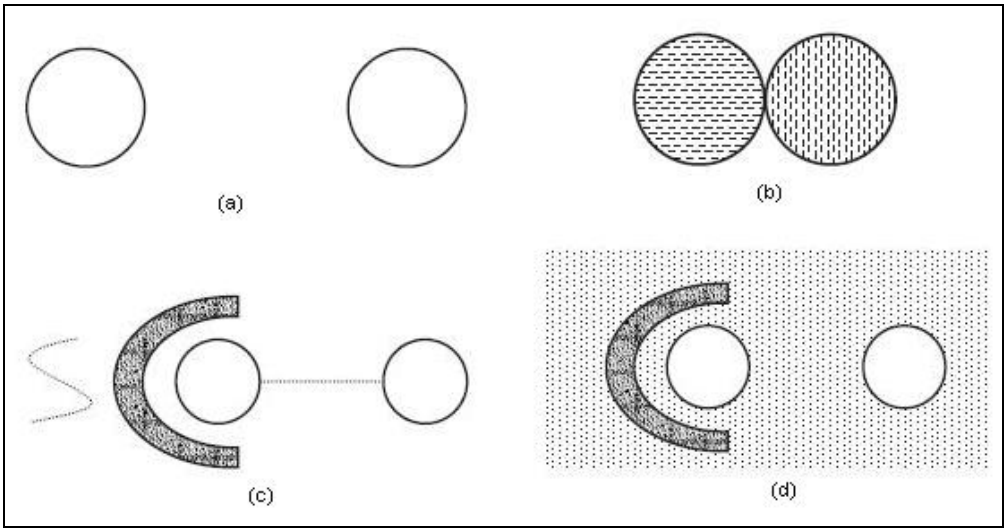


Figura 3.2: Tipos de agrupamentos. Extraído de (STEINBACH et. al., 2005)

3.1.2. Características dos Algoritmos de Agrupamento

Além das características apresentadas pelos agrupamentos, também se faz necessário avaliar as características dos algoritmos responsáveis por agrupá-los. Na tentativa de organizar a vastidão de diferentes técnicas e métodos relacionados à tarefa de agrupamento, alguns termos e conceitos surgiram para classificar os procedimentos.

Uma das características mais determinantes de um algoritmo de agrupamento consiste no modo em que este forma os agrupamentos, podendo estes ser formados hierarquicamente ou particionadamente (DICKINSON et. al., 2001a).

Agrupamento Particionado: Nesta forma de agrupamento cada objeto faz parte de apenas um agrupamento. A população é dividida em um número predeterminado de agrupamentos. Os agrupamentos são formados interativamente, reorganizando objetos aos diferentes agrupamentos de acordo com os critérios de similaridade.

Agrupamento Hierárquico: O Agrupamento hierárquico introduz o conceito de sub-agrupamentos, podendo os mesmos serem organizados em "árvores" denominadas dendogramas. Cada "nó da árvore" (exceto para os "nós folhas") contém a união de seus sub-agrupamentos e a "raiz da árvore" é o agrupamento contendo todos os objetos (STEINBACH et. al., 2005). O processo é realizado passo-a-passo, podendo ser considerado aglomerativo ou divisivo. O processo aglomerativo inicialmente aloca cada objeto em um agrupamento diferente e então começa a unir agrupamentos minimamente dissimilares até restarem um número predeterminado de agrupamentos. O processo divisivo ocorre de maneira contrária. Inicialmente todos os objetos são alocados a um agrupamento e a cada passo o agrupamento com mais dissimilaridades é dividido em dois agrupamentos distintos até que se tenha um número de agrupamentos predeterminado.

As técnicas hierárquicas são mais rápidas que as particionadas, porém possuem a desvantagem de que uma vez um agrupamento unido ou dividido este processo não pode ser desfeito (diferindo do particionado, cujo processo pode ser revertido a cada iteração).

Outra característica delimitadora encontrada nos diferentes algoritmos está relacionada à exclusividade dos objetos em relação aos agrupamentos a que estão designados estes podendo ser (STEINBACH et. al., 2005):

Agrupamentos Exclusivos: Onde cada objeto pertence a apenas um agrupamento em um determinado momento;

Agrupamentos Sobrepostos: Podendo os objetos pertencerem a mais de um agrupamento simultaneamente;

Agrupamentos Probabilísticos (*Fuzzy*): Onde todos os objetos pertencem a todos os agrupamentos em um nível de aderência que varia de 0 (absolutamente não pertence) a 1 (absolutamente pertence).

Os algoritmos de agrupamento ainda podem ser classificados em relação à sua completude como:

Agrupamento Completo: Havendo a necessidade de que todos os objetos estejam organizados a pelo menos um agrupamento, não podendo existir a possibilidade de algum objeto permanecer não-agrupado;

Agrupamento Parcial: Onde a necessidade de cobrir todos os objetos do grupo não existe, permitindo que objetos permaneçam não-alocados a algum agrupamento.

Por fim, ainda há as diferenças de homogeneidades:

Agrupamento Heterogêneo: Que forma e trabalha com agrupamentos de tamanhos, formatos e/ou densidades completamente diferentes.

Agrupamento Homogêneo: Forma e trabalha com agrupamentos de dimensões iguais ou similares;

3.1.3. Seleção de uma Técnica de Agrupamento

Observando os passos existentes em uma tarefa de agrupamento e as várias possibilidades existentes para resolver cada um destes, percebe-se que "selecionar" uma técnica de agrupamento para resolver um determinado problema na verdade se resume a selecionar um modo de implementar cada um de seus passos e montá-los em uma técnica específica para o problema em questão.

Selecionando um Padrão de Representação: Este passo é certamente o que apresenta a menor possibilidade de se usar algum método já consagrado, pois é totalmente dependente do formato dos dados a mão, muitas vezes oferecendo pouco ou nenhum controle direto sobre como são adquiridos. Não há como escapar de uma análise cuidadosa dos dados disponíveis para se chegar a um resultado satisfatório. Uma transformação bem refletida sobre os dados pode gerar grandes ganhos para o resultado do agrupamento, enquanto uma transformação imprudente pode inviabilizar a atividade por completo. Um bom objetivo a se perseguir neste passo é gerar um padrão de representação que seja simples de ser compreendido. Uma

representação complexa pode vir a dificultar ou até mesmo impossibilitar, discernir os diferentes agrupamentos (JAIN et. al., 1999).

Um padrão de representação pode ser usado para descrever qualquer objeto, físico ou abstrato, porém o modo mais comum de representá-lo é através de um vetor multidimensional onde cada característica do objeto é traduzida em uma de suas dimensões, quantitativamente – como peso, quantidade e tempo - ou qualitativamente - como "leve" ou "pesado", "pouco" ou "muito" e "lento" ou "rápido".

Selecionando uma Medida de Similaridade: Como a similaridade entre os objetos é algo fundamental para a tarefa de agrupamento, esta é uma escolha bastante crítica para a técnica resultante. Devido à possível grande variedade de diferentes naturezas entre as características a serem comparadas entre os objetos, à distância entre estas (ou as distâncias) precisam ser avaliadas cuidadosamente. O mais comum na realidade é de se medir a dissimilaridade entre os objetos, ou seja, a magnitude da diferença entre os objetos e não semelhança entre eles (ABONYI e FEIL, 2007).

A Métrica de Dissimilaridade (MD) mais popular encontrada na literatura de agrupamentos é a Distância Euclidiana Multidimensional que pode ser usada para calcular a distância euclidiana entre dois vetores multidimensionais *Xi* e *Xj*, aplicando-se diretamente a fórmula 3.1 aos dados em forma de vetores.Na fórmula, *k* indica cada posição dos vetores multidimensionais (JAIN et. al., 1999).

$$d_2(\mathrm{X}_i, \mathrm{X}_j) = \left(\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2 \right)^{1/2} = \left\| \mathrm{X}_i - \mathrm{X}_j \right\|_2 \tag{3.1}$$

Outras transformações podem vir a ser desejáveis antes de se aplicar a fórmula diretamente como: A **Métrica Binária** que substitui valores de atributos diferentes de zero por 1, objetivando dar ênfase a existência desse valor e não a diferença de sua magnitude; A **Métrica Proporcional** que é a normalização de cada atributo, sendo que a variação de cada atributo é computada e então cada valor é mapeado para a sua posição no espectro; **Métrica Desvio Padrão** que também normaliza os atributos, porém o desvio padrão de cada atributo é computado e cada valor de atributo é então dividido por este; e a **Métrica Histograma** que para cada atributo existe um histograma indicando a frequência relativa de diferentes valores de atributos, sendo que cada valor de atributo é então mapeado para 1 menos a frequência

relativa correspondente (isso é feito para enfatizar valores de atributos que ocorrem mais raramente). Além destas aplicações isoladas, é possível recorrer também ao uso de qualquer combinação destas.

Selecionando um Algoritmo de Agrupamento: Como já citado anteriormente, a variedade de algoritmos de agrupamentos na literatura é imensa. Para se chegar a uma escolha satisfatória para este passo da técnica de agrupamento, é necessário basear-se nas já descritas características inerentes aos algoritmos de agrupamento, e confrontá-las às características dos dados a serem agrupados. Dessa forma é possível restringir o número imenso de algoritmos a um grupo menor e mais específico.

3.2. Agrupamentos Hierárquicos Aglomerativos

Aprofundando-se no conceito de Algoritmos de Agrupamento Hierárquicos Aglomerativos - citados na Seção 3.1.1 - percebe-se que cada algoritmo específico pertencente a esta categoria, apesar de possuir suas particularidades, baseia-se no fato de que: inicia seu processamento com agrupamentos de apenas um objeto (todos os objetos individualmente são considerados um agrupamento), e sucessivamente agrupa os dois agrupamentos mais apropriados (critérios que diferem de algoritmo para algoritmo) até que reste apenas um agrupamento. Esta estratégia pode ser representada mais formalmente por:

1. Calcula-se a matriz de similaridade, se necessário;
2. Agrupam-se os agrupamentos mais apropriados (critério pré-definido);
3. Atualiza-se a matriz de similaridades com os valores do agrupamento recém formado;
4. Repete-se o passo 2 em diante até que reste apenas um único agrupamento.

Alguns dos principais representantes dentre os algoritmos hierárquicos aglomerativos estão descritos a seguir.

3.2.1. Agrupamento por Ligação Simples

No algoritmo de agrupamento por ligação simples (JAIN e DUBES, 1988) dois agrupamentos são fundidos quando os dois objetos (em agrupamentos diferentes) mais próximos (ou mais similares) de todo o conjunto de objetos são encontrados. Representado na Figura 3.3(a).

O algoritmo funciona muito bem para agrupar áreas não-elípticas do espaço de dados, mas é muito sensível a ruídos. O ruído normalmente altera sensivelmente o valor da

similaridade entre os objetos, e com o algoritmo em questão usa apenas uma informação pontual para tomar suas decisões, a ordem em que os agrupamentos são unidos pode ser facilmente corrompida.

3.2.2. Agrupamento por Ligação Completa

Também conhecido como "análise do vizinho mais distante" representa quase o conceito oposto do apresentado no Agrupamento por Ligação Simples (JAIN et. al., 1999). Neste algoritmo a fusão entre dois agrupamentos se baseia na maior proximidade (na maior similaridade) entre os objetos mais distantes (ou menos similares) de dois agrupamentos, como mostrado na Figura 3.3(b).

Este algoritmo freqüentemente produz agrupamentos maximamente conectados e agrupamentos esféricos (ao contrário do algoritmo de ligação simples que normalmente produz sequências e correntes), ótimo para enfatizar descontinuidades, mas apropriado apenas para estes fins específicos.

3.2.3. Agrupamento por Ligação Intermediária

O algoritmo por ligação intermediária tenta resolver o problema de ruído do algoritmo de ligação simples (STEINBACH et. al., 2005), estipulando um número mínimo de objetos que precisam ser identificados como mais próximos (ou mais similares) entre dois agrupamentos, para que estes estejam aptos a se fundirem, caso representado na Figura 3.3(c).

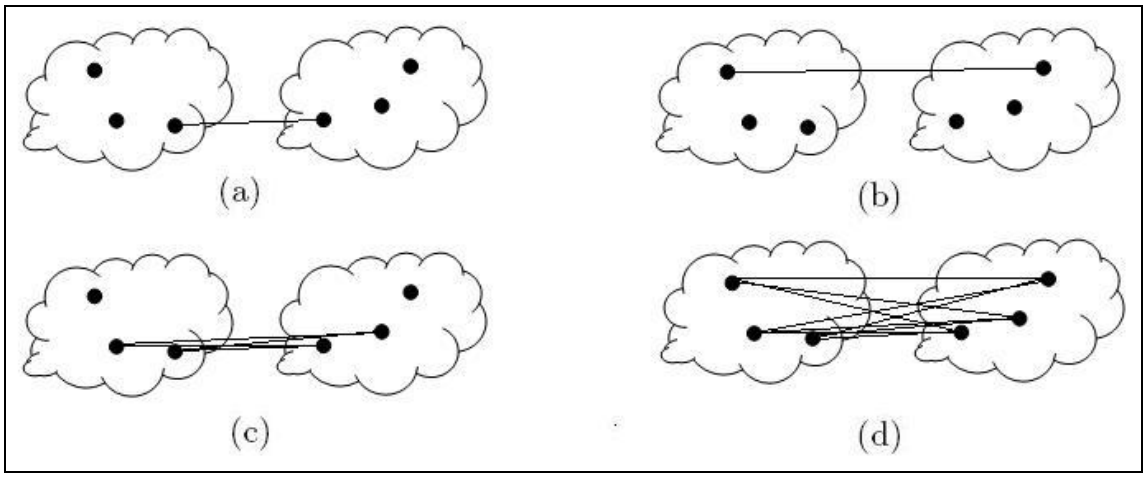


Figura 3.3: Tipos de Ligações entre elementos de agrupamentos.

Também conhecido na literatura como Ligação Proporcional o número limite pode não ser exatamente um valor predeterminado, mas uma proporcionalidade. Se, por exemplo, a proporção de ligações escolhida for C = 0,5, significa que 50% dos objetos alocados a dois agrupamentos precisam estar mais próximos (ou serem mais similares) do que os 50% dos objetos em outros agrupamentos para que possam se fundir.

3.2.4. Agrupamentos por Média Aritmética Direta e Ponderada

Já mais eficiente do que o algoritmo por ligação intermediária são os algoritmos classificados como agrupamentos por média do grupo (STEINBACH et. al., 2005), que baseiam a decisão de fundir agrupamentos em um cálculo que envolve as proximidades (ou similaridades) entre todos os objetos de ambos os agrupamentos analisados como mostrado na Figura 3.3(d). No caso do Agrupamento por Média Aritmética Direta (*UPGMA – Unweighted Pair-Group Method using Arithmetic Average*), a maior proximidade (ou maior similaridade) é encontrada pelo cálculo da média aritmética - fórmula 3.2 - de todas as distâncias entre os objetos de um dos agrupamentos em relação aos objetos do outro agrupamento. Os agrupamentos que apresentarem o resultado de maior proximidade (ou similaridade) são fundidos. Na fórmula, *n* indica o número de elementos em cada agrupamento e *X* cada elemento do vetor.

$$d(A,B) = \frac{1}{n_A n_B}\left(\sum_{i=1}^{n_A}\left(\sum_{j=1}^{n_B} dist\left(x_{A_i}, x_{B_j}\right)\right)\right) \tag{3.2}$$

Uma variação interessante do Agrupamento por Média Aritmética Direta é o Agrupamento por Média Aritmética Ponderada (*WPGMA – Weighted Pair-Group Method using Arithmetic Average* ou Método de McQuitty). É bastante frequente que os agrupamentos de objetos durante o processo de agrupamento formem grupos de tamanhos significativamente diferentes. Infelizmente, a presença de grandes quantidades de objetos similares pode vir a distorcer os resultados da média aritmética direta, diluindo pequenas diferenças no todo. Como eliminar objetos dos agrupamentos para equalizar seus tamanhos raramente é uma opção (pois significaria muitas vezes descartar informação valiosa), a média aritmética ponderada diminui a influência do agrupamento maior ao associar pesos iguais aos dois ramos que potencialmente irão se fundir - fórmula 3.3. Na fórmula *X* indica cada elemento do vetor.

$$d(A,B) = \frac{1}{2}\left(\sum_{i=1}^{n_A}\left(\sum_{j=1}^{n_B} dist\left(x_{A_i}, x_{B_j}\right)\right)\right) \tag{3.3}$$

3.3. Distância de Hamming

A distância de Hamming é uma forma de se medir a distância entre dois vetores multidimensionais de mesmo tamanho, sendo que o resultado – um número inteiro – indica o número de posições dos dois vetores que contém informações diferentes (HAMMING, 1950). Em outras palavras, mede o número de substituições que seriam necessárias para transformar um dos vetores no outro.

Por exemplo, a distância de Hamming entre as sequências de letras "máximo" e "mínimo" é 2, pois para que uma palavra se transforme na outra é necessário que as segundas e as terceiras letras sejam igualadas "áx" e "ín".

Se imaginarmos também um vetor binário de 4 posições, facilmente pode-se chegar ao hiper-cubo representado na Figura 3.4, que conecta todas as possibilidades de valores dos vetores com uma posição alterada. Basta que se contem as arestas do cubo para se ter a distância entre duas instâncias do vetor. O caminho superior tem distância 1 enquanto o caminho inferior tem distância 3.

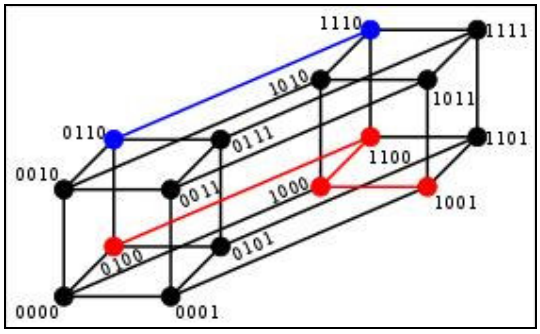


Figura 3.4: Representação do Hiper-cubo das distâncias de Hamming.

O hiper-cubo para sequências maiores pode vir a ser bastante complexo, mas o processo "passo-a-passo" para obtê-lo é simples. Basta que se compare dois vetores, posição a posição, e se enumere aquelas que são diferentes. Repetindo esse processo para todas as

combinações possíveis do vetor binário de 4 posições, por exemplo, tem-se a matriz simétrica das distâncias de Hamming representada na Figura 3.5.

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	0	1	1	2	1	2	2	3	1	2	2	3	2	3	3	4
0001	1	0	2	1	2	1	3	2	2	1	3	2	3	2	4	3
0010	1	2	0	1	2	3	1	2	2	3	1	2	3	4	2	3
0011	2	1	1	0	3	2	2	1	3	2	2	1	4	3	3	2
0100	1	2	2	3	0	1	1	2	2	3	3	4	1	2	2	3
0101	2	1	3	2	1	0	2	1	3	2	4	3	2	1	3	2
0110	2	3	1	2	1	2	0	1	3	4	2	3	2	3	1	2
0111	3	2	2	1	2	1	1	0	4	3	3	2	3	2	2	1
1000	1	2	2	3	2	3	3	4	0	1	1	2	1	2	2	3
1001	2	1	3	2	3	2	4	3	1	0	2	1	2	1	3	2
1010	2	3	1	2	3	4	2	3	1	2	0	1	2	3	1	2
1011	3	2	2	1	4	3	3	2	2	1	1	0	3	2	2	1
1100	2	3	3	4	1	2	2	3	1	2	2	3	0	1	1	2
1101	3	2	4	3	2	1	3	2	2	1	3	2	1	0	2	1
1110	3	4	2	3	2	3	1	2	2	3	1	2	1	2	0	1
1111	4	3	3	2	3	2	2	1	3	2	2	1	2	1	1	0

Figura 3.5: Matriz de distâncias de Hamming

3.4. Considerações Finais

Este capítulo teve o intuito de descrever a tarefa de agrupamento, tão necessária para o mecanismo do método desenvolvido, assim como os detalhes do algoritmo hierárquico aglomerativo, o Agrupamento por Média Aritmética Ponderada e a Distância de Hamming, partes do método desenvolvido por este estudo.

Após o capítulo 2 descrever todos os detalhes sobre testes de regressão e o capítulo 3 apresentar as informações necessárias sobre tarefas de agrupamento, tem-se base suficiente para iniciar a discussão sobre a metodologia de pesquisa e o método desenvolvido – assunto do capítulo 4 deste documento.

Capítulo 4

Método Desenvolvido

O capítulo que aqui se inicia tem por finalidade descrever a metodologia de pesquisa utilizada e o método desenvolvido. Todo o embasamento teórico necessário para o entendimento das origens do método de Priorização de Casos de Testes de Regressão usando Amostragem por Perseguição de Defeitos foi discutido no capítulo 2 – Testes de Sistemas e Testes de Regressão – e no capítulo 3 – Tarefa de Agrupamento, deixando para o capítulo 4 a missão de descrever o método propriamente dito.

A aplicação empírica do método e os resultados obtidos serão abordados no capítulo 5 - Resultados Obtidos.

4.1. Metodologia Utilizada

De todas as metodologias de Testes de Regressão Otimizados (TRO) estudadas, optou-se por usar uma técnica de Priorização de Casos de Teste (PCT), por estas apresentarem uma grande independência dos recursos disponíveis para os testes.

Analisando-se as demais metodologias, conclui-se que as técnicas de Seleção de Testes de Regressão (STR), apresam um grande potencial para reduzir o número de CT usados, mas deixam a desejar em relação às PCT, pois mesmo o GT sendo reduzido não é garantido que este caiba em uma determinada janela de tempo. Já as técnicas de Redução de Grupo de Testes (RGT) poderiam até mesmo ser adotada em paralelo às outras, porém como técnica selecionada fugiria do objetivo proposto, por eliminar permanentemente CT do GT.

Por sua vez, as técnicas de PCT podem ser aplicadas sem impactos permanentes ao GT, pois apenas os ordena de diferentes formas conforme exigir o caso. Estas técnicas se adaptam a qualquer necessidade de tempo disponível, devido ao fato de os CT mais importantes serem executados no início. Mesmo que os Testes de Regressão (TR) acabem por ser terminados de maneira abrupta, tem-se a convicção de que o tempo utilizado foi aproveitado da melhor forma possível.

Dentre as técnicas de PCT, optou-se pelo uso de uma Técnica Baseada em Distribuição (TBD). Já foi demonstrado que, apesar de as Técnicas Baseadas em Cobertura (TBC) e TBD possuírem um desempenho bastante similar, para programas maiores as TBD demonstram um desempenho superior (LEON e PODGURSKI, 2003).

4.1.1. Metodologia de Desenvolvimento do Processo de Priorização

A metodologia utilizada – definida durante a fase de Projeto de Dissertação de Mestrado - diz respeito aos passos que foram planejados para adaptar o mecanismo de priorização e à sua validação:

1. Definição formal da adaptação da APD. O primeiro passo baseou-se na própria definição formal da técnica de priorização por Amostragem por Perseguição de Defeito, onde as suas fases de agrupamento e amostragem foram adaptadas. As adaptações foram baseadas em conhecimentos adquiridos durante os créditos e na literatura levantada.
2. Base de dados de testes. Com a técnica formalizada, o próximo passo planejado diz respeito à seleção da base de dados para os testes de validação. A base de dados – com características bastante específicas – contendo dados detalhados, organizados em forma de versões, informações sobre as modificações de código que ocorreram entre uma versão e suas versões adjacentes, listas de testes que possam ser usadas para relacionar os defeitos às modificações de código e assim por diante.

3. Levantamento de atributos dos casos de testes (CT). Nesta fase ocorreu uma análise detalhada das informações disponíveis nos CT e se definiu os atributos usados para compor os critérios pelos quais os elementos são priorizados, formando suas representações formais.
4. Validação do Mecanismo de Agrupamento e Amostragem. A fase em questão representa as atividades geração de dados preliminares usando a técnica adaptada e a comparação destes com os resultados gerados por não usar uma técnica específica – deixando os CT serem executados em sua ordem original, a execução aleatória dos CT e os resultados gerados ao se executar a técnica original. O objetivo foi confirmar o potencial de melhora de eficiência da adaptação.
5. Calibração e Ajustes do Mecanismo de Agrupamento e Amostragem. Compreende uma geração mais extensa de dados de priorização para que os impactos das alerações do parâmetros *m* e *k* possam ser avaliados e ajustados para uma melhor eficiência da técnica.

4.2. Introdução ao Método Desenvolvido

O método descrito neste documento - Priorização de Casos de Testes de Regressão usando Amostragem por Perseguição de Defeitos - foi desenvolvido a partir da já existente e eficiente técnica de Amostragem por Perseguição de Defeitos (APD), adaptada para melhorar a eficiência da atividade de Priorização de Casos de Testes (PCT), para um grupo particular dos Testes de Sistemas (TS), denominados Testes de Regressão (TR).

De todas as metodologias de Testes de Regressão Otimizados (TRO) estudadas - descritas na Seção 2.1.3 - optou-se por escolher uma técnica de Priorização de Casos de Teste (PCT), por esta reunir o melhor conjunto de características para manter o método desenvolvido como uma solução genérica.

Primeiramente os métodos de PCT podem ser aplicados a um Grupo de Testes (GT) sem impactos permanentes ao(s) Caso(s) de Teste(s) (CT)s, pois apenas alteram a ordem em que os mesmos são executados conforme exigido por critérios momentâneos. Adicionalmente, as PCTs também se adaptam diretamente a diferentes janelas de tempo, sendo que os CTs mais significativos serão, teoricamente, sempre executados no início. Se houver qualquer necessidade - inclusive súbita - de se restringir o tempo ou o tamanho do GT, ou até mesmo se

os TR acabarem por terminar de maneira abrupta, tem-se a convicção de que o tempo utilizado foi aproveitado da melhor forma possível.

A adaptação realizada na técnica de Amostragem por Perseguição de Defeitos está baseada em um conjunto adicional de informações com o qual a técnica original não pode contar - e portanto, não está preparada para usar - que são as informações resultantes dos testes de sistemas realizados em uma versão anterior do sistema a ser testado. Como descrito na Seção 2.1.2, quando se trata de testes de regressão está implícita a informação de que os testes estão sendo refeitos em uma versão posterior do sistema, para assegurar que o programa que já funcionava no passado continue funcionando na versão atual.

Parte-se então do princípio que uma versão específica de um sistema esteja sendo utilizado para o determinado fim a que se propõe. Quando um defeito é encontrado, conclui-se que não existiu um Caso de Teste (CT) presente no Grupo de Testes (GT) que tenha sido capaz de identificar este defeito específico. Torna-se necessário então que:

a) Ou um novo CT seja criado,
b) Ou um CT já existente seja alterado para que o GT possa ser capaz de identificar o defeito específico citado.

Este CT novo (ou adaptado) pode então, inclusive, ser usado para confirmar que o defeito não mais pode ser revelado, o que valida a correção efetuada.

O CT novo (ou adaptado) deve também ser armazenado para que, no futuro, possa garantir que as próximas versões do sistema continuem livres deste defeito específico. Porém, na versão que deve ser produzida para que exista uma versão livre do defeito atual, o CT recém comentado já garantiu que o defeito foi eliminado e não precisa ser utilizado como TR, visto que estes têm o objetivo de encontrar eventuais efeitos colaterais que a correção do defeito possa ter causado.

Por outro lado, é bastante provável que se houver algum efeito colateral nesta nova versão do sistema, a probabilidade deste efeito estar ligado à área de código que foi modificada é bastante elevada (áreas de código que não foram modificadas devem se comportar sem problemas).

4.2.1. Amostragem por Perseguição de Defeitos para Testes de Regressão

A idéia de se adaptar a técnica de APD Original para TRs tem por objetivo direcionar os testes para a área de código que foi modificada. Para tal, duas alterações principais foram

introduzidas na APD Original, transformando-a em uma APD exclusiva para regressão. Uma modificação foi feita na Fase de Agrupamento do método e outra na Fase de Amostragem.

Durante a Fase de Agrupamento: todo o CT alterado ou criado para confirmar a efetividade da correção dos defeitos da versão em questão - as melhores evidências em termos de CT em relação à área de código que foi alterada - será usado como ponto de partida. Estes CTs não serão usados durante os TR, mas fornecerão suas descrições para formarem agrupamentos que os contenham. Estes agrupamentos contendo os CT recém incorporados serão considerados como agrupamentos que tendem a conter CT ligados às áreas de código modificadas (LEON e PODGURSKI, 2003).

Durante a Fase de Amostragem: durante a primeira coleta de amostras (em que normalmente "*k*" vizinhos não são selecionados, e sim, um CT é retirado de cada agrupamento), esta terá seu comportamento alterado. Apesar de os CT recém incorporados não fazerem parte da execução dos TR propriamente ditos, estes serão considerados como tendo falhado na execução dos TR para a versão anterior (o que seria um fato se os mesmos já existissem no período em questão), e já na primeira coleta de amostras, "*k*" vizinhos de seus respectivos agrupamentos são selecionados para o GT. Esta alteração é responsável por aumentar a densidade de CT ligados às áreas de código modificadas.

Novamente, lançando-se mão da formalidade, volta-se a definir um grupo finito de CTs de *T1* a *Tn* denominado *G*, porém também um grupo de CTs *Tα* a *Tω* que representam os CTs que foram criados ou alterados por causa da versão anterior do sistema.

O funcionamento da técnica de priorização de casos de testes APD específica para TR pode ser exemplificado através dos seguintes passos (as alterações para a técnica original estão evidenciadas em **negrito**) – também representados na Figura 4.1:

1. Separa se *G* em um número *m* de agrupamentos onde todos os CTs de *T1* a *Tn* (**que incluem os CTs *Tα* a *Tω***) são separados em agrupamentos que evidenciem suas similaridades quanto ao código que testam;

1a. **Com os agrupamentos formados, inicia-se o método com um Passo 4 adaptado que parte do princípio que os CTs *Tα* a *Tω* evidenciaram defeitos – na versão anterior - *k* "vizinhos" de *Tα* a *Tω* são coletados de seus agrupamentos;**

1b. **Executam-se os CTs recém adicionados a *G'* sobre *P';***

1c. **O Passo 1a é repetido até que os agrupamentos originais de *Tα* a *Tω* tenham sido esgotados;**

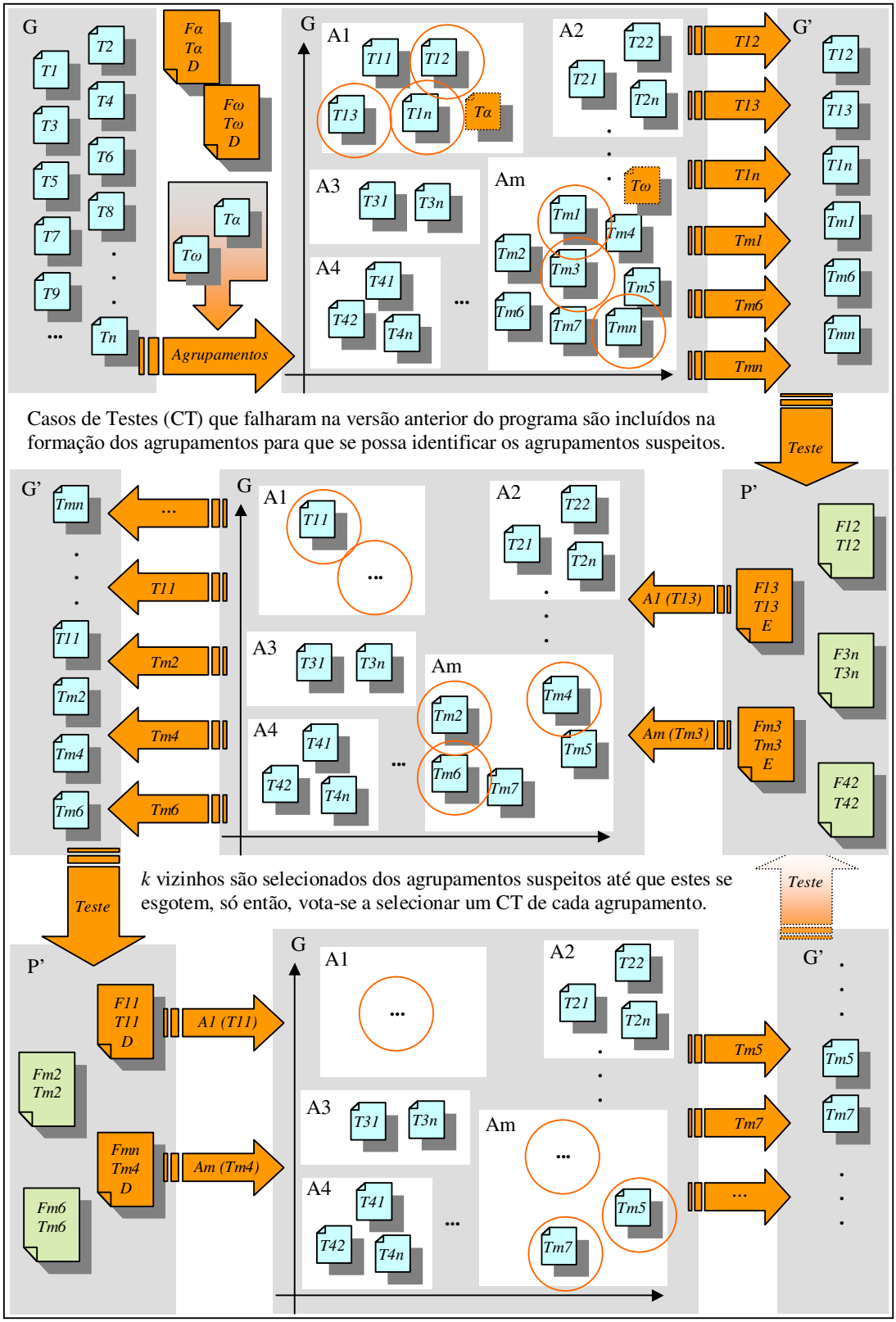


Figura 4.1: Fluxo do Método Desenvolvido

2. Coleta-se aleatoriamente *n* CTs de cada agrupamento (**dos agrupamentos restantes que não continham nenhum dos CTs *Tα* a *Tω***), que são então designados como testes da lista ordenada *G'*;
3. Executam-se os CTs recém adicionados a *G'* sobre *P'*;
4. Caso haja algum *Tx* em que evidencie algum defeito em *P'*, *k* "vizinhos" de *Tx* são coletados de seu antigo agrupamento (os *k* vizinhos coletados são os que possuem a maior similaridade com *Tx*), e o Passo 3 é repetido. O Passo 4 é repetido com qualquer CT que evidencie algum defeito até que nenhum defeito seja mais evidenciado;
5. Repete-se toda a sequência do Passo 2 em diante até que todos os CT tenham sido coletados formando *G'*.

4.3. Considerações Finais

O presente capítulo teve como objetivo descrever a metodologia de pesquisa utilizada para se chegar ao método desenvolvido, assim como todos os detalhes do método desenvolvido propriamente dito. A descrição do método – Priorização de Casos de Testes de Regressão usando Amostragem por Perseguição de Defeitos – consiste em especificar quais as alterações necessárias para adaptar a já conhecida técnica de Amostragem por Perseguição de Defeitos, exclusivamente para o uso em Testes de Regressão.

Em seguida – no Capítulo 5 – descreve-se os passos necessários para utilização empírica do método, deixando bastante claro quais são as partes que são consideradas genéricas – podendo ser utilizadas em qualquer outro cenário de Testes de Regressão – e quais são as partes que precisariam de algum tipo de adaptação para serem reutilizadas. Também são descritos em detalhes os resultados obtidos.

Capítulo 5

Resultados Obtidos

A sequência de capítulos anteriores forma uma trilha que consiste em todo o embasamento teórico necessário para o entendimento e a geração do método desenvolvido, assim como o método propriamente dito, culminando no capítulo em questão onde são discutidos: sua aplicação e os resultados obtidos.

O capítulo 5 tem por objetivo descrever de que forma o método desenvolvido foi utilizado empiricamente e os resultados obtidos independentemente de suas eficiências, apresentando os fatos encontrados e as conclusões lógicas atingidas, deixando para o capítulo 6 a tarefa da conclusão geral.

Os resultados obtidos estão dispostos na ordem cronológica em que foram obtidos, mantendo-se assim a descrições das impressões e conclusões conseguidas com a análise de cada passo seqüencial.

O capítulo ainda inclui uma seção dedicada à discussão das dificuldades encontradas durante a utilização do método desenvolvido, assim como uma sessão dedicada a outros trabalhos relacionados com o estudo realizado.

5.1. Bases de Teste e Ferramentas de Apoio

Sendo fato que partes do método desenvolvido são formadas por técnicas e métodos já existentes na literatura, optou-se por buscar algumas ferramentas e dados externos para complementar as atividades que deram origem ao método de Priorização de Casos de Testes de Regressão Usando Amostragem por Perseguição de Defeitos.

5.1.1. Infra-estrutura de Repositório de Artefatos de Software ("*SIR*")

Independente da técnica ou método utilizados para se efetuar um estudo na área de otimização de testes de sistemas – ou muitas vezes, até mesmo independente desta área específica – é imprescindível que se tenha uma base de dados razoavelmente extensa e bastante específica como "objeto de estudo". Neste caso é necessário que se tenha dados detalhados, organizados na forma de um programa / sistema (ou mais de um), com dados "históricos" organizados em versões, informações sobre as modificações de código que ocorreram entre uma versão e suas versões adjacentes, listas de testes que possam ser usadas para relacionar os defeitos às modificações de código e assim por diante.

Com o avanço das iniciativas de software abertos fica gradativamente mais fácil encontrar dados para esse tipo de estudo, como os compiladores C GNU (DICKINSON et. al., 2001b) ou GCC (LEON e PODGURSKI, 2003), porém, a comunidade de estudos de otimizações de testes de sistemas já vem realizando estudos nesta área antes disto (ROTHERMEL e HARROLD, 1997), (ROTHERMEL et. al., 2001), (ROTHERMEL et. al., 2004), e portanto, mantém um repositório "público" (é necessário efetuar um cadastro a priori) chamado Infra-estrutura de Repositório de Artefatos de Software (ou *SIR – Software-artifacts Infrastructure Repository*) (ROTHERMEL et. al., 2009). A base de dados em questão é bastante detalhada e direcionada para a área de testes de sistemas, onde pode-se encontrar armazenados um grupo razoável de programas, contendo todo o tipo de informações necessárias para realizar estudos na área de testes de sistemas. O repositório está organizado por programas, tanto para arquitetura SUN quanto Linux, contendo dados elaborados como: diferentes versões de códigos, traces, matrizes de defeitos, indicadores de cobertura, facilitando que haja uma mínima possibilidade de comparações de desempenho e eficiência entre os vários métodos e estudos realizados.

Dentre os programas armazenados no repositório existe um conjunto de programas simples denominados Aplicativos Siemens – Tabela 5.1 - (chamados assim por terem sido primeiramente usados por pesquisadores ligados a empresa em questão), que são encontrados com bastante frequência desde os primeiros estudos na área (ROTHERMEL e HARROLD, 1997), até estudos bem mais recentes (YOO et. al., 2009), os detalhes destes trabalhos são citados na Seção 5.5.

Programas	Linhas de Código Executável	Número de Versões	Número de Testes
print_tokens	402	7	4130
print_tokens2	483	10	4115
replace	516	32	5542
schedule	299	9	2650
schedule2	297	10	2710
Tcas	138	41	1608
tot_info	346	23	1052

Tabela 5.1: Lisa de Aplicativos Siemens.

5.1.2. Ferramenta para Análises de Agrupamento - PermutMatrix

Apesar da relativa variedade de ferramentas para análise de agrupamentos disponíveis - WEKA (GARNER, 1995), gCluto (KARYPIS, 2009), TreeView (EISEN, 2009) - nenhuma chegou a reunir as características específicas necessárias para este estudo tão satisfatoriamente quanto à ferramenta de análise de agrupamentos denominada PermutMatrix (GASCUEL, 2009) - Figura 5.1.

As características mencionadas como diferenciais da ferramenta são:

Resultado Final Detalhado: apesar de as várias ferramentas disponíveis apresentarem uma poderosa variedade de análises de agrupamentos contendo vários algoritmos distintos totalmente configuráveis, a maioria delas mantém-se limitada quanto ao resultado final apresentado. As ferramentas em geral são quantitativas, informando quantos elementos – e as percentagens destes - que foram direcionados para cada agrupamento, porém não exatamente fornecendo uma trilha indicando exatamente qual elemento específico foi designado para cada agrupamento. Esta informação é vital para a técnica de APD e encontrada na ferramenta PermutMatrix como visto na Figura 5.2.

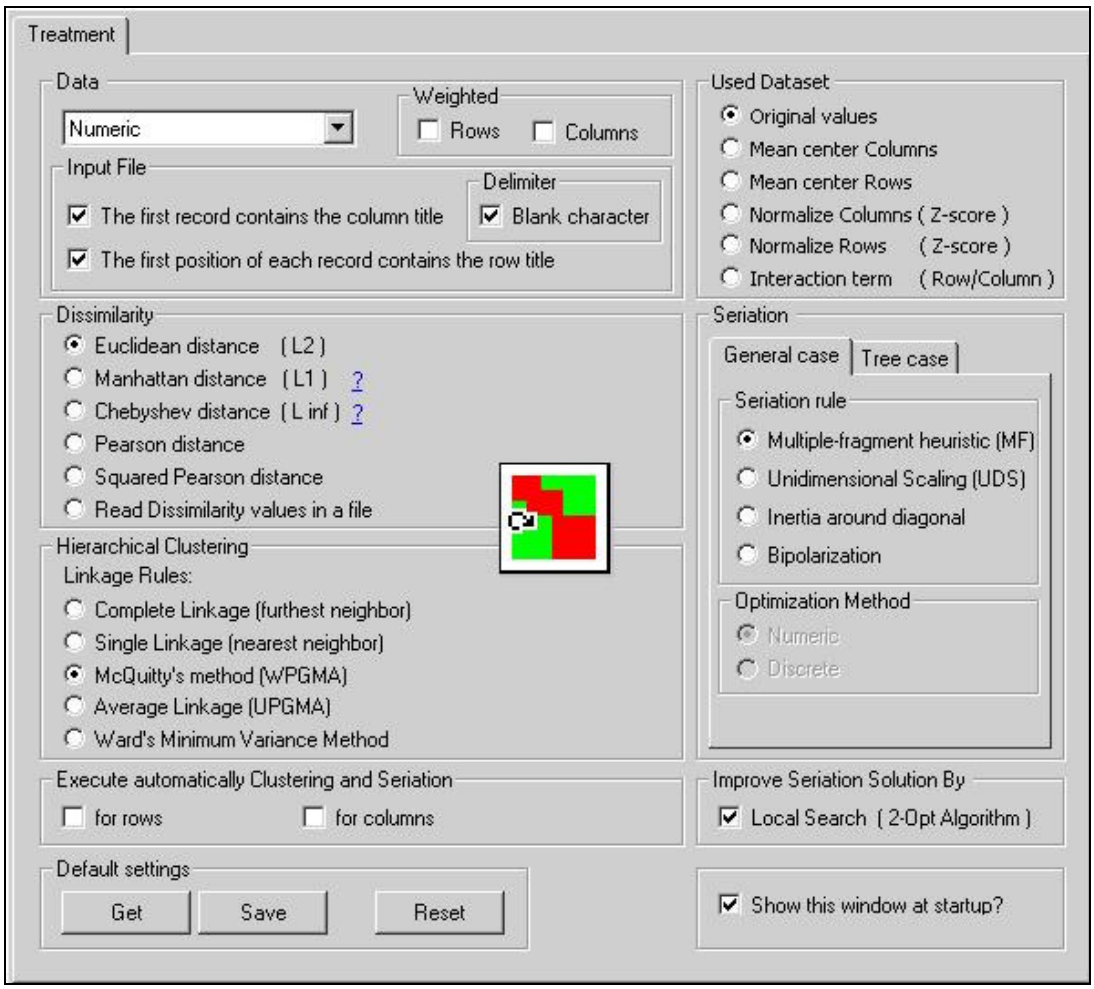


Figura 5.1: Tela de configurações e logo da ferramenta PermutMatrix.

Interface para Arquivo de Dissimilaridade: a ferramenta PermutMatrix, apesar de proporcionar o uso das medidas de dissimilaridade mais comuns encontradas na literatura (Distância Euclidiana, Distancia de Manhattan, Distância de Chebysheve, entre outras), ainda permite que um arquivo contendo uma matriz de dissimilaridade particular seja incorporado e usado pelos algoritmos implementados.

Por fim, a ferramenta PermutMatrix oferece um grau menor de configuração e variedade de técnicas e algoritmos quando comparada com a maior parte das ferramentas encontradas na literatura, mas este fator não prejudicou o estudo em questão, adaptando-se perfeitamente às necessidades específicas inerentes ao método.

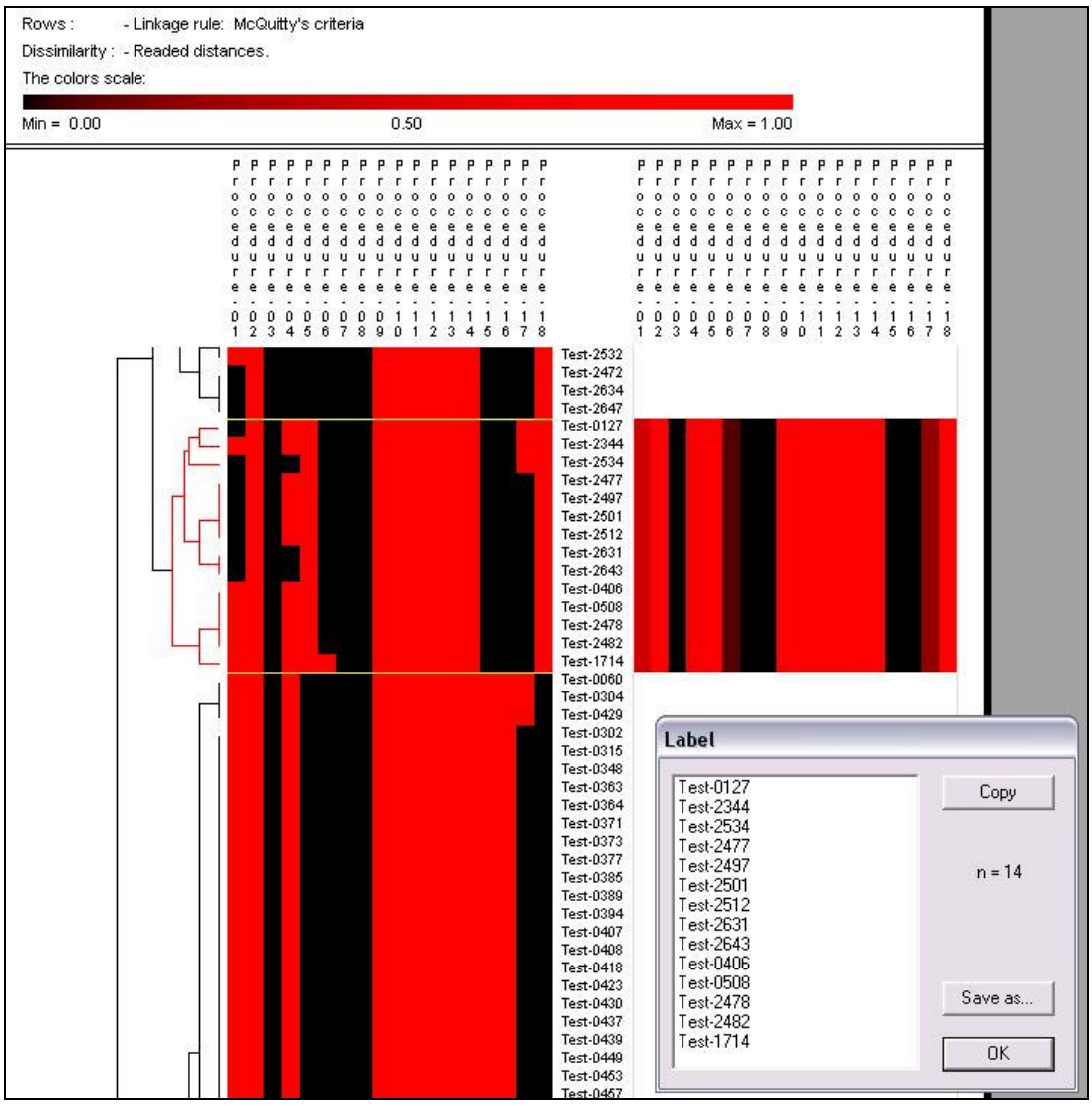


Figura 5.2: Tela de resultados da ferramenta PermutMatrix.

5.1.3. Pressupostos Iniciais

Antes que o método como um todo pudesse ser colocado em prática, três importantes decisões – afetadas por fatores externos – precisaram ser realizadas.

Quanto aos Dados de Entrada: após os Aplicativos Siemens (Tabela 5.1) terem sido cuidadosamente examinados, optou-se pela utilização do programa "*schedule*" por este ser o programa de maior número de versões que contém um arquivo de *trace* bastante interessante denominado "*thcovt*" (com a exceção de "*printtoken*", os demais arquivos da Lista Siemens não contêm este arquivo), que indica quais as funções e comandos que foram acionados pela execução de cada teste dos seus respectivos grupos de testes.

O interesse baseia-se no fato de que, como já citado anteriormente, o método desenvolvido precisa ser alimentado com informações que ressaltem a similaridade – ou não – dos testes que se relacionam com as mesmas áreas de código que executam, tornando as informações contidas neste trace de grande utilidade para o método, sem que informações adicionais precisassem ser geradas.

Também é necessário mencionar que um segundo arquivo do programa ainda é usado como fonte de informação, porém esse é comum à maioria dos programas da Lista Siemens. O arquivo "*fault-matrix*" contém dados que descrevem quais testes específicos foram capazes de encontrar defeitos em determinadas versões dos programas.

O pacote de informações disponíveis para o programa *schedule* – além dos arquivos já citados – compreende (entre outros arquivos que não foram utilizados) um Grupo de Testes (GT) de 2650 Casos de Testes (CTs), e também nove versões distintas do código, produzidas artificialmente.

Esta produção artificial de versões significa que o programa *schedule* original – escrito em C – teve seu código propositalmente alterado para que alguns dos TCs do GT as percebessem como defeitos. De uma versão para a próxima estes assim chamados defeitos artificiais foram corrigidos, porém, de um modo que mantivessem ainda alterações em relação ao programa original, que também são percebidos como defeitos. Esse processo foi repetido oito vezes, gerando nove diferentes versões do mesmo programa.

Quanto a Métrica de Dissimilaridade: uma vez esclarecido o conteúdo dos dados de entrada para a execução do método, também pode-se direcionar a escolha da medida de similaridade entre estes dados.

Tendo em mãos os dados sobre quais funções - e contidas nas funções quais comandos – são acionados por cada teste, percebeu-se que uma vez uma função acionada, os comandos executados dentro destas funções permanecem relativamente iguais, não trazendo nenhum grande ganho de informação ao método, portando, considerou-se que: "**Testes similares entre si são aqueles que acionam a maior quantidade possível de mesmas funções durante suas execuções**".

Imaginando que os testes então sejam descritos pelas funções que acionam e deixam de acionar durante suas execuções do programa, pode-se afirmar que cada teste pode ser representado por um vetor binário (cujo tamanho é especificado pelo número total de funções

deste programa), preenchido com “1” nas posições de funções que são acionadas e preenchido com “0” nas posições das funções que não são acionadas.

Percebe-se então que a similaridade entre dois vetores binários (ou testes) é ditada pelo número de posições de valor “1” (ou funções acionadas) que possuem em comum - basicamente o mesmo que define a Distância de Hamming descrita na Seção 3.3 e portanto selecionada como Métrica de Dissimilaridade do método desenvolvido.

Como exemplo, imagine-se que haja dois testes cujos descritores binários sejam Teste-1 [11011] e Teste-2 [01001]. A Distância de Hamming entre estes dois testes seria de 2, pois diferem na primeira posição e na quarta posição apenas ([**1**10**1**1] / [**0**10**0**1]), totalizando duas diferenças. Caso os descritores fossem idênticos – significando que possuem o mesmo comportamento ao passarem pelo código durante suas respectivas execuções – a Distância de Hamming entre os testes seria 0.

Quanto ao Algoritmo de Agrupamento: Já quanto ao Algoritmo de Agrupamento necessário ao método, optou-se por basear a decisão em experiências de trabalhos similares como em (YOO et. al., 2009) onde se obteve bons resultados associando a Distância de Hamming com técnicas de Algoritmos de Agrupamento Hierárquicos Aglomerativos (Detalhados na Seção 3.2 deste documento). Dentre as técnicas já descritas, decidiu-se então por usar o Agrupamento por Média Aritmética Ponderada (*WPGMA – Weighted Pair-Group Method using Arithmetic Average* ou Método de McQuitty) pela boa capacidade desta lidar com agrupamentos formados por grupos de tamanhos significativamente diferentes - que é o caso deste estudo – sem que estas magnitudes distintas influenciem o resultado da Tarefa de Agrupamento (Seção 3.2.4).

5.2. Utilização do Método

A utilização do método de Priorização de Casos de Testes de Regressão usando Amostragem por Perseguição de Defeitos está separada em três seções básicas denominadas Pré-processamento, Processamento e Pós-processamento. O intuito desta separação é de que o método em si – contido na Seção Processamento – possa ser uma solução genérica para qualquer tipo de programa ou sistema. Cabe então à Seção de Pré-processamento preparar os dados de um programa ou sistema específico, para que possam ser introduzidos diretamente na interface de entrada do método. Assim como cabe a Seção de Pós-processamento, receber

os dados da interface de saída do método para prepará-los para qualquer fim que lhes sejam designados.

Dessa forma, toda vez que o método tenha que se adaptar a um novo cenário, basta que as seções de Pré-processamento e Pós-processamento sejam adaptadas, mantendo-se a Seção Processamento em si, inalterada.

5.2.1. Pré-Processamento

O pré-processamento do método desenvolvido consiste em duas atividades de formatação e geração de dados que compreendem:

A Formatação do Arquivo de Trace "*thcovt*": esta atividade é puramente de formatação de dados. Os arquivos denominados "*thcovt*" encontram-se em um formato contendo muita informação desnecessária e de pouco uso prático (tendo em torno de centenas de milhares de linhas), portanto, foi desenvolvido um programa "*parse*" em C que percorre o arquivo de trace do início ao fim, identificando e extraindo as informações necessárias, gerando um novo arquivo matricial padrão usado pela maioria das ferramentas de análise de agrupamentos, que incluem a ferramenta *PermutMatrix*. A conversão dos arquivos está representada na Figura 5.3.

O arquivo gerado – no formato padrão esperado na entrada do processamento - contém uma matriz que lista os identificadores de todos os testes do grupo de teste na primeira coluna, lista os identificadores de todas as funções do programa na primeira linha e apresenta um valor binário indicando qual função de cada teste foi acionada ou não – função acionada com valor "1" e função não acionada com valor "0". Desta forma, as linhas do arquivo identificam as funções que cada teste acionou ou não, e vice-e-versa, as colunas indicam os testes que acionaram ou não a função identificada.

A Formatação do Arquivo de Matriz de Defeitos "*fault-matrix*": de modo análogo à conversão do arquivo "*thcovt*", um segundo programa "*parse*" foi desenvolvido em C para percorrer o arquivo "*fault-matrix*", identificar e extrair as informações necessárias, gerando um arquivo texto contendo a lista de testes do grupo de testes que identificaram um defeito em uma respectiva versão do código. A conversão do arquivo "*fault-matrix*" está representada na Figura 5.4.

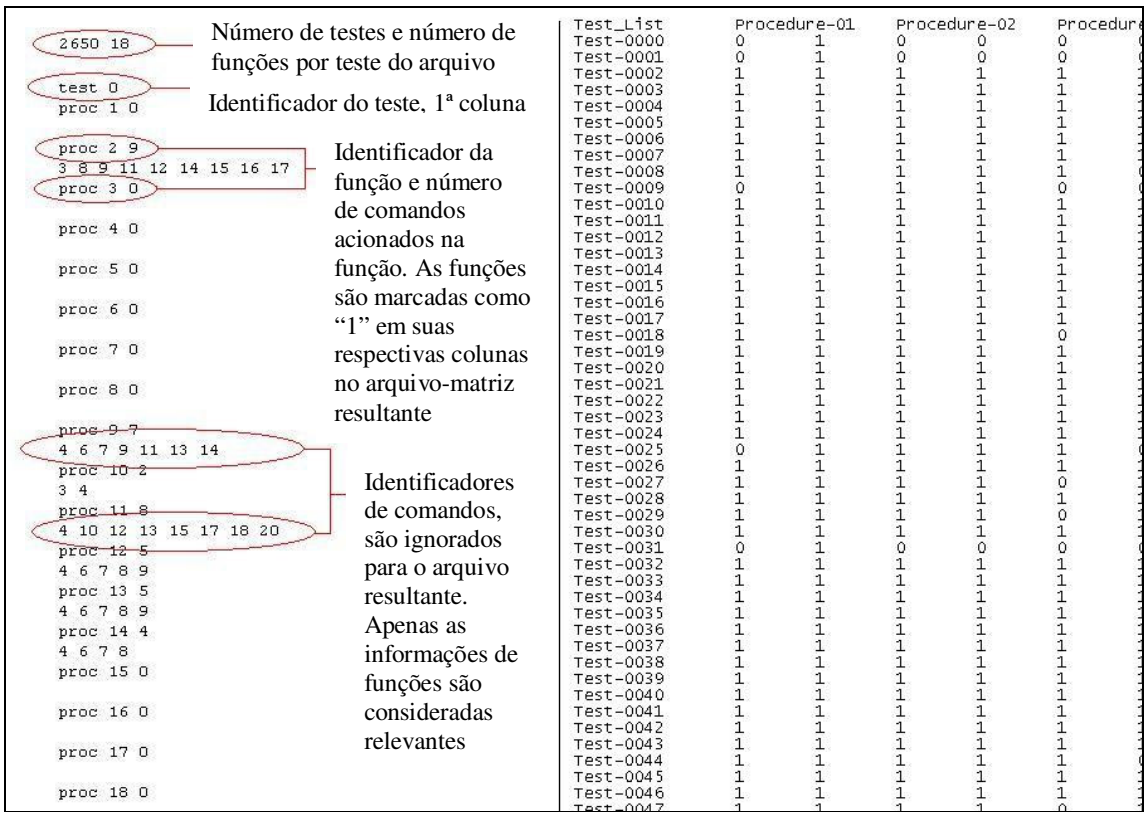


Figura 5.3: Representação da conversão do arquivo "*thcovt*"

O arquivo resultante – no formato esperado na entrada do processamento - contém três colunas que identificam respectivamente: o teste que evidenciou um defeito em uma determinada versão do programa, a versão do programa (o arquivo permanece ordenado por esta informação), e o valor binário "1" indicando a evidencia de um defeito (esta última coluna possui uma informação redundante, visto que as linhas com valores "0" foram expurgadas do arquivo para diminuir-lhe o tamanho).

Certamente, estas atividades de pré-processamento precisarão ser redefinidas para qualquer outro cenário que não apresente os mesmos formatos de dados do programa "*schedule*", selecionado para este estudo. O objetivo da redefinição seria também formatar os novos dados para as entradas esperadas para o processamento.

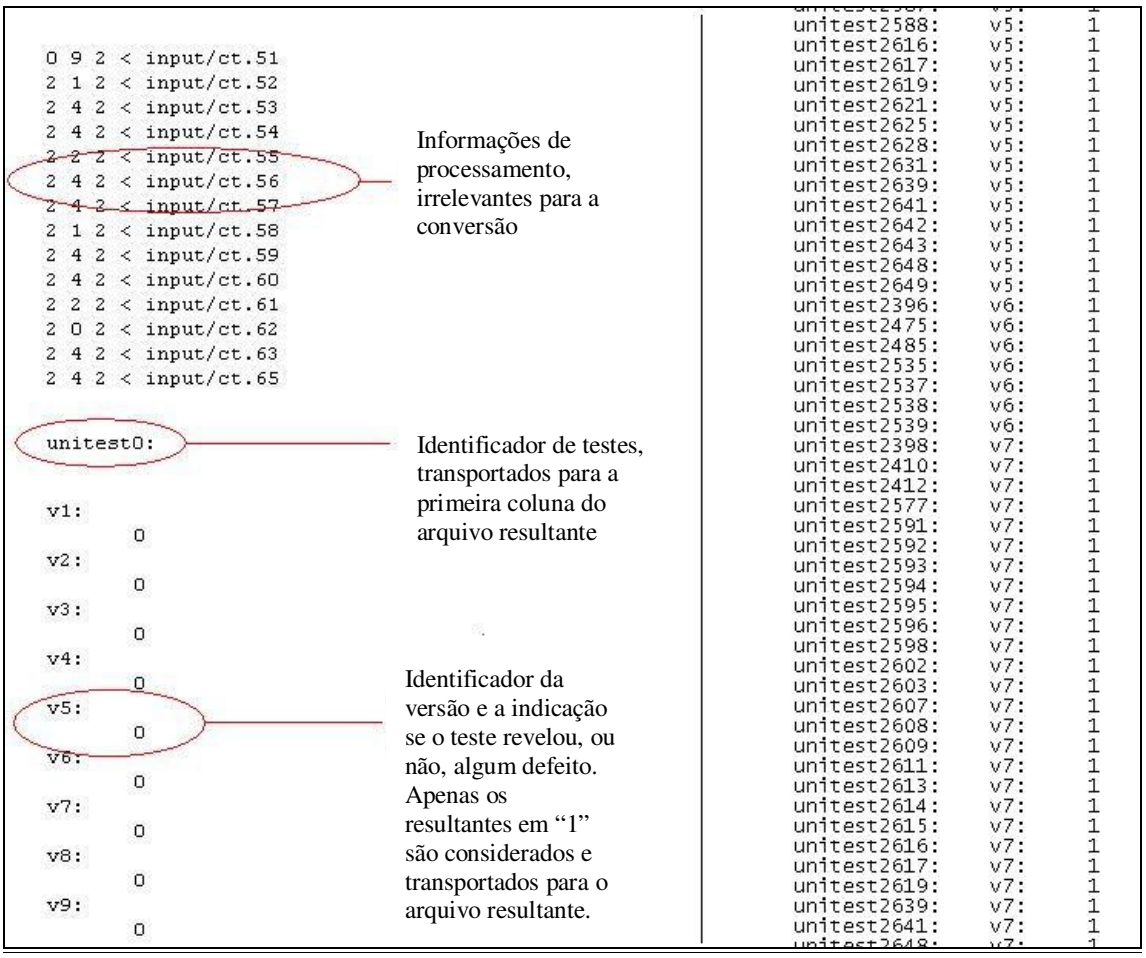


Figura 5.4: Representação da conversão do arquivo "*fault-matrix*"

5.2.2. Processamento

O processamento – que foi mantido genérico para qualquer caso que respeite o formato dos dados de entrada – é a seção onde todas as atividades inerentes ao método de Priorização de Casos de Teste de Regressão usando Amostragem por Perseguição de Defeitos, propriamente dito, são realizadas. Os passos estão descritos na ordem que devem ser executados.

A Geração da Matriz de Distâncias de Hamming: usando o arquivo de testes gerado pela conversão do arquivo "*thcovt*" durante o pré-processamento, foi criado um programa adicional, também desenvolvido em C, para gerar um arquivo contendo agora a matriz de Distâncias de Hamming como descrito na Seção 3.3. O programa gera um arquivo contendo uma matriz simétrica que relaciona cada teste do arquivo de testes, com todos os demais testes contidos neste arquivo de testes. O cruzamento dessas relações entre os testes contém o valor

da Distância de Hamming. A distância entre os testes é calculada ao comparar-se cada posição respectiva do descritor de dois testes e contabilizar-se quantas dessas posições contém valores diferentes, como exemplificado na Figura 5.5.

Por exemplo, observando-se as sequências binárias parcialmente apresentadas na Figura 5.6 (Teste-0000 [01000], Teste-0001 [01000], Teste-0002 [11111]), pode-se determinar a Distância de Hamming entre as sequências da seguinte forma: A Distância de Hamming entre as sequências Teste-0000 e Teste-0001 é igual a 0, pois as mesmas são idênticas, possuindo todos os valores iguais. Já a Distância de Hamming entre as sequências Teste-0001 e Teste-0002 é igual a 4, pois diferem na primeira, na terceira, na quarta e na quinta posição (**01000 / 11111**).

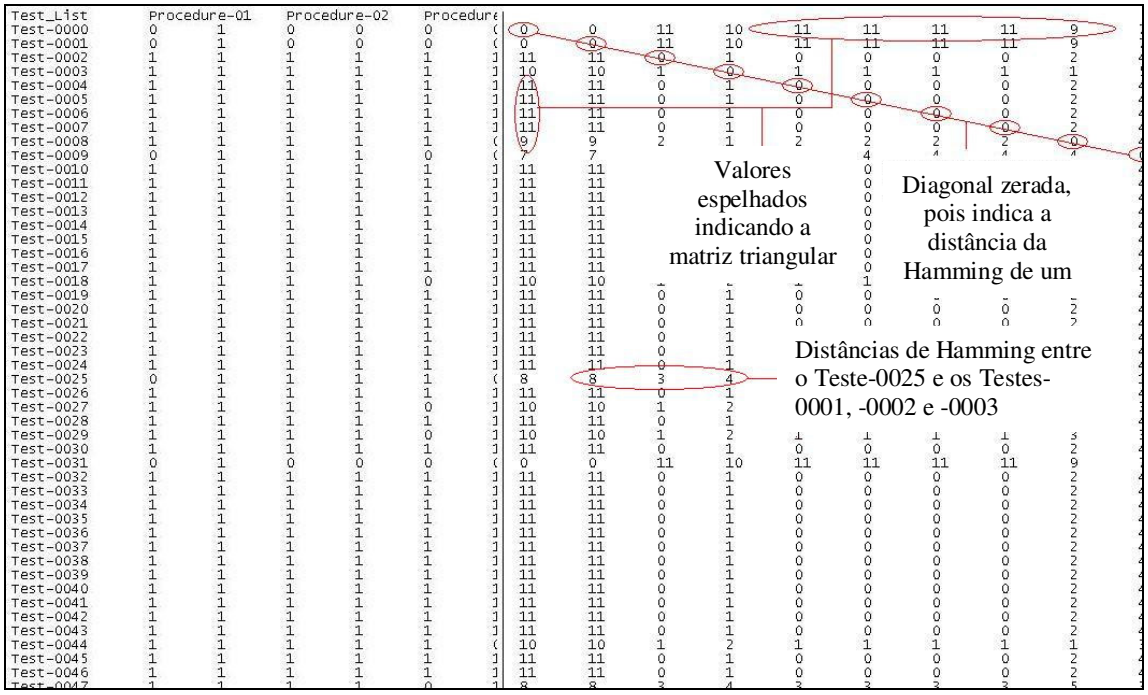


Figura 5.5: Representação da geração do arquivo de Distâncias de Hamming

A Geração dos Agrupamentos: de posse de ambos os arquivos de entrada para o PermutMatrix é possível gerar os agrupamentos do método desenvolvido. Os passos de configuração realizados na ferramenta citada estão enumerados abaixo e também apresentados na Figura 5.6:

1. Na caixa de opções "*Data*" se seleciona a opção "*Binary*" indicando que o arquivo de entrada é formado de dados binários;

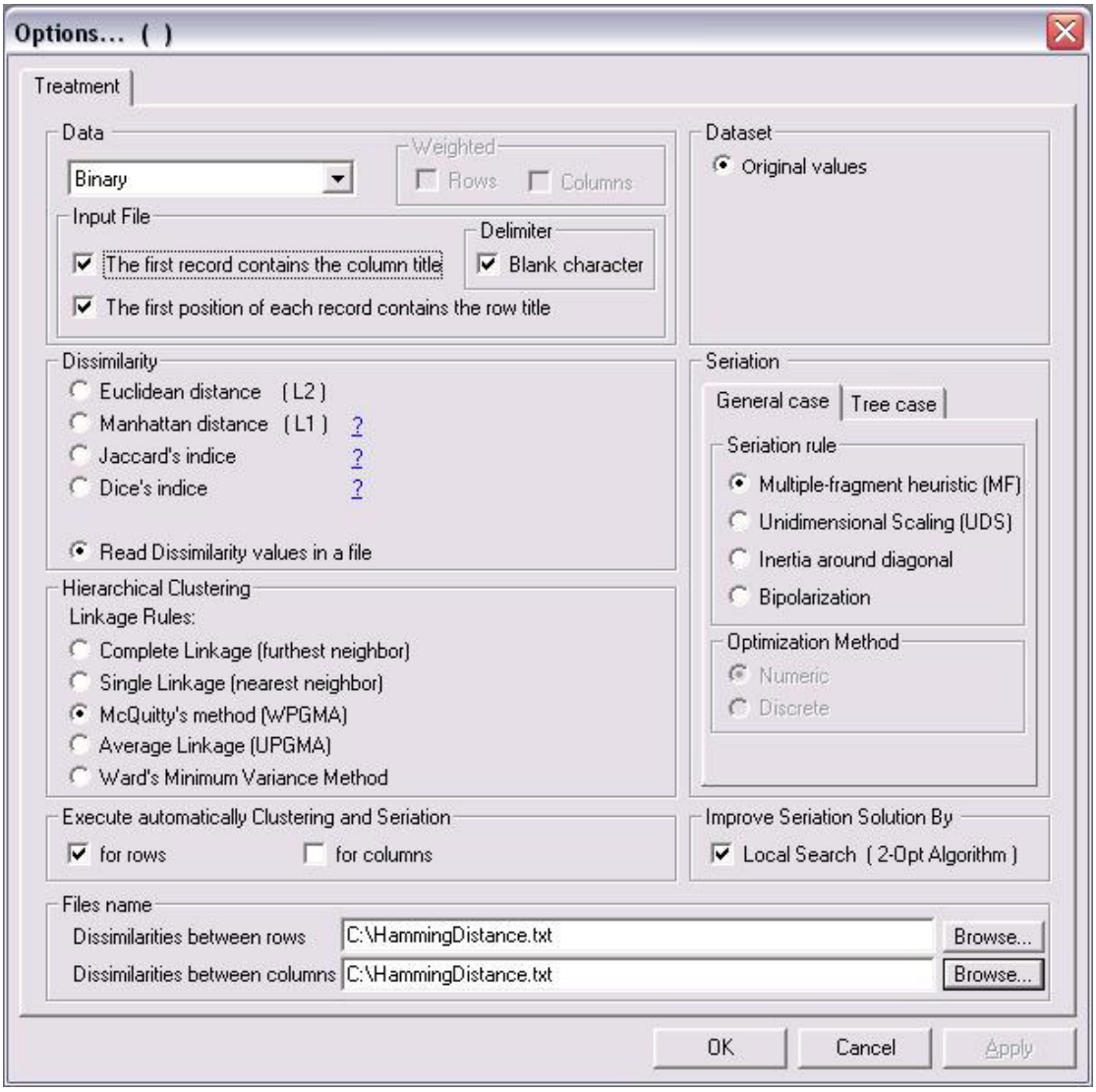


Figura 5.6: Opções de geração agrupamentos na ferramenta PermutMatrix.

2. Na caixa de opções "*Input File*" mantêm-se todas as opções marcadas ("*The first record contains the column title*", "*The first position of each record contains the row title*", "*Delimiter – Blank character*"), indicando os detalhes da configuração do arquivo de entrada;
3. Na caixa "*Dissimilarity*", seleciona-se a opção "*Read Dissimilarity values in a file*", indicando à ferramenta que esta deve buscar os dados de dissimilaridade de um arquivo externo;

4. Na caixa "*Hierarchical Clustering*" se seleciona a opção "*McQuitty's method (WPGMA)*", indicando que a ferramenta deve usar o algoritmo de agrupamento por Média Aritmética Ponderada;
5. Na caixa "*Execute automatically Clustering and Seriation*" marca-se apenas a opção "*for rows*", indicando que apenas as linhas do arquivo de entrada devem ser agrupadas (que significam os testes, e não as colunas, que indicariam as funções do programa);
6. Na caixa "*Files name – Dissimilarities between rows*" insere-se o caminho para o arquivo de Distâncias de Hamming que foi gerado no passo anterior;
7. Pressiona-se o botão "*OK*", sendo as demais opções irrelevantes para o estudo.

O resultado da ferramenta PermutMatrix é apresentado em três janelas distintas (1 – Dados sequencialmente carregados, 2 – Dados agrupados e 3 – Dados agrupados e otimizados para visualização). Para evitar uma variável desnecessária, a terceira janela - Dados agrupados e otimizados para visualização - foi desconsiderada, pois devido à grande magnitude dos dados esta não fornece nenhum auxilio para a visualização dos dados. A segunda janela, portanto – Dados Agrupados - foi considerada como resultado da tarefa de agrupamento. A janela resultado, mostrada em detalhes na Figura 5.7, contém o resultado dos agrupamentos realizados.

Explicando-se o que a janela de resultado apresenta, à esquerda da Figura 5.7 pode se ver o dendograma resultante. A extrema esquerda ficam representados os "nós" mais próximos à "raiz" (o agrupamento contendo todos os objetos) e à direita da "árvore", os "nós folhas" (indicando os agrupamentos unitários contendo cada um dos testes do GT).

Os CTs unitários são descritos pelas "linhas" bicolores (que a primeira vista parecem um só bloco bicolor) identificadas à sua direita pelo seu respectivo identificador. As linhas apresentam duas cores, pois indicam em cor mais clara as funções que os testes ativaram durante sua execução e em preto as funções que não ativaram. Encabeçando o "bloco bicolor" ficam representados os identificadores das funções.

A área à direita da janela de resultado – que possui uma representação quase idêntica à área da esquerda – tem a função de indicar as mesmas informações, porém, somente para uma área que foi selecionada, limitando um agrupamento específico "cortado" a uma determinada altura da "arvore".

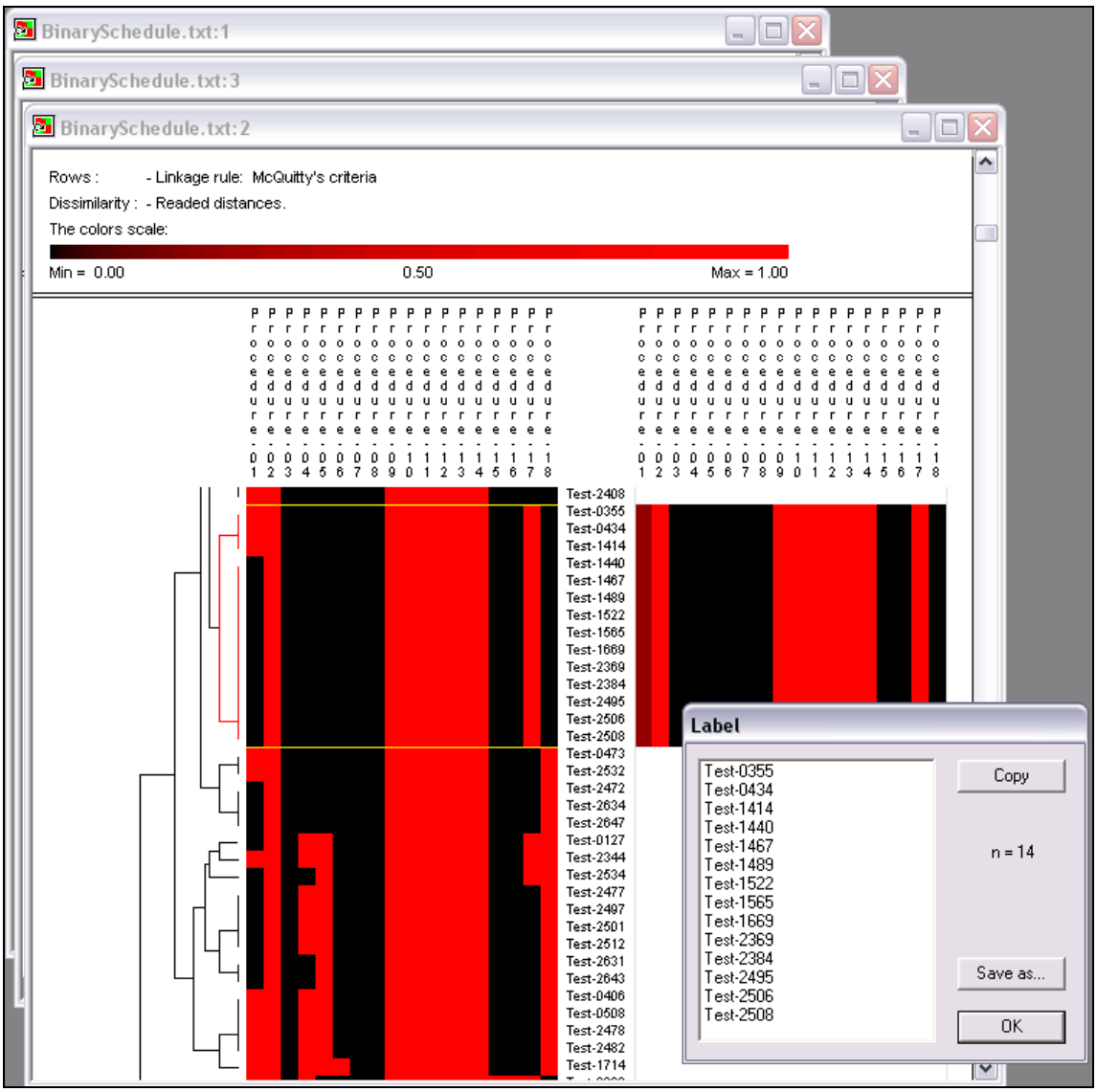


Figura 5.7: Janela de resultados de agrupamentos da ferramenta PermutMatrix

Ponto de Poda do Dendograma: possivelmente o passo mais crítico do processamento do método, o ponto de poda do dendograma tem grande influência no resultado do método desenvolvido. A capacidade do método de Priorização de Casos de Testes de Regressão usando Amostragem por Perseguição de Defeitos de direcionar a priorização dos CTs de modo efetivo depende de um equilíbrio delicado entre o "Nó Raiz" (contendo todos os CTs do agrupamento) e os "Nós Folhas" (contendo apenas os CTs que possuem um comportamento idêntico quando executados sobre o código) – decisão representada na Figura 5.8.

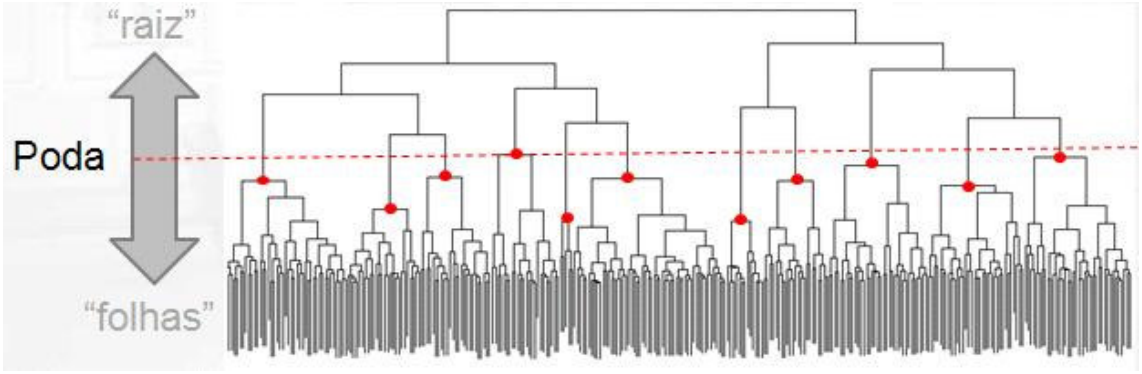


Figura 5.8: Decisão de poda do dendograma

O conceito por trás do método baseia-se em: "Aglomerar os agrupamentos um a um – usando a média aritmética ponderada das distâncias Hamming entre os testes como critério de similaridade – até que todos os testes que revelaram defeitos na versão anterior do programa, estejam agrupados com os testes que revelam defeitos na versão presente", ou seja, caso o dendograma resultante seja podado muito próximo do topo – em um dos últimos passos da execução do algoritmo – existe uma grande probabilidade de se ter agrupado CTs em excesso, desnecessários, que serão priorizados erroneamente e influenciarão a eficiência do método de forma negativa. Por outro lado, caso o dendograma resultante seja podado muito próximo da base – em um dos primeiros passos da execução do algoritmo - é possível que se perca a relação "intra-versões" dos CTs, por estes conterem diferenças eventualmente mais acentuadas de padrão impossibilitando o método isolar em agrupamentos selecionados, os CTs de interesse.

Geração da Lista de Casos de Testes Priorizados: assim que o número de agrupamentos resultantes é fixado – a poda do dendograma tenha sido realizada - pode-se iniciar a priorização dos CTs para formar uma nova lista de prioridades obedecendo às regras de amostragem proposta pelo método de Priorização de Casos de Testes de Regressão usando Amostragem por Perseguição de Defeitos. Os passos realizados estão descritos abaixo e sumarizados nos fluxogramas das Figuras 5.9 e 5.10:

1. O início da priorização depende da disponibilidade dos agrupamentos formados, coletados da saída da ferramenta PermutMatrix; dos CTs que identificaram defeitos na versão anterior, encontrados no arquivo resultante da conversão do arquivo "*fault-matrix*"; e de um valor arbitrado de "*k* vizinhos" de CTs a serem selecionados;

1a. De posse de um CT aleatório da lista de CTs da versão anterior, identifica-se o agrupamento em que este CT encontra-se e então retira-se, aleatoriamente, "*k"* CTs deste agrupamento para a Lista Priorizada (o CT que identificou o agrupamento não é adicionado à Lista Priorizada);

1b. Examina-se o arquivo resultante da conversão do arquivo "*fault-matrix*", caso os CTs recém retirados no Passo (1a) tenham revelado algum defeito na versão atual, retorna-se ao passo (1a) para retirar mais "*k*" CTs, aleatoriamente, do mesmo agrupamento. Esse processo é repetido até que um grupo de CTs tenha sido retirado e nenhum destes tenha sido responsável por revelar algum defeito no arquivo examinado;

1c. Retorna-se então ao passo (1a) para repetir o processo para um novo CT da lista de CTs da versão anterior. Este processo é repetido até que todos os agrupamentos identificados pelos CTs da versão anterior tenham sido esgotados, ou seja, todos os CTs destes agrupamentos, identificados por CTs da versão anterior, tenham sido retirados para a Lista Priorizada;

2. Retira-se então um CT aleatório de cada agrupamento restante para a Lista Priorizada;
3. Examina-se o arquivo resultante da conversão do arquivo "*fault-matrix*" para verificar se os CTs recém retirados no Passo (2) tenham revelado algum defeito na versão atual;
4. Caso algum CT tenha sido encontrado no Passo (3), retira-se aleatoriamente "*k*" CTs de cada um destes agrupamentos em questão para a Lista Priorizada. Retorna-se ao Passo (3) até que nenhum CT selecionado revele defeitos;
5. Retorna-se ao Passo (2) até que todos os CTs tenham sido retirados dos agrupamentos para a Lista Priorizada.

O resultado final do método desenvolvido consiste em uma lista de todos os CTs disponíveis ordenados pela prioridade formada pela técnica de amostragem supracitada, o que significa que: nas primeiras posições da lista estão localizados os testes que tem a maior probabilidade de encontrarem comportamentos indesejados que, eventualmente, possam ter sido inseridos na área de código modificada – pois executam essa mesma área de código e suas adjacências imediatas. Os testes mais abaixo na lista de prioridades caracterizam-se por possuírem um envolvimento gradativo cada vez menor com a área de código modificada, sendo que os últimos testes da lista, possivelmente, não possuem relação nenhuma com a mesma.

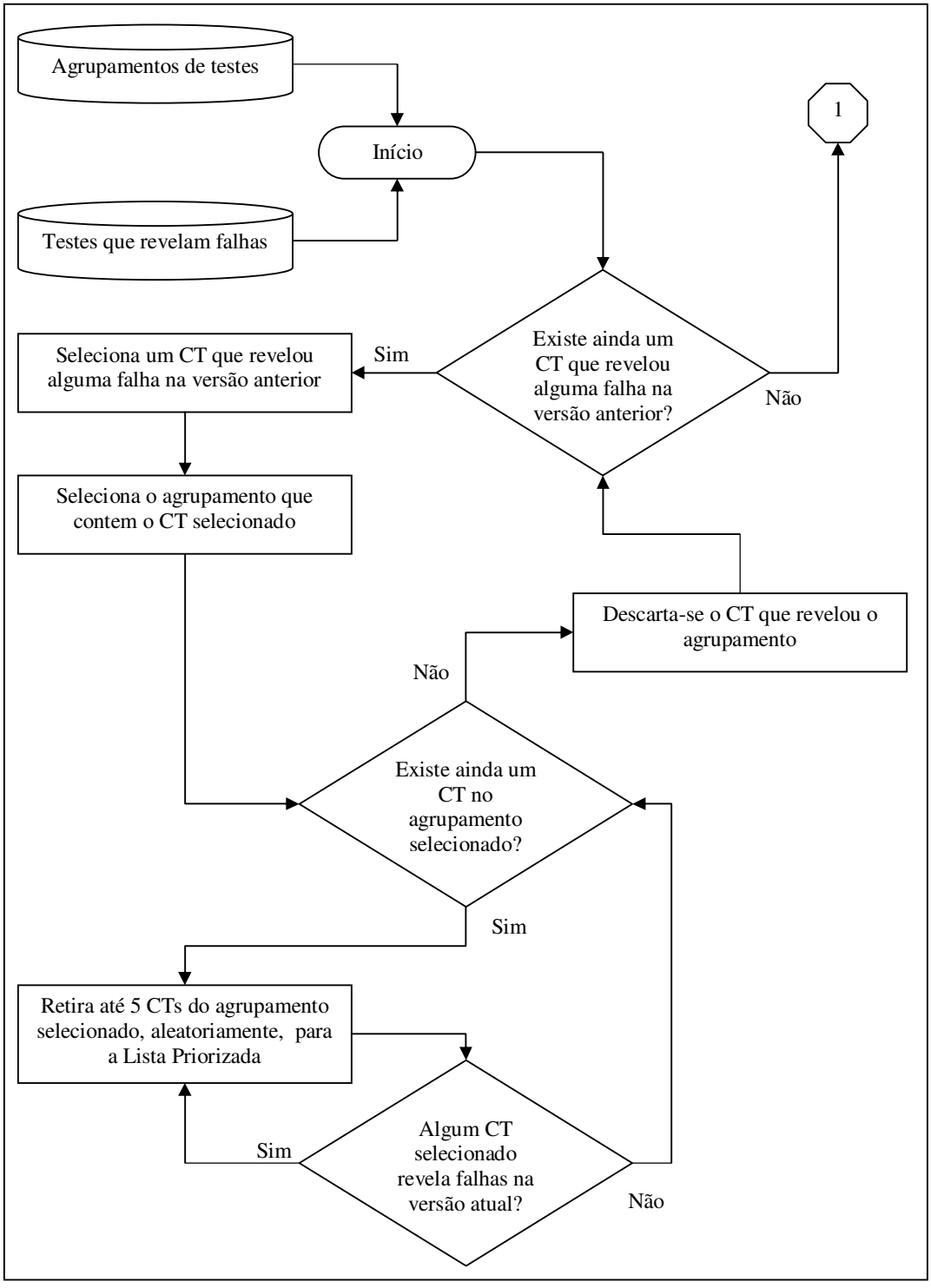


Figura 5.9: Fluxograma da Adaptação da Fase de Amostragem (Parte 1)

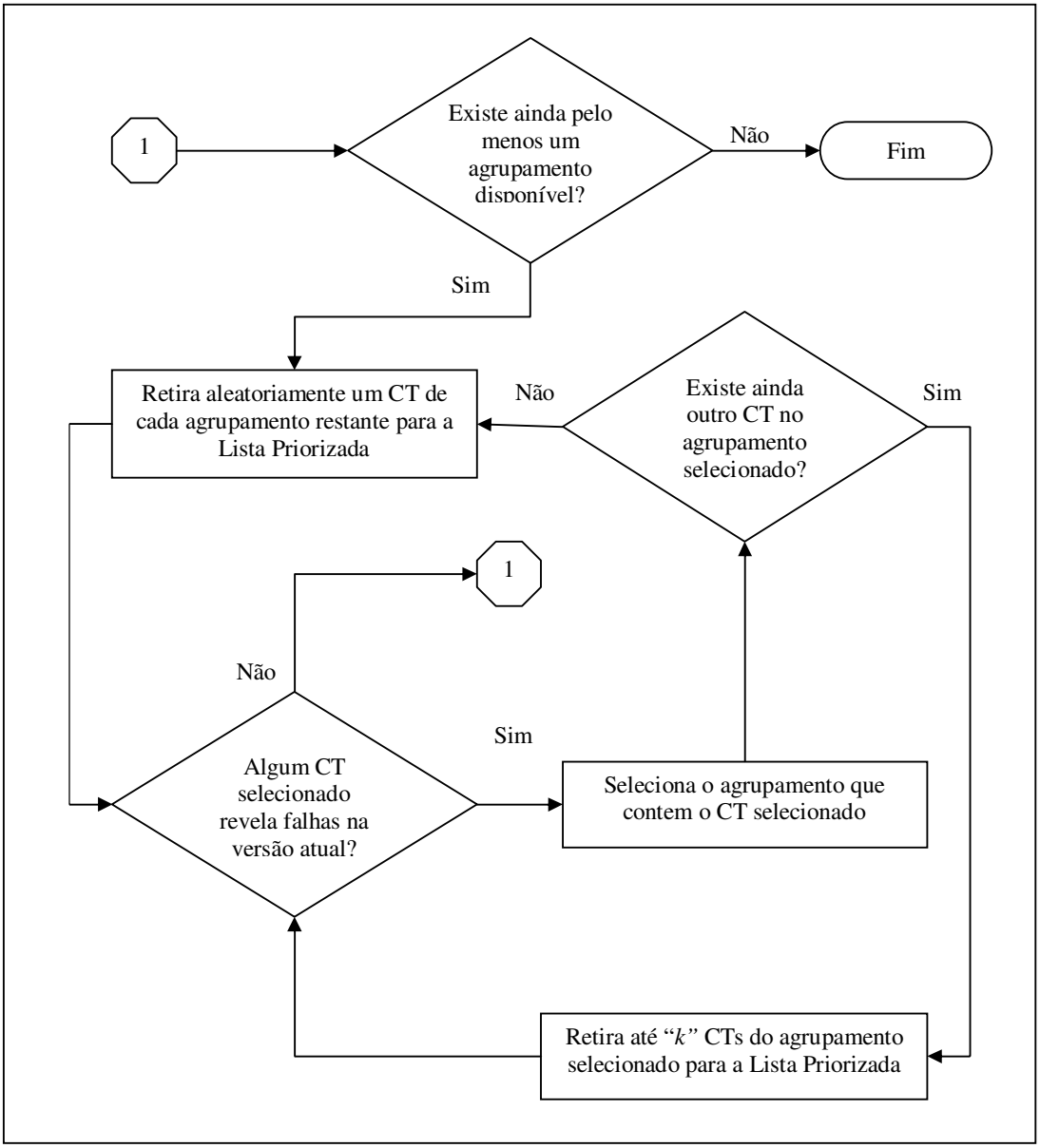


Figura 5.10: Fluxograma da Adaptação da Fase de Amostragem (Parte 2)

5.2.3. Pós-Processamento

Com o fim da seção de Processamento tem-se então a lista priorizada de CTs, que é o resultado que o método desenvolvido almeja, porém, o formato deste resultado muitas vezes ainda precisa passar por uma fase posterior de processamento para adequá-lo ao propósito específico ao qual se destina. Por exemplo, um formato que permita transportar a ordem dos testes a serem executados para uma ferramenta de suporte a testes, ou até mesmo uma ferramenta de testes automáticos, etc..

No caso deste documento - como se pretende demonstrar a eficiência do método desenvolvido- o Pós-processamento basicamente consistiu em importar a lista priorizada de CTs resultante da aplicação do método desenvolvido e o resultado da conversão do arquivo "*fault-matrix*" para a ferramenta Excel. Esta ferramenta foi usada para produzir os gráficos de Média da Percentagem de Detecção de Defeito (MPDD) - descritos na Seção 2.2.3 – que, por sua vez, foram usados para demonstrar a eficiência do método de Priorização de Casos de Testes de Regressão usando Amostragem por Perseguição de Defeitos.

O gráfico MPDD representa de forma bidimensional a relação temporal entre a execução dos testes do GT – eixo horizontal – e os testes do GT que são capazes de revelar defeitos na determinada versão de código – eixo vertical. A área formada representa a eficiência da ordem de execução dos testes, quanto maior a área maior é a relevância dos testes executados com prioridade.

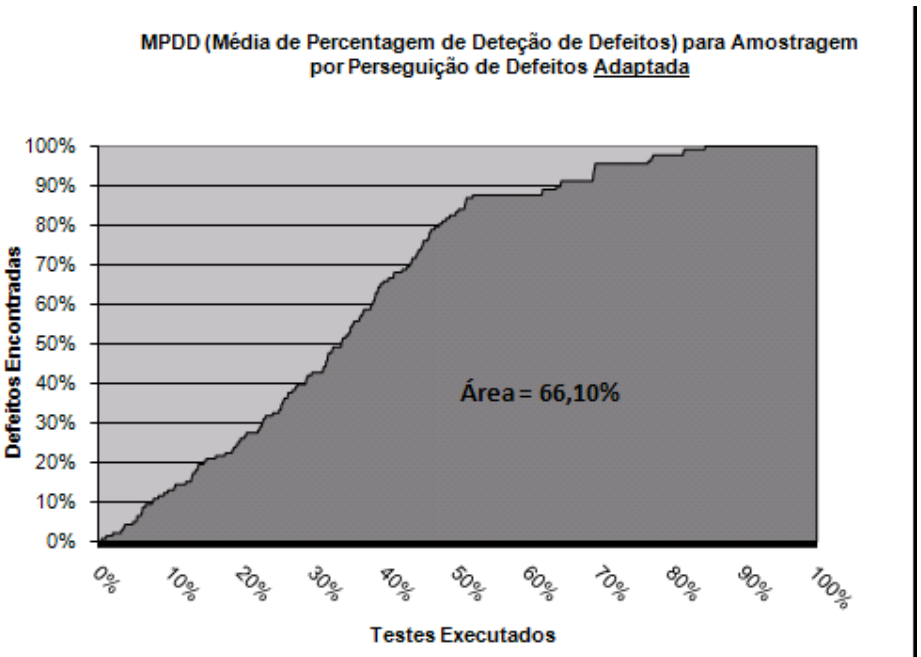
Eixo Horizontal (Testes Executados): Para formar o eixo horizontal do gráfico MPDD apenas se importou a lista priorizada de CTs resultante da aplicação do método desenvolvido (coluna Testes Ordenados da Figura 5.11) e usou-se a quantidade de testes (coluna Testes Executados da Figura 5.11) para representar as quantidades parciais em forma de percentagem do todo (coluna Percentagem de Testes da Figura 5.11 - sendo a lista completa considerada como 100%).

Eixo Vertical (Testes Reveladores): Já o eixo vertical do gráfico MPDD é formado extraindo-se do arquivo resultante da conversão do arquivo "*fault-matrix*", as informações sobre quais são os testes específicos que revelam algum defeito (coluna Defeitos da Figura 5.11), sumarizando-as (coluna Acúmulo de Defeitos da Figura 5.11) e representando-as em forma de percentagem (coluna Percentagem de Defeitos da Figura 5.11). Com todas as colunas preenchidas, usa-se então a coluna Integração da Área da Figura 5.11 para o cálculo da área do gráfico.

5.3. Apresentação de Resultados

O método desenvolvido – Priorização de Casos de Testes de Regressão usando Amostragem por Perseguição de Defeitos – foi executado repetidamente sobre os dados disponíveis para as várias versões do programa "*schedule*" - descrito na Seção 5.1.3 - e os resultados são apresentados abaixo através de uma sequência de gráficos de Média da Percentagem de Detecção de Defeitos (MPDD) - descritos na Seção 2.2.3.

Para cada geração do gráfico MPDD usando o método desenvolvido, também foram realizadas – para efeitos de comparações de eficiência - duas outras produções de gráficos MPDD similares, onde a única diferença residiu nas ordens da lista de CTs utilizadas, produzindo para cada versão de programa três gráficos MPDD:



	2650	66,10%		138		241740
Percentagem de Testes	Testes Executados	Testes Ordenados	Defeitos	Acumulo de Defeitos	Percentagem de Defeitos	Integração da Área
0%	1	Test-2464		0	0%	0
0%	2	Test-2146		0	0%	0
0%	3	Test-2060		0	0%	0

Figura 5.11: Representação da Geração dos Gráficos MPDD

1. Lista de CTs sem alteração, usando a ordem original dos CTs,
2. Lista de CTs ordenada por Amostragem por Perseguição de Defeitos original,
3. Lista de CTs ordenada por Amostragem por Perseguição de Defeitos adaptada.

Como o método desenvolvido tem por objetivo priorizar testes de regressão (TR), precisando portanto ser alimentado com informações de uma versão anterior do programa, não é possível aplicar o método sobre a primeira versão de cada programa.

A necessidade de se produzir uma versão adicional de um programa está diretamente ligada a uma alteração de código. Seja para corrigir um defeito, seja para inserir ou alterar uma funcionalidade, uma nova versão de código apresenta alterações no mesmo. O método desenvolvido precisa ser "informado" das alterações de código que ocorreram da versão anterior do programa para a atual – informações que balizam suas decisões - e as consegue através das descrições dos Casos de Testes (CTs) específicos que revelaram defeitos na versão anterior do programa (exatamente os defeitos que motivaram as alterações realizadas).

A título de informação (não é escopo deste estudo abordar o custo da otimização) os programas "C" desenvolvidos para:

1. Converter os arquivos textos "*thcovt*" (1.153 kb) e "*fault-matrix*" (392 kb) nos arquivos textos "ScheduleBinario" (125 kb) e "ScheduleMatrizDefeitos" (16 kb) levou aproximadamente 6 segundos.
2. Gerar a matriz das distâncias de Hamming - com base no arquivo "ScheduleBinario" - levou aproximadamente 4 minutos e 30 segundos e gerou um arquivo texto chamado "ScheduleDistanciasHamming" (14.401 kb).

A execução da tarefa de agrupamento na ferramenta PermutMatrix – usando os arquivos "ScheduleBinario" e "ScheduleDistanciasHamming" – levou aproximadamente 20 segundos.

5.3.1. Resultados do Programa *Schedule* usando Poda 46 e 5 Vizinhos

Os parâmetros definidos para o método desenvolvido para esta parte do estudo foram selecionados como "Poda 46", significando que o dendograma resultante do Agrupamento Hierárquico Aglomerativo por Média Aritmética Ponderada, foi podado quando o número de agrupamentos (inicialmente, igual ao número de testes 2650) chegou a 46 agrupamentos representados pela Figura 5.12. Esta definição foi tomada analisando-se visualmente o dendograma resultante da fase de agrupamento, e traçando-se uma "linha" a dois níveis de distância dos "nós folha". E "Vizinhos 5", indicando que o valor 5 foi usado para o parâmetro "*k*" vizinhos do método desenvolvido - selecionado de maneira análoga ao valor comumente encontrado na literatura base (LEON e PODGURSKI, 2003), (YOO et. al., 2009).

Como já citado anteriormente, as informações disponíveis para o programa *schedule* compreendem nove versões distintas do mesmo. Os resultados discutidos abaixo foram produzidos para versões 3 a 8 (v3, v4, v5, v6, v7 e v8), e não foram produzidos para as

versões 1, 2 e 9, como representado na Tabela 5.2 – que também apresenta os valores de áreas observados para cada versão, que por sua vez, são discutidos individualmente na continuidade desta seção.

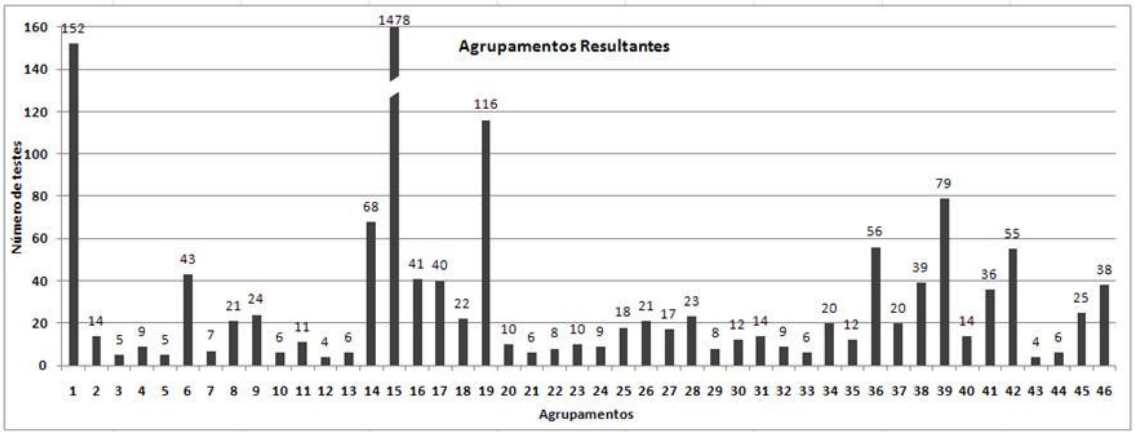


Figura 5.12: Agrupamentos do programa *schedule* usando nível de Poda 46

A primeira versão do programa não pode ser usada pelo método desenvolvido (a não ser como fonte de informação) por motivos já citados, não havendo uma versão anterior disponível. E a segunda e a nona são consideradas "*outliers*" e, portanto, discutidas na Seção 5.4 – Dificuldades Encontradas.

Versões do Programa Schedule	Número Total de Testes que Revelam Defeitos	Ordem de Testes Original	Ordenação por Amostragem por Perseguição de Defeitos Original	Ordenação por Amostragem por Perseguição de Defeitos Adaptada	Observação
Versão 2	*"Outlier"*				
Versão 3	138 de 2650	54,65%	39,50%	66,10%	Boa otimização
Versão 4	235 de 2650	49,65%	29,72%	69,20%	Boa otimização
Versão 5	40 de 2650	5,07%	44,41%	66,62%	Boa otimização
Versão 6	5 de 2650	5,32%	49,10%	27,92%	Sem otimização
Versão 7	27 de 2650	2,35%	45,09%	45,91%	Sem otimização
Versão 8	30 de 2650	53,80%	27,82%	72,80%	Boa otimização
Versão 9	*"Outlier"*				

Tabela 5.2: Sumário de Resultados Obtidos para Poda 46 e Vizinhos 5

Gráficos MPDD para a Versão 3 do Programa Schedule: Para a versão 3 do programa *schedule* (utilizando as informações sobre os testes que revelaram defeitos na versão 2), tem-se um caso de melhora significativa provocada pelo método desenvolvido. Uma melhora de

aproximadamente 11,5 pontos percentuais em relação à sequência original e aproximadamente 22 pontos percentuais em relação ao método original.

O ganho de eficiência ainda não é completo, pois pode se observar que nos gráficos da Figura 5.13 e Figura 5.14 tem-se uma escalada agressiva na execução dos CTs que revelam defeitos na versão 3 – dos 138 CTs que revelam defeitos, 120 são executados até 51% do total dos testes (1358 dos 2650) – porém, além dessa percentagem a taxa de execução de CTs que revelam defeitos diminui bastante. O método desenvolvido ainda é capaz de executar todos os CTs que revelam defeitos (os 138) quando a execução dos testes alcança a marca de 84% do total (2238 CTs).

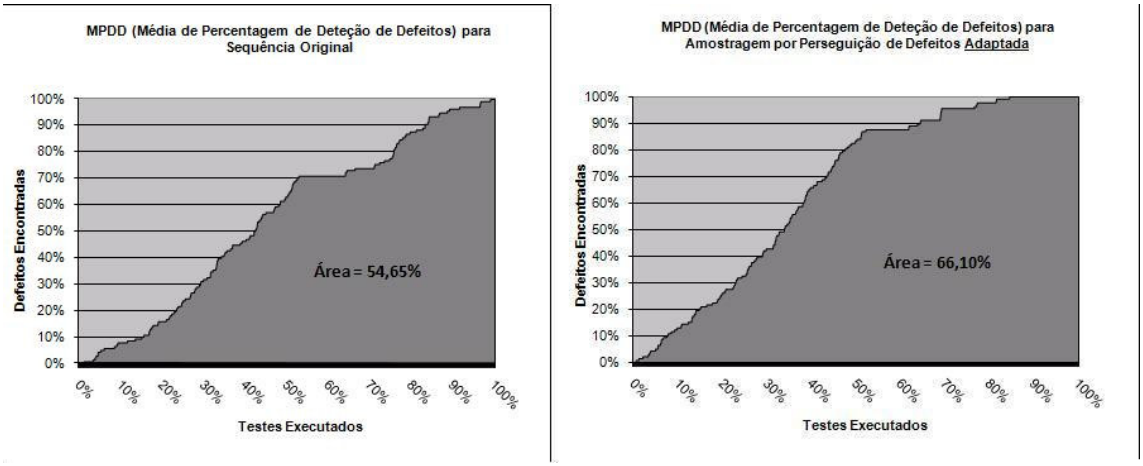


Figura 5.13: Gráficos MPDD da Versão 3 do Programa *Schedule* (Parte 1)

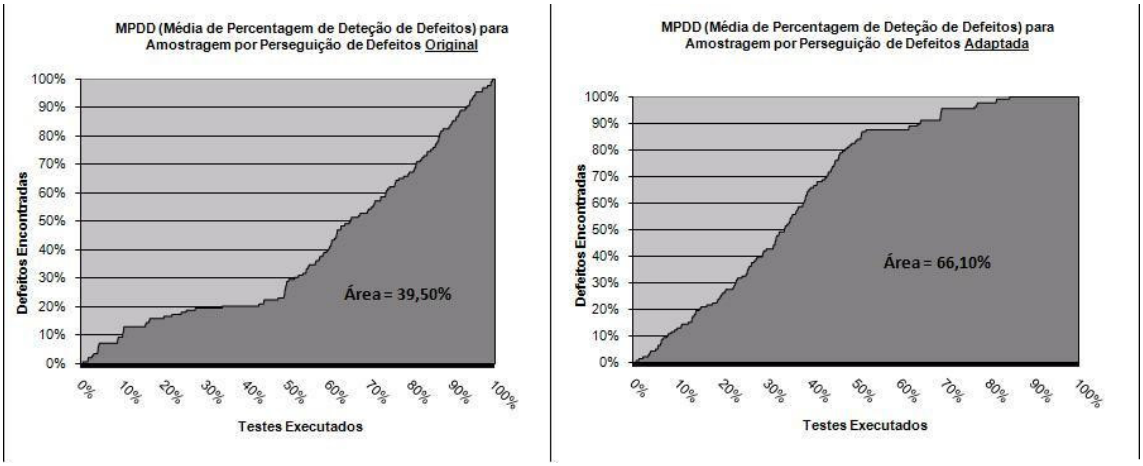


Figura 5.14: Gráficos MPDD da Versão 3 do Programa *Schedule* (Parte 2)

Essa mudança drástica na taxa de execução de CTs que revelam defeitos, se da pelo fato de que parte da informação incorporada de versão anterior (versão 2) foi corretamente interpretada – que possibilitou a correta priorização dos primeiros 120 CTs que revelam defeitos – enquanto parte da informação não foi identificada corretamente – deixando18 CTs que revelam defeitos em prioridade normal.

Para se conseguir uma eficiência completa do método desenvolvido é necessário ajustar o momento de poda do dendograma formado pelo Algoritmo Hierárquico Aglomerativo para que o conjunto de CTs que revelaram defeitos na versão 2, relacionados a estes 18 CTs não priorizados que revelam defeitos na versão 3, sejam colocados em um mesmo agrupamento.

Gráficos MPDD para a Versão 4 do Programa Schedule: Para a versão 4 do programa *schedule* (usando as informações sobre os testes que revelaram defeitos na versão 3), representa um caso de eficiência completa de otimização produzida pelo método desenvolvido (e o resultado almejado). Enquanto a ordem original e o método original (Figura 5.15 e Figura 5.16 respectivamente) apresentam uma progressão mais tímida da utilização dos CTs que revelam defeitos na versão 4 (um total de 235 CTs), o método desenvolvido utiliza informações sobre a área de código modificada desde a versão anterior (versão 3), e prioriza os testes que são aplicados nesta mesma região de código alterada.

Pode-se observar uma melhora de aproximadamente 20 pontos percentuais em relação à sequência original e aproximadamente 40 pontos percentuais em relação ao método original.

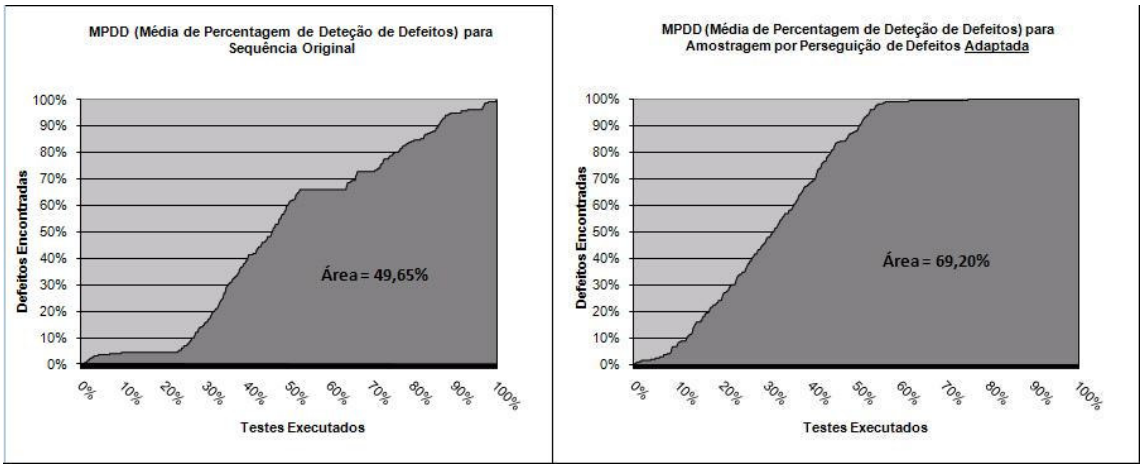


Figura 5.15: Gráficos MPDD da Versão 4 do Programa *Schedule* (Parte 1)

O resultado é de que os 235 CTs que revelam defeitos na versão 4 são em sua maioria executados (207 CTs que revelam defeitos) antes que o progresso da totalidade dos testes chegue a 50% (1325 CTs), e todos os 235 CTs que revelam defeitos são executados quando a totalidade dos testes alcança 75% do todo (1975 CTs).

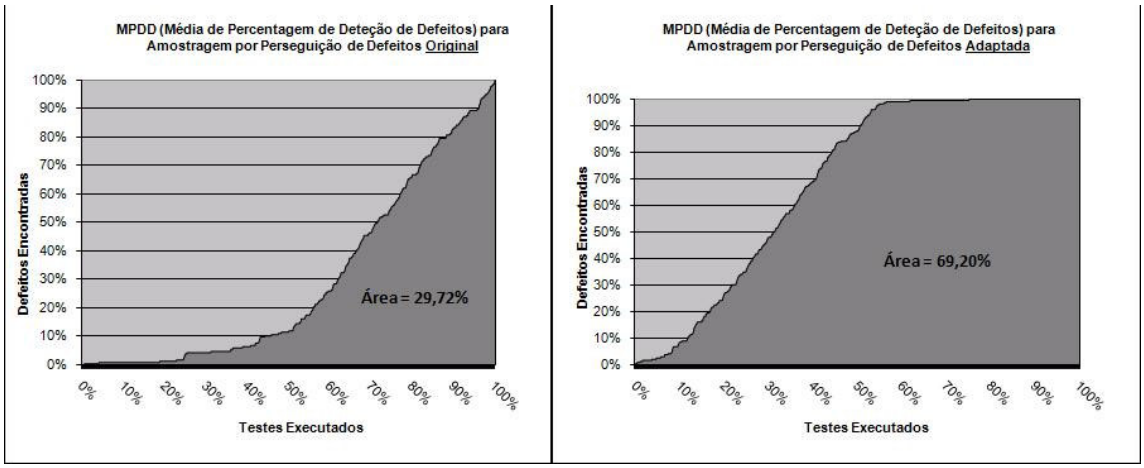


Figura 5.16: Gráficos MPDD da Versão 4 do Programa *Schedule* (Parte 2)

Gráficos MPDD para a Versão 5 do Programa Schedule: Para a versão 5 do programa *schedule* (usando as informações sobre os testes que revelaram defeitos na versão 4) também se tem um ótimo resultado de otimização. Nesta versão percebe-se que se os CTs fossem executados na sequência em que foram definidos (Figura 5.17), a execução dos CTs precisaria ser quase completamente realizada (1934 CTs, 73% do todo) para que algum dos 40 CTs que revelam defeitos na versão 5 fosse utilizado. E estes 40 CTs são então executados apenas nos últimos 25% do GT.

Com a ordenação do método original consegue-se uma progressão mediana da utilização dos testes prioritários (Figura 5.18). Porém, ao se usar o método desenvolvido, ao invés de deixar o acaso produzir o resultado, consegue-se novamente uma otimização onde todos os 40 CTs que revelam defeitos na versão 5 são executados quando o progresso dos testes atinge a marca de 62% do todo (1655 CTs executados).

Uma melhora de aproximadamente 60 pontos percentuais em relação à sequência original e aproximadamente 22 pontos percentuais em relação ao método original.

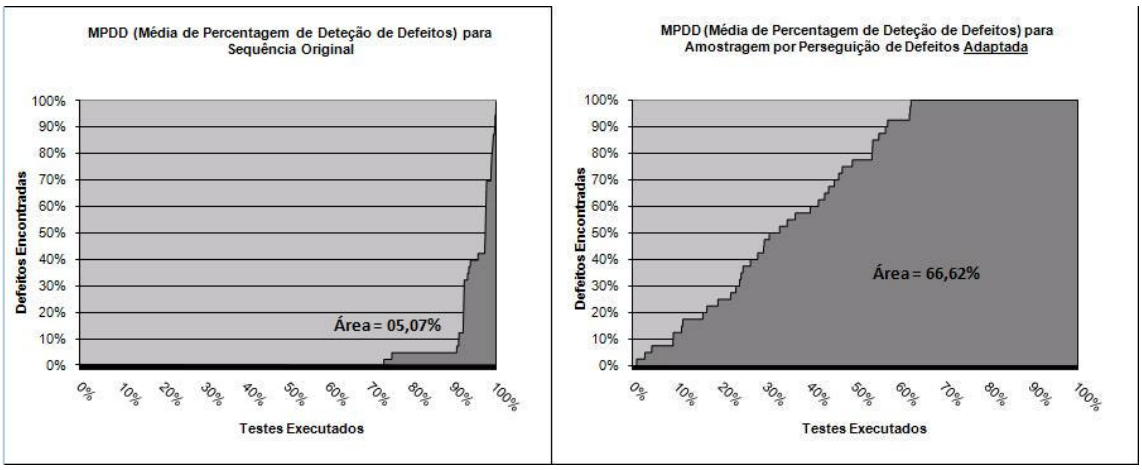


Figura 5.17: Gráficos MPDD da Versão 5 do Programa *Schedule* (Parte 1)

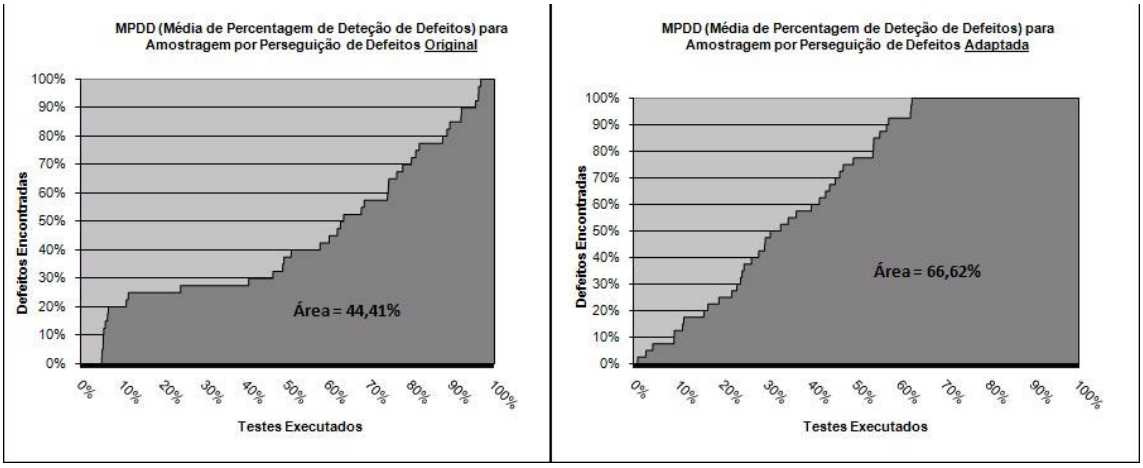


Figura 5.18: Gráficos MPDD da Versão 5 do Programa *Schedule* (Parte 2)

Gráficos MPDD para a Versão 6 do Programa Schedule: Para a versão 6 do programa *schedule* (usando as informações sobre os testes que revelaram defeitos na versão 5), apesar de haver uma otimização de aproximadamente 12,5 pontos percentuais em relação aos testes executados na ordem original (Figura 5.19), o primeiro CT que encontra defeitos é executado aos 59% (1572) do todo e todos os CTs que encontram defeitos na versão 6 são executados aos 87% (2296), enquanto na ordem original, o primeiro CT a encontrar um defeito é executado aos 90% do todo (2397) e os 5 CTs que encontram defeitos nesta versão só são executados aos 96% (2540) – essa "eficiência" é falsa, como pode ser observado quando o resultado do método desenvolvido é comparado ao resultado do método original (Figura 5.20).

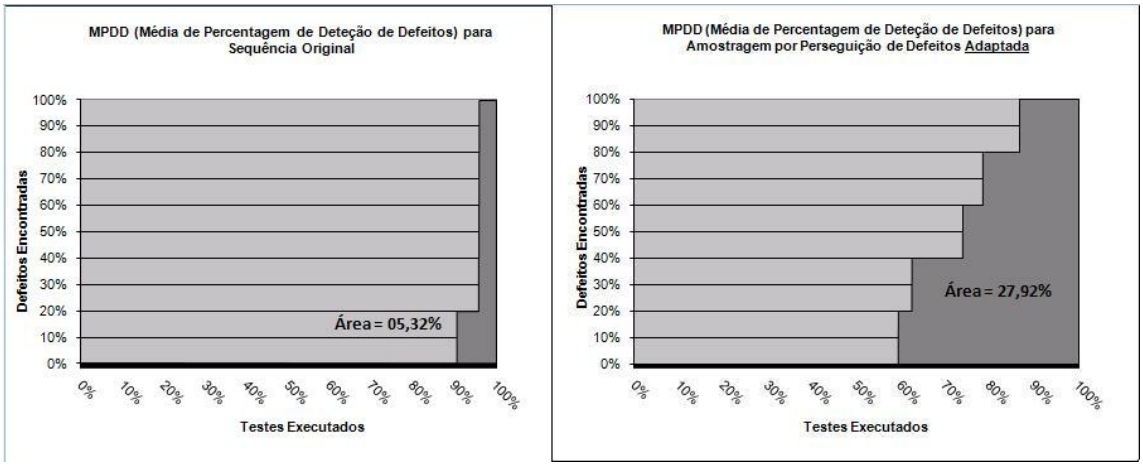


Figura 5.19: Gráficos MPDD da Versão 6 do Programa *Schedule* (Parte 1)

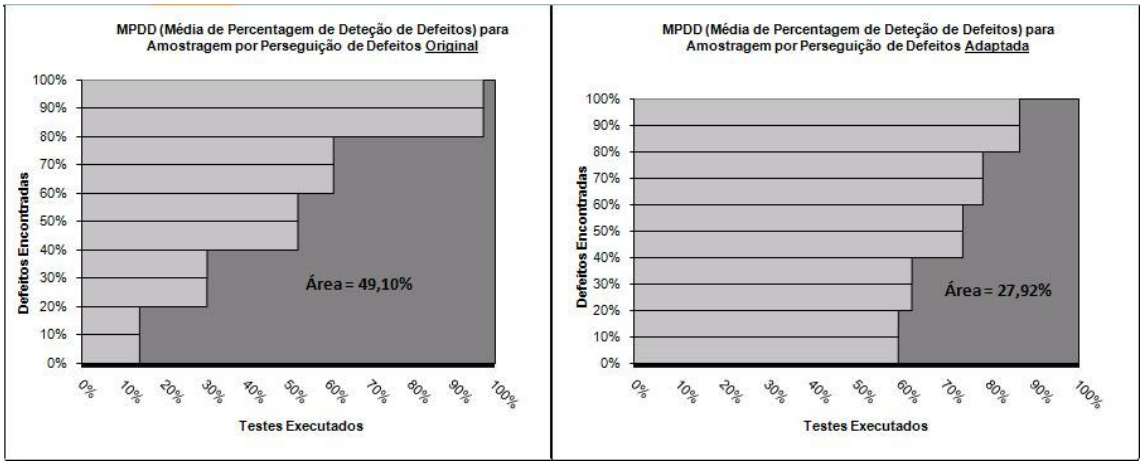


Figura 5.20: Gráficos MPDD da Versão 6 do Programa *Schedule* (Parte 2)

Este resultado pode ocorrer por uma série de motivos. Primeiramente, se o parâmetro de poda do dendograma formado pelo Agrupamento Hierárquico Aglomerativo não se encontra bem ajustado, fixando o número final de agrupamentos em um momento muito prematuro, ou seja, mantendo os TCs que revelaram defeitos na versão 5 e os TCs que revelam defeitos na versão 6 em agrupamentos distintos.

Por outro lado, se for necessário diminuir demais o número de agrupamentos resultantes, isso pode significar que há algo de errado com a relação entre as informações dos TCs que revelam defeitos das duas versões adjacentes. Por exemplo, os defeitos da versão 6 em questão não serem defeitos de regressão, ou seja, não terem sido produzidas por uma correção de defeito na versão anterior. Ou até mesmo, algum tipo de problema gerado pela

produção artificial dos defeitos, como não simularem de maneira apropriada a relação inerente a um defeito de regressão.

É necessário ter cuidado quando se seleciona uma situação com um número menor de agrupamentos resultantes, pois a cada passo a mais do algoritmo, mais e mais TCs que não revelam defeitos podem vir a ser misturados com os que revelam, diluindo a prioridade dos agrupamentos como um todo.

Gráficos MPDD para a Versão 7 do Programa Schedule: Para a versão 7 do programa *schedule* (usando as informações sobre os testes que revelaram defeitos na versão 6), tem-se um resultado bastante interessante se analisado mais a fundo.

De maneira análoga aos resultados conseguidos para a versão 3 tem-se também uma divisão dos CTs que encontram defeitos (Figura 5.21 e Figura 5.22) – embora muito mais precoce nesse caso - onde 5 CTs que encontram defeitos são rapidamente priorizados e executados, até 12% do total (321), pois são identificados usando informações dos CTs que encontraram defeitos na versão 6. Mas os demais 21 CTs não o são (a já conhecida questão da poda precoce do dendograma).

Porém, percebe-se que uma vez que os CTs que revelam defeitos na versão 7 começam novamente a ser executados, uma agressiva taxa de execução se impõe, fazendo com que todos os 27 CTs que revelam defeitos sejam executados até que 91% dos testes (2414) se completem. Um resultado tão bom quanto o da ordenação usando o método original (Figura 5.22).

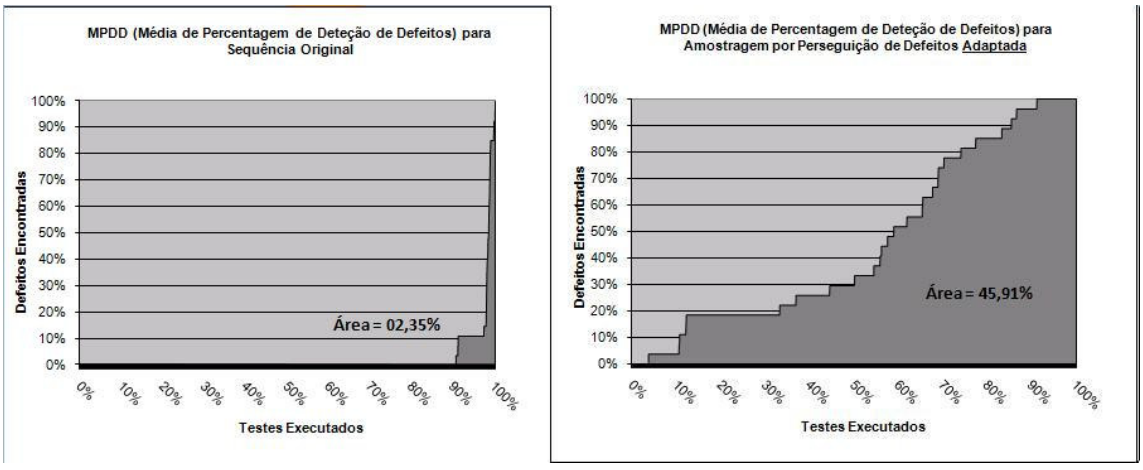


Figura 5.21: Gráficos MPDD da Versão 7 do Programa *Schedule* (Parte 1)

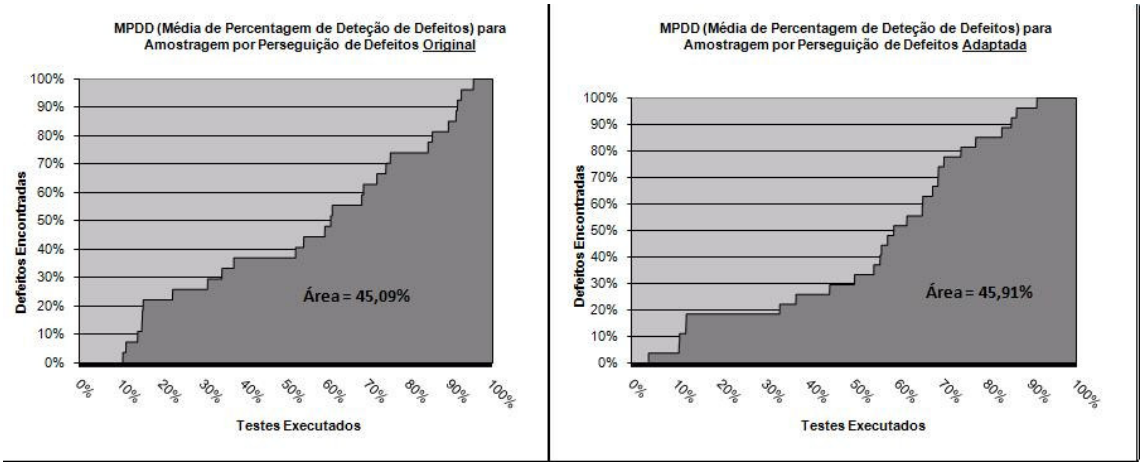


Figura 5.22: Gráficos MPDD da Versão 7 do Programa *Schedule* (Parte 2)

Mas a questão digna de nota neste resultado é que, esta taxa mais agressiva de execução de CTs que encontram defeitos está baseada no parâmetro dos vizinhos, ou seja, uma vez que o método encontre um CT que revele um defeito, este selecionará mais 5 CTs do mesmo agrupamento e os executará. Se algum destes também revelar algum defeito, mais 5 CTs do mesmo agrupamento serão selecionados e assim por diante. O método só voltará a executar CTs de outro agrupamento, quando os 5 vizinhos selecionados não revelarem defeito algum.

Devido ao fato de alguns agrupamentos da poda do dendograma em 46 agrupamentos – poda 46 e Vizinhos 5 - conterem grandes números de CTs (Por exemplo: agrupamentos com 116 CTs, 153 CTs e até mesmo um agrupamento com 1482 CTs – representados na Figura 5.12) é bastante provável que uma parâmetro de "*k*" vizinhos de valor maior do que 5 seja mais eficiente, pois aumentaria a probabilidade de que CTs vizinhos que encontrem defeitos serem encontrados, distribuídos em um agrupamento de um CT que acaba de revelar um defeito.

Gráficos MPDD para a Versão 8 do Programa Schedule: Os resultados para a versão 8 do programa *schedule* (usando as informações sobre os testes que revelaram defeitos na versão 7), também apresentam dados bastante otimizados para o método desenvolvido.

Uma melhora de aproximadamente 17 pontos percentuais em relação à sequência original e aproximadamente 43 pontos percentuais em relação ao método original.

Enquanto a ordem original e a ordenação pelo método original (Figura 5.23 e Figura 5.24 respectivamente), voltam a apresentar progressões ineficientes dos CTs que revelam

defeitos (um total de 30 CTs), o método desenvolvido encontra todos os 30 CTs que encontram defeitos quando o progresso dos testes chega a 58% (1526 CTs) do total.

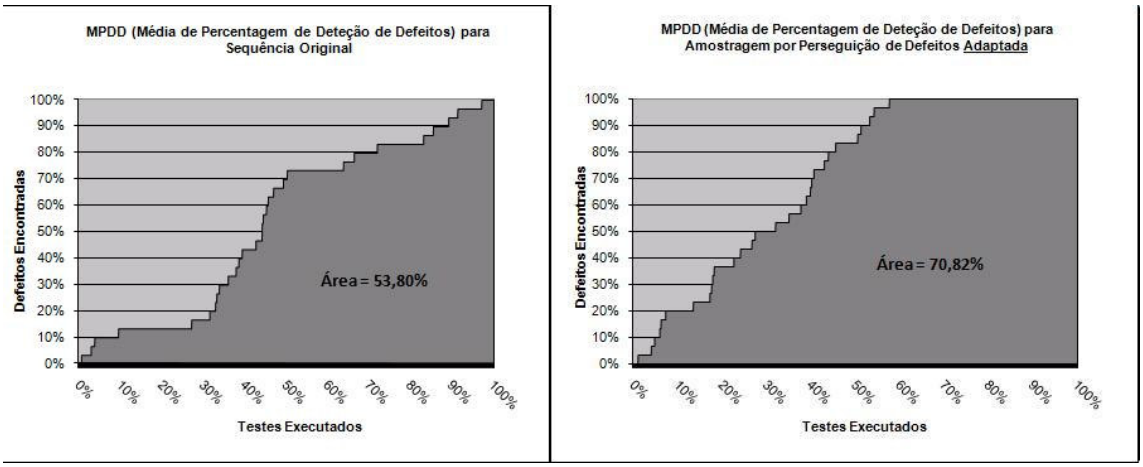


Figura 5.23: Gráficos MPDD da Versão 8 do Programa *Schedule* (Parte 1)

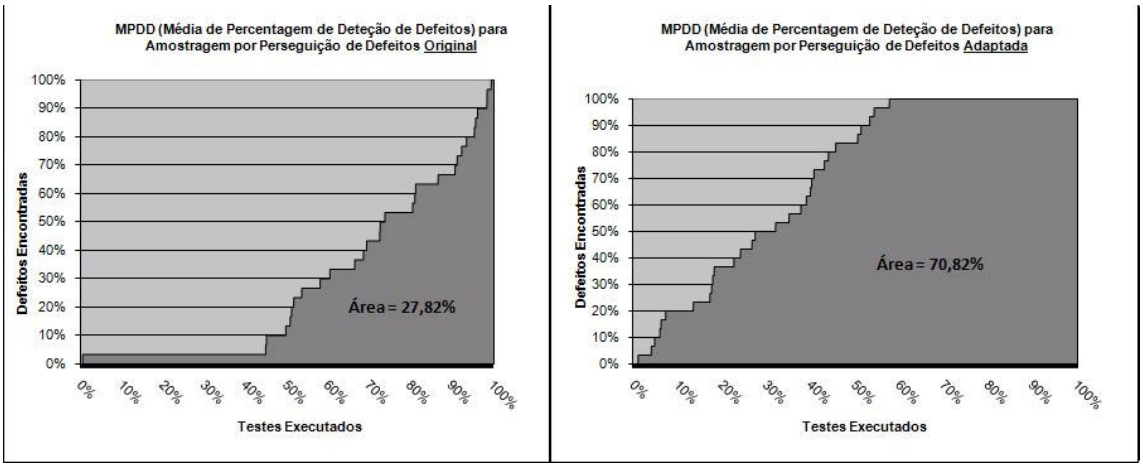


Figura 5.24: Gráficos MPDD da Versão 8 do Programa *Schedule* (Parte 2)

5.3.2. Resultados do Programa *Schedule* usando Poda 7 e 50 Vizinhos

Uma vez de posse dos resultados para a parametrização usada na Seção 5.3.1 – Poda 46 e 5 Vizinhos – optou-se por observar o desempenho do método desenvolvido no outro extremo da parametrização.

Teorizando-se que a poda precoce do dendograma levou ao não agrupamento dos CTs que detectam defeitos entre algumas das versões do programa *schedule,* decidiu-se por decrementar o parâmetro de poda de 46 para 7 agrupamentos restantes (Figura 5.25) –

visualmente “cortando” o dendograma a dois níveis da raiz - objetivando garantir que a relação entre os testes que evidenciam defeitos nas versões adjacentes sejam contempladas. A decisão baseou-se na familiaridade obtida com os dados após analisá-los a fundo durante a geração dos resultados da Seção 5.3.1.

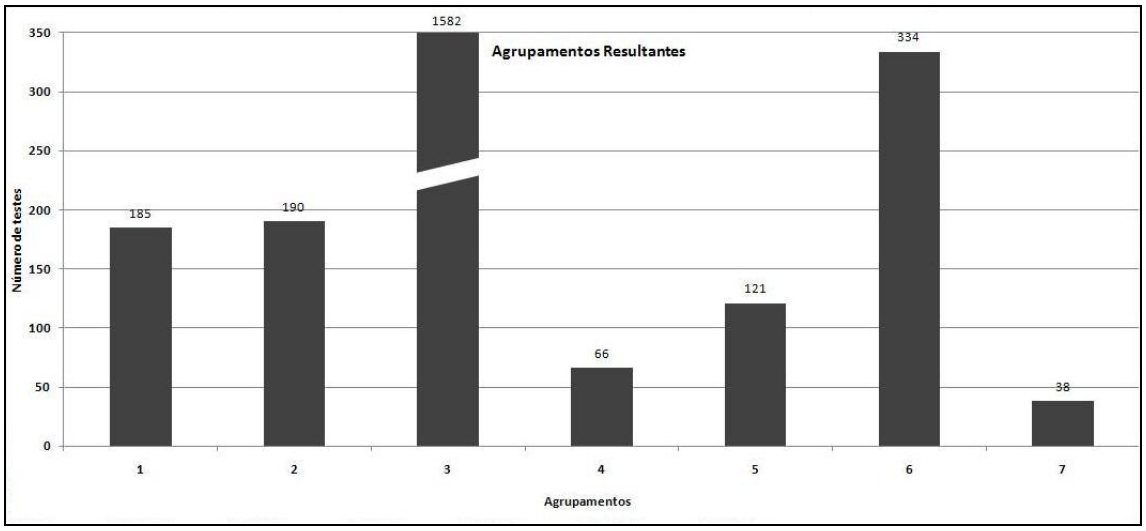


Figura 5.25: Agrupamentos do Programa *Schedule* Usando Nível de Poda 7.

Da mesma forma, teorizou-se que o valor de 5 vizinhos utilizado na literatura base trata-se de um valor “tímido” oriundo do fato de que o método original – Amostragem por Perseguição de Defeitos – não possui as informações prévias de quais são os agrupamentos “suspeitos” de conterem defeitos. O método original precisa buscar os defeitos cuidadosamente. Em contra partida, o método desenvolvido – alimentado com informações que o direcionam aos agrupamentos de interesse – pode se “dar ao luxo” de dedicar mais atenção a estes, em detrimento de postergar a amostragem nos demais agrupamentos não considerados “suspeitos”.

Os resultados a seguir foram gerados para três casos distintos – colhidos da Seção 5.3.1:

a. Para a versão 3 – um caso onde houve uma otimização “quebrada” – para observar o impacto gerado pela alteração dos parâmetros.
b. Para a versão 4 – um caso onde houve uma boa otimização – para garantir que a alteração dos parâmetros – se não melhorar o resultado – não impacte os gráficos de maneira negativa.

c. Para a versão 6 - um caso onde não houve otimização - para observar como a poda do dendograma em uma situação posterior pode alterar o resultado de direcionamento da eficiência do método desenvolvido.

Os resultados desta fase estão sumarizados na Tabela 5.3, porém os valores individuais são discutidos adiante nesta seção.

Versões do Programa Schedule	Número Total de Testes que Revelam Defeitos	Ordenação com Poda 46 e 5 Vizinhos	Ordenação com Poda 7 e 50 Vizinhos	Observação
Versão 3	138 de 2650	66,10%	63,35%	Sem alteração
Versão 4	235 de 2650	69,20%	70,92%	Sem alteração
Versão 6	5 de 2650	27,92%	82,02%	Boa otimização

Tabela 5.3: Sumário de Resultados Obtidos para Poda 46 e Vizinhos 5

Gráfico MPDD para a Versão 3 do Programa Schedule: O resultado da alteração de parâmetros para a versão 3 do programa *Schedule* não apresenta nenhuma diferença significativa, uma deterioração de menos de 3 pontos percentuais. Esta pequena alteração na curva MPDD – Figura 5.26 - baseia-se no fato de o método desenvolvido conter componentes de aleatório, inerentes a parte de seu processamento. Porém, apesar de pequena diferença, a alteração realizada no parâmetro Vizinhos (de 5 para 50) não alterou de maneira impactante a taxa de execução dos CTs que revela defeitos (inclinação da curva MPDD), de fato, a escalada da curva é até mais lenta devido ao número de CTs que não revelam defeitos que foram inseridos nos agrupamentos devido à poda tardia do dendograma.

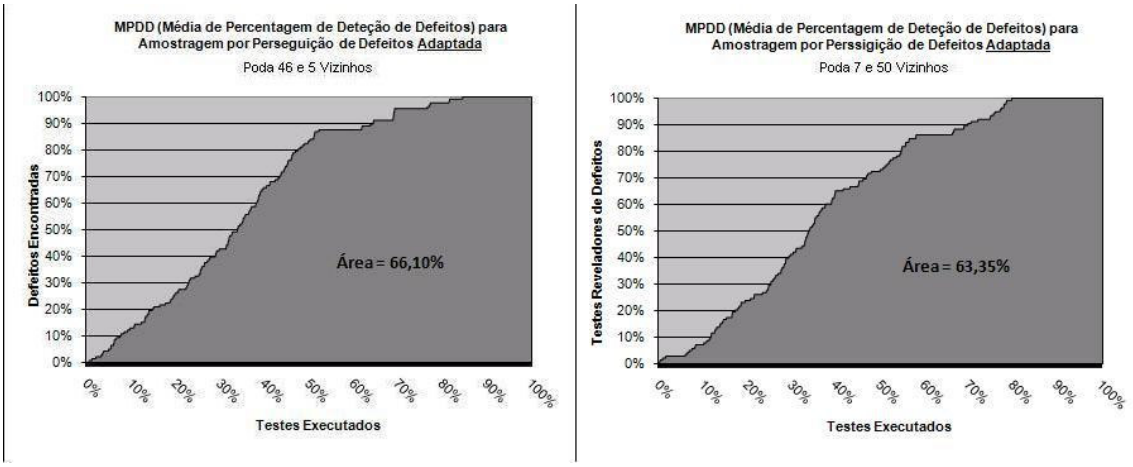


Figura 5.26: Gráficos MPDD das APDs da versão 3 do Programa *Schedule*.

Também a alteração do parâmetro Poda (de 46 agrupamentos para 7) – apesar da aparente suavização da "quebra" na curva do gráfico MPDD, também não obteve sucesso ao tentar incluir a totalidade dos CTs que revelam defeitos na versão 3. Parte dos CTs não foi identificada como prioritária, reforçando a teoria de que os mesmos não contêm a relação de regressão necessária para o mapeamento inter-versões proposto.

Gráfico MPDD para a Versão 4 do Programa Schedule: Para o caso da versão 4 do programa *schedule* - onde o resultado já fora satisfatório na Seção 5.3.1 – pode-se ver uma melhora insignificante de pouco mais de 1 ponto percentualde da curva MPDD – Figura 5.27. Todos os CTs que revelam defeitos (um total de 235) são encontrados na marca de 72% (1911), ao invés de 75% (1989 CTs) como na situação anterior.

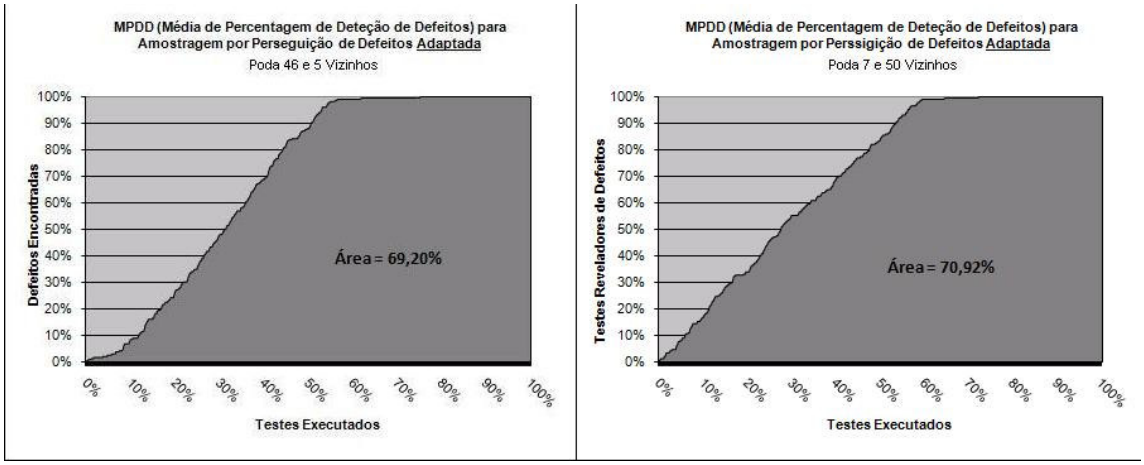


Figura 5.27: Gráficos MPDD das APDs da versão 4 do Programa *Schedule*.

No entanto, essa melhora é muito mais significativa do que aparenta. Se apenas o parâmetro de Vizinhos tivesse sido alterado, certamente o ganho teria sido maior. Porém, como também foi alterado o parâmetro de poda do dendograma – para um número de 7 agrupamentos – vários CTs que não revelam defeitos foram incorporados nos agrupamentos de interesse, e fizeram parte da fase de amostragem.

Gráfico MPDD para a Versão 6 do Programa Schedule: O caso da versão 6 é o que apresenta a melhora mais significativa – aproximadamente 55 pontos percentuais. Como teorizado, pode-se observar que a passagem da poda para um momento posterior foi capaz de integrar as informações da versão anterior (versão 5) nos agrupamentos de interesse, que direcionaram a busca por defeitos na versão 6.

No caso em questão – ao invés de ter o primeiro CT que revela algum defeito executado aos 59% (1572) do todo (Figura 5.28), tem-se a primeira execução aos 16% (427 - perfeitamente aceitável devido à baixa distribuição de 5 CTs que revelam defeitos em um total de 2650). Porém, ainda mais impressionante, é o ritmo agressivo que o parâmetro de 50 Vizinhos impõe uma vez que o primeiro CT que revela defeitos é encontrado, fazendo que todos os 5 CTs sejam revelados na marca de 21% (545) do total – em comparação com os 87% (2296) da situação anterior.

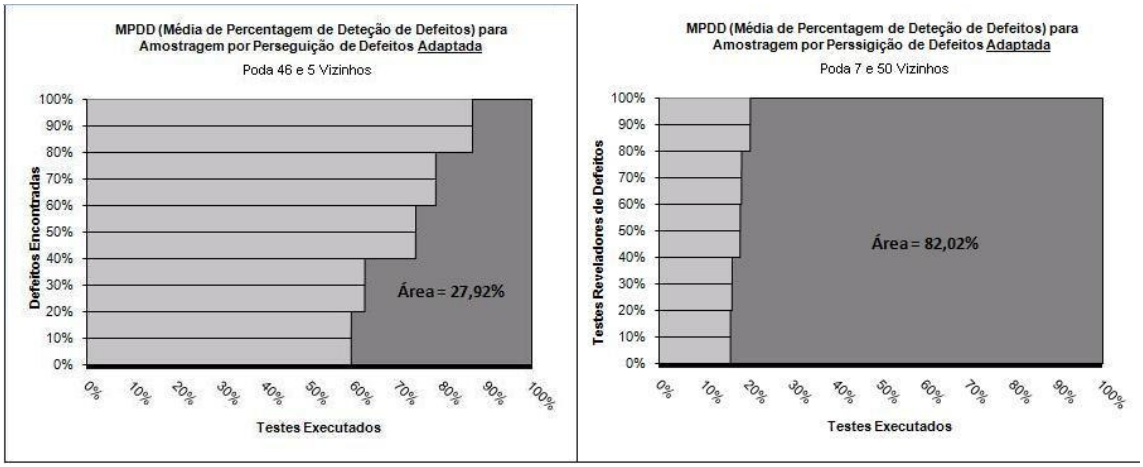


Figura 5.28: Gráficos MPDD das APDs da versão 6 do Programa *Schedule*.

5.4. Dificuldades Encontradas

Durante o desenvolvimento deste estudo algumas ações que haviam sido planejadas ou imaginadas precisaram ser alteradas ou completamente redefinidas, devido a algumas dificuldades encontradas.

Nenhuma Menção às Versões dos Programas Siemens: A idéia inicial deste estudo foi utilizar-se de programas e técnicas encontradas na literatura base como: os Aplicativos Siemens, as Medidas de Percentagem de Detecção de Defeitos (MPDD) e a Amostragem por Perseguição de Defeitos, para poder exatamente fazer-se uso dos dados já gerados por outros experimentos e estudos para comparações, sem precisar efetivamente repeti-los. Infelizmente, nenhum dos trabalhos imaginados como comparativos em potencial faz qualquer menção às diferentes versões dos Programas Siemens utilizados, tornando qualquer comparação bastante subjetiva.

A Origem dos Dados é Artificial e não Real: Apesar do bom resultado apresentado pelo método desenvolvido, os dados que foram usados no estudo são de fonte artificial, ou seja, as versões de código tiveram seus defeitos propositalmente inseridos.

Ainda existe a questão de como o método desenvolvido se comportará frente a dados de entrada reais, porém, como o método está baseado na relação entre a descrição dos testes de uma versão, para com os testes de uma versão anterior – uma relação bastante palpável e bem embasada – acredita-se que, se fosse existira alguma dificuldade, seria a de perder esta relação para versões criadas artificialmente – formadas sem o intuito de preservá-las. Para dados reais, imagina-se possível alcançar resultados ainda melhores.

Uma das dificuldades que foram encontradas durante a aplicação do método – baseada no fato dos dados serem simulados – é que existem testes que revelam defeitos em ambas às versões adjacentes de código. Estes testes foram considerados "*outliers*" e desconsiderados como fonte de informação para o estudo, pois:

a) Não seriam encontrados em um caso real, visto que seriam considerados testes funcionais e não de regressão. O código não seria levado a uma fase de testes de regressão, se os testes que falharam em uma versão anterior continuassem falhando após a correção;
b) Mascarariam o resultado final do estudo – apresentando uma melhora falsa – sendo que seriam testes que certamente estariam nos mesmos agrupamentos devido a suas descrições idênticas (são o mesmo teste);

A Diferença Entre Defeitos e Testes que Encontram Defeitos: De inicio, imaginou-se poder contar com uma Matriz de Defeitos que indicassem quais são os testes do Grupo de Testes (GT) que fossem capazes de encontrar quais defeitos específicos. Ao invés disso, tem-se como dado de entrada uma Matriz de Defeitos que indica apenas quais os testes do GT que indicam defeitos - e não qual defeito específico é relacionado ao Caso de Teste (CT) em questão.

A dificuldade que essa diferença de dado de entrada traz – que não impossibilita o uso do método em si, apenas poderia mascarar certas comparações com outros métodos – é de que não se sabe na realidade quantos defeitos foram encontrados no programa, visto que alguns CTs – ou até todos – podem estar revelando o mesmo defeito no Programa.

As Versões 2 e 9 do Programa *Schedule* Consideradas "*Outliers*": As versões 2 e 9 do programa *schedule* são consideradas "*outliers*"- dados que fogem do padrão e não trazem ganhos se fizerem parte do estudo.

No caso da versão 2 os testes que revelam defeitos e os testes que revelam defeitos na versão anterior (versão 1), possuem descrições de comportamento tão diferentes entre si que sugerem que nenhum cuidado foi tomado – para manter a relação inerente a um defeito de regressão – no momento em que a correção do defeito anterior e o defeito artificial foram produzidos de uma versão para a outra.

Para a versão, a grande maioria dos CTs da lista do GT, ao passarem pelo programa, não executam função alguma. Isso significa que a alteração de código feita para gerar a versão 9 (sobre a versão 8) impossibilita o código de ser executado. O programa tem início, e antes de executar qualquer função, termina. Esse comportamento, portanto, elimina qualquer relação que os CTs poderiam ter desta versão para a anterior, não fazendo sentido considerá-la como exemplo.

5.5. Trabalhos Relacionados

Vários estudos envolvendo Testes de Regressão Otimizados (TRO) vêm sendo realizados recentemente em âmbito mundial.

Na Escola de Engenharia Elétrica e Ciências Computacionais da Universidade do Estado do Oregon em Corvallis, Gregg Rothermel conduz - há mais de uma década - estudos de TRO usando diferentes técnicas de várias metodologias - Refazer Todos os testes (RTT), Seleção de Testes de Regressão (STR), Redução de Grupo de Testes (RGT) e Priorização de Casos de Testes (PCT). Seus resultados oferecem comparações bastante interessantes sobre os diferentes desempenhos de cada técnica e metodologia, por serem usados sempre sobre a mesma lista de Programas Siemens (ROTHERMEL e HARROLD, 1997), (ROTHERMEL et. al., 2001), (ROTHERMEL et. al., 2004).

Os resultados de Rothermel dizem respeito a dois algoritmos de STR que foram desenvolvidos sob sua orientação (DejaVu1 e DejaVu2 – algoritmos que usam técnicas baseadas em Segurança para garantir 100% de cobertura de código). Seus trabalhos demonstram a eficiência destes algoritmos quando comparados com outros similares (ROTHERMEL e HARROLD, 1997) ou outras técnicas de TRO (ROTHERMEL et. al., 2004).

Rothermel também apresenta resultados na área de PCT onde adota estratégias baseadas em Segurança e novamente – lançando mão dos Programas Siemens para dar continuidade as suas comparações – foca seus estudos na cobertura de código usando Técnicas Baseadas em Cobertura (TBC) com ótimos resultados, usando instrumentação de código (ROTHERMEL et. al., 2000). Um dos melhores resultados gerados neste campo de trabalho é a sua já citada medida de Média de Percentagem de Detecção de Defeitos (MPDD - usada por este estudo para expor seus resultados).

Dando sequência ao trabalho de Rothermel encontramos Sebastian Elbaum, do Departamento de Ciências Computacionais e Engenharias da Universidade de Nebraska em Lincoln, que também está conduzindo estudos similares usando a citada lista de Programas Siemens para efeito de comparação. A diferença é que Elbaum inclui ainda um aplicativo maior chamado *space* desenvolvido pela Agencia Espacial Européia (contendo por volta de 6000 linhas) (ELBAUM et. al., 2002); (ELBAUM et. al., 2004). A sua comparação de 18 técnicas específicas de PCT é bastante interessante e foi usada para embasar a metodologia deste estudo.

Rothermel (e Mary Jean Harrold) ainda apresentam trabalhos de STR empregando alvos de estudo mais complexos na área de Orientação a Objeto para C++ (ROTHERMEL et. al., 2000) e Java (HARROLD et. al., 2001). Bons exemplos de base para – no futuro - aumentar-se o escopo e a complexidade dos dados de entrada para o problema apresentado por este estudo.

Em Bangalore – Índia - está sendo realizado um trabalho que diz respeito ao uso de um protótipo denominado InARTS que determina o impacto de atualizações em aplicações .NET e sugere uma redução de TR (PASALA et. al., 2008). É interessante observar sobre esse projeto, que o mesmo foi desenvolvido sobre uma óptica de que nem o código fonte do software testado, nem as alterações específicas efetuadas entre versões são necessários para o seu funcionamento. O aplicativo faz uso de uma análise do comportamento dinâmico do software sob teste. Esta abordagem traz uma vantagem inerente por sugerir um escopo reduzido de testes baseado no real uso prático de um aplicativo, possuindo uma boa precisão. Porém esta vantagem também reduz a sua aplicação a componentes de código que têm uma pequena variedade de diferentes funções.

Já os trabalhos mais próximos a este estudo são descritos a seguir.

Elaborado por David Leon e Andy Podgurski (LEON e PODGURSKI, 2003). Em seu experimento, apresentam o resultado de uma comparação entre algumas técnicas de PCT - Minimização de GT, Priorização por Cobertura Adicional, Filtragem de Agrupamentos por Amostragem de Um–por-Agrupamento e Amostragem por Perseguição de Defeitos (APD) - duas das quais pertencentes à subdivisão de TBD. Porém, o estudo faz referência a testes de sistemas em geral e não especificamente a TR. Não é incluída nenhuma referência a dados provenientes de versões anteriores para priorizar os CTs.

No Centro de Pesquisa em Evolução, Busca e Testes da Universidade Real de Londres Shin Yoo e Mark Harman, em conjunto com Paolo Tonella e Angelo Susi em Trento na – Itália – combinam esforços para desenvolver uma pesquisa para – utilizando-se de algoritmo denominado Processo Hierárquico Analítico – reduzir um número grande de CTs, (na ordem de milhares), até um número de 100 agrupamentos de CTs similares, para que possam então ser priorizados por esforço humano (YOO et. al., 2009). O trabalho em questão usa o comportamento dinâmico dos testes – aos serem executados no código – para formar agrupamentos de testes similares cuja apenas um elemento de cada cluster possa ser usado na comparação, representando o comportamento do cluster Omo um todo. Dessa forma milhares de comparações potencialmente necessárias são diminuídas para um número limite. Essencialmente, o método de Shin Yoo e Mark Harman é bastante próximo do trabalho apresentado nesta dissertação, porém, este trabalho vai além ao propor uma priorização final também automática.

5.6. Considerações Finais

Com a descrição do capítulo 5 realizada, todos os aspectos do método desenvolvido estão devidamente apresentados. O documento é iniciado pelos capítulos 2 e 3, que descrevem a base teórica necessária para o entendimento do método desenvolvido – descrevendo os Testes de Regressão e a Tarefa de Agrupamento respectivamente – no capítulo 4 tem-se então a descrição do método de Priorização de Casos de Testes de Regressão usando Amostragem por Perseguição de Defeitos e, por fim, o capítulo 5 que enuncia a aplicação do método e os resultados obtidos.

Os passos realizados e os resultados obtidos estão dispostos de maneira cronológica, o que facilita o entendimento das interpretações e ajustes feitos durante a aplicação do método desenvolvido e da geração de resultados. Apesar de algumas conclusões terem sido traçadas

no capítulo 5, estas são feitas individualmente sobre cada resultado encontrado. Uma conclusão geral sobre o estudo é apresentada no capítulo a seguir.

O capítulo 5 ainda apresentou uma visão detalhada sobre os problemas encontrados durante a aplicação do método desenvolvido, assim como uma seção dedicada aos trabalhos relacionados.

Capítulo 6

Conclusão

Dando continuidade a sequência lógica dos capítulos – capítulos 2 e 3 contendo o embasamento teórico necessário para o entendimento dos temas que envolvem o método, capítulo 4 contendo a descrição da metodologia utilizada e o método desenvolvido, e o capítulo 5, descrevendo os resultados obtidos com a aplicação do método; o capítulo 6 encerra o documento resultante do estudo em questão.

Este capítulo tem por objetivo evidenciar a comprovação da hipótese inicialmente proposta e comentar os passos dados até a conclusão do estudo.

6.1. Conclusão

O objetivo principal desta pesquisa foi validar a proposta de uma técnica de priorização de casos de testes - adaptando uma técnica existente já comprovada para usos em testes em geral - para o uso específico em testes de regressão.

A hipótese inicialmente enunciada neste documento foi comprovada. A estratégia de adaptar a técnica de priorização de casos de testes conhecida por Amostragem por Perseguição de Defeitos (LEON e PODGURSKI, 2003) – exclusivamente para teste de regressão – foi bastante eficiente. Ao se referir aos resultados obtidos percebe-se que, de modo geral, os resultados são superiores aos conseguidos sem o uso de uma técnica de otimização.

Já de maneira mais específica, fica comprovado o fato de que - se parametrizado de forma correta - o método pode trazer uma grande eficiência de otimização como por exemplo nas versões 5 do programa *schedule* – onde todos os testes que têm a capacidade de revelar algum defeito são executados até 62% do todo – e a versão 8 do programa *schedule* – apresentando uma execução de todos os testes que precisariam ser priorizados até 58% do todo. Adicionalmente, familiarizando-se com o caso de teste específico é possível ainda ajustar os parâmetros do método para conseguir resultados ainda melhores, como por exemplo o caso da versão 6 onde foi possível executar todos os testes significativos apenas executando-se 21% dos testes totais.

6.2. Perspectivas Futuras

O estudo finalizado, apesar de alcançar os objetivos propostos – geral e específicos – ainda revelou algumas perspectivas futuras que poderiam ser desenvolvidas para acrescentar resultados interessantes para o tema específico discutido neste documento.

Comparar a Eficiência do Método Desenvolvido a Outros Métodos: Uma perspectiva com um bom potencial de conseguir informações interessantes viria de se comparar os resultados obtidos neste estudo com os de outros métodos de otimização de testes de regressão. Como comentado anteriormente, uma comparação imediata não foi possível devido ao fato de – apesar de alguns trabalhos relacionados usarem os mesmos programas alvos e os mesmos gráficos comparativos – estes não se referem, em momento algum, às diferentes versões dos programas usados, tornando qualquer comparação direta muito subjetiva. Seria

necessário, portanto, conseguir as informações pertinentes a cada versão distinta, ou produzi-las, executando os métodos novamente.

Aplicar o Método Desenvolvido sobre Dados Reais: Uma pergunta que se imagina a resposta, porém que ficou em aberto por não terem-se gerado dados específicos, é a questão sobre a eficiência do método desenvolvido em relação a dados reais (e não criados artificialmente). Seria necessário obter uma base de dados real, contendo informações sobre suas diferentes versões de código, defeitos revelados, base de testes, assim como os demais dados usados neste estudo.

Aplicar o Método Desenvolvido sobre Programas Mais Complexos: Uma ótima linha de estudo também seria aplicar o método desenvolvido e validar sua eficiência (e principalmente o nível de processamento) em relação a programas mais complexos, com mais arquivos e funções – talvez Orientados a Objetos.

Referências

(ABONYI e FEIL, 2007) ABONYI, J.; FEIL, B. *Cluster analysis for data mining and system identification.* Boston and Basel, Switzerland: Birkhäuser Basel, 2007.

(BROOKS, 1995) BROOKS, P. *The Mythical Man-Month: Essays on Software Engineering*, *20th Anniversary Edition*. Reading, MA: Addison-Wesley, 1995.

(DICKINSON et. al., 2001a) DICKINSON, D.; LEON D.; PODGURSKI, A. *Finding Failures by Clustering Analysis of Execution Profiles.* ICSE'01, Toronto, Canada, IEEE 2001, pp. 339-348.

(DICKINSON et. al., 2001b) DICKINSON, D.; LEON, D.; PODGURSKI, A. *Pursuing Failure: The Distribution of Program Failures in a Profile Space.* 9o ACM SIGSOFT, Vienna Austria, 2001, pp. 246 - 255.

(DUNOP et. al., 2001) DUNOP, C.; HICKEN, W.; KOLAWA, A. *Bulletproofing Web Applications.* Hungry Minds, 2001.

(DUSTIN et. al., 1999) DUSTIN, E.; RASHKA, J.; PAUL, J. *Automated Software Testing: Introduction, Management, and Performance*. Addison-Wesley Professional; Bk&CD Rom edition, 1999.

(DUSTIN, 2002) DUSTIN, E. *Effective Software Testing: 50 Specific Ways to Improve Your Testing*. Addison-Wesley Professional, 1a Edição. 2002.

(EISEN, 2009) EISEN, M. *TreeView: Graphically browse results of clustering.* Disponível em <http://rana.lbl.gov/EisenSoftware.htm>, acesso em 15/11/2009.

(ELBAUM et. al., 2002) ELBAUM, S.; MALISHEVSKY A.; ROTHERMEL, G. *Test Case Priorization: A Family of Empirical Studies.* IEE Transactions on Software Engineering, Vol.28, nº2, 2002, pp. 159-182.

(ELBAUM et. al., 2004) ELBAUM, S.; ROTHERMEL, G.; KANDURI, S.; MALISHEVSKY, A. *Selecting a Cost-Effective Test Case Prioritization Technique.* Software Quality Journal, 12(3), 2004, pp.185-210.

(ELFRIEDE, 2003) ELFRIEDE, D. *Automate Regression Tests When Feasible, Automated Testing: Selected Best Practices*. Pearson Education, Inc, 2003.

(FEWSTER e GRAHAM, 1999) FEWSTER, M.; GRAHAM, D. *Software Test Automation.* Addison-wesley Professional, 1a Edição, 1999.

(GARNER, 1995) GARNER, S.; *WEKA: The Waikato Environment for Knowledge Analysis.* The Second New Zealand Computer Science Research Students' Conference, Universidade de Waikato, Hamilton, 1995.

(GASCUEL, 2009) GASCUEL, O. *PermutMatrix A Software for analyzing and visualizing data*. Disponível em <http://www.lirmm.fr/~caraux/PermutMatrix/>, acesso em 15/11/2009.

(GRAVES et. al., 2001) GRAVES, T.; HARROLD, M.; KIM, J.; PORTER, A. ROTHERMEL, G.; *An Empirical Study of Regression Test Selection Techniques.* ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 10, nº 2, 2001, pp. 184 – 208.

(HAMMING, 1950) HAMMING, R.; *Error Detecting and Error Correcting Codes.* Bell System Technical Journal 26(2):147-160, 1950.

(HARROLD et. al., 2001) HARROLD, M.; JONES, A.; LI, T. LIANG, D.; *Regression Test Selection for Java Software.* ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Londres, Inglaterra, 2001, pp. 312-326.

(JAIN e DUBES, 1988) JAIN, A.; DUBES, R. *Algorithms for Clustering Data*. Prentice Hall, 1988.

(JAIN et. al., 1999) JAIN, A.; MURTY, M.; FLYNN, P. *Data Clustering: A Review*. ACM Computing Surveys, Vol 31, Nº3, 1999, pp.264-323.

(KANELLOPOULOS et. al., 2006) KANELLOPOULOS, Y.; DIMOPULOS, T.; TJORTJIS, C.; MAKRIS, C. *Mining Source Code Elements for Comprehending Object-Oriented Systems and Evaluating Their Maintainability.* ACM SIGKDD Explorations Newsletter, Vol.8, nº1, 2006, pp.33-40.

(KARYPIS et. al., 1999) KARYPIS, G.; HAN, E.; KUMAR, V. *CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling*. COMPUTER, 1999, 32, 68-75.

(KARYPIS, 2009) KARYPIS, G. *gCLUTO – Graphical Clustering Toolkit*. Disponível em <http://glaros.dtc.umn.edu/gkhome/cluto/gcluto/download>, acesso em 15/11/2009.

(LEUNG e WHITE, 1990) LEUNG, H.; WHITE, L. *A study of integration testing and software regression at the integration level.* Software Maintenance (ICSM), Tübingen, Alemanha, 1990, pp. 290-301.

(LEON e PODGURSKI, 2003) LEON, D.; PODGURSKI, A. *A Comparison of Coverage-Based and Distribution-Based Techniques for Filtering and Prioritizing Test Cases.* Software Reliability Engineering (ISSRE), 14o Simpósio Internacional em Denver, Colorado, 2003, pp. 442-453.

(MYERS, 2004) MYERS, G. *The Art of Software Testing*. – JohnWiley & Sons, 2a. Edição. 2004.

(PASALA et. al., 2008) PASALA, A.; FUNG, Y.; AKLADIOS, F.; GORTHI, R. *An approach to select regression tests to validate .NET applications upon deployment of components upgrades.* ACM, 1ª Conferencia Anual de Computação de Bangalore, Índia, 2008.

(PRESSMAN, 2009) PRESSMAN, R. *Software Engineering: Practitioner's Approach*. McGraw-Hill, 7a. Edição, 2009.

(RAJA e TRETTER, 2007) RAJA, U.; TRETTER, M. *Predicting Software Outcomes Using Data Mining and Text Mining.* The SAS Global Fórum, Florida, USA 2007, pp 1-9.

(RAKITIN, 2001) RAKITIN, S. *Software Verification and Validation: A Practitioner's Guide.* Artech Computer Science Library, 2001.

(ROTHERMEL e HARROLD, 1997) ROTHERMEL, G.; HARROLD, M. *A Safe, Efficient Regression Test Selection Technique.* ACM Transactions on Software Engineering and Methodology (TOSEM), Vol.6, nº2, 1997, p.173-210.

(ROTHERMEL et. al., 2000) ROTHERMEL, G.; HARROLD, M.; DEDHIA, J. *Regression Test Selection for C++ Software.* Journal of Software Testing, Verification and Reliability, Vol.10, nº2, 2000, pp. 77-109.

(ROTHERMEL et. al., 2001) ROTHERMEL, G.; UNTCH, R.; CHU, C.; HARROLD, M. *Prioritizing Test Cases For Regression Testing.* IEEE Transactions on Software Engineering, Vol.27, nº10, 2001, pp. 929-948.

(ROTHERMEL et. al., 2004) ROTHERMEL, G.; ELBAUM, S.; MALISHEVSKY, A.; KALLAKURI, P.; QIU, X. *On Test Suite Composition and Cost-Effective Regression Testing*. ACM Transactions on Software Engineering and Methodology (TOSEM), Vol.13, n º3, 2004, p.277-331.

(ROTHERMEL et. al., 2009) ROTHERMEL, G.; ELBAUM, S.; KINNEER, A.; HYUNSOOK, D. *Software-artifacts Infrastructure Repository.* Disponível em <http://sir.unl.edu>, acesso em 15/11/2009.

(SCHUWABER, 2004) SCHUWABER, K. *Agile Project Management with Scrum.* Microsoft Press, 2004.

(SOMMERVILLE, 2003) SOMMERVILLE, I. *Engenharia de Software*, 6ª Edição. Pearson Education do Brasil, 2003, p.39, p.51, p.361.

(STEINBACH et. al., 2005) STEINBACH, M.; KUMAR, V.; TAN, P. *Cluster Analysis: Basic Concepts and Algorithms*. Addison-Wesley Companion Book Site, 2005, p.487-568.

(SUNDMARK et. al., 2005) SUNDMARK, D.; PETTERSSON, A.; THANE, H. *Regression Testing of Multi-Tasking Real-Time Systems: A Problem Statement.* ACM SIGBED, Vol.2, nº 2, 2005, p. 31 – 34.

(WEBER et. al., 1990) WEBER, T.; JANSCH-PÔRTO, I.; WEBER, R. *Fundamentos de tolerância a falhas.* Vitória: SBC/UFES - X Congresso da Sociedade Brasileira de Computação, 1990.

(WITTEN e FRANK, 2005) WITTEN, I. H.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Applications*. Elsevier, 2005, 525p.

(XIE e NOTKIN, 2002) XIE, T.; NOTKIN, D. *Macro and Micro Perspectives on Strategic Software Quality Assurance in Resource Constrained Environments.* EDSER-4, Florida, USA, 2002, p.66-70.

(YOO et. al., 2009) YOO, S.; HARMAN, M.; TONELLA, P.; SUSI, A. *Clustering Test Cases to Achieve Effective & Scalable Prioritization Incorporating Expert Knowledge.* ISSTA '09: Proceedings of the 18th International Symposium in Tests and Software Analysis.