



UNIVERSIDADE ESTADUAL DO CEARÁ

MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

MATHEUS HENRIQUE ESTEVES PAIXÃO

**UMA ABORDAGEM BASEADA EM OTIMIZAÇÃO ROBUSTA PARA O PROBLEMA
DO PRÓXIMO RELEASE NA PRESENÇA DE INCERTEZAS**

FORTALEZA - CEARÁ

2014

MATHEUS HENRIQUE ESTEVES PAIXÃO

UMA ABORDAGEM BASEADA EM OTIMIZAÇÃO ROBUSTA PARA O PROBLEMA DO
PRÓXIMO RELEASE NA PRESENÇA DE INCERTEZAS

Dissertação submetida à Comissão Examinadora do Programa de Pós-Graduação Acadêmica em Ciência da Computação da Universidade Estadual do Ceará como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Jerffeson Teixeira de Souza

FORTALEZA - CEARÁ

2014

Dados Internacionais de Catalogação na Publicação

Universidade Estadual do Ceará

Sistema de Bibliotecas

Paixão, Matheus Henrique Esteves.

Uma abordagem baseada em otimização robusta para o problema do próximo release na presença de incertezas [recurso eletrônico] / Matheus Henrique Esteves Paixão. - 2014.

1 CD-ROM: il.; 4 ¾ pol.

CD-ROM contendo o arquivo no formato PDF do trabalho acadêmico com 85 folhas, acondicionado em caixa de DVD Slim (19 x 14 cm x 7 mm).

Dissertação (mestrado acadêmico) - Universidade Estadual do Ceará, Centro de Ciências e Tecnologia, Mestrado Acadêmico em Ciência da Computação, Fortaleza, 2014.

Área de concentração: Engenharia de Software.

Orientação: Prof. Dr. Jerffeson Teixeira de Souza.

1. Problema do Próximo Release. 2. Otimização Robusta. 3. Engenharia de Software Baseada em Busca. I. Título.



UNIVERSIDADE ESTADUAL DO CEARA – UECE
PRO-REITORIA DE POS-GRADUACAO E PESQUISA - PRPGPq
CENTRO DE CIENCIAS E TECNOLOGIA – CCT
Mestrado Acadêmico em Ciência da Computação – MACC

Av. Paranjana, 1700. Bairro: Serrinha. Campus do Itapery Fone: (85) 3101 9776

MACC


ATA DA SEXAGÉSIMA QUINTA DEFESA PÚBLICA DE DISSERTAÇÃO DE MESTRADO



Aos doze dias do mês de agosto de dois mil e catorze, no miniauditório do prédio de Pesquisa e Pós-Graduação em Computação, do Mestrado Acadêmico em Ciência da Computação – MACC, realizou-se a sessão pública de defesa da dissertação de **MATHEUS HENRIQUE ESTEVES PAIXAO**, aluno regularmente matriculado no Mestrado Acadêmico em Ciência da Computação – MACC, tendo como título: “Uma Abordagem Baseada em Otimização Robusta para o Problema do Próximo Release na Presença de Incertezas”. A Banca Examinadora reuniu-se no horário de 8h às 10:00 horas, sendo constituída pelos professores **Prof. Dr. Jerffeson Teixeira de Souza, Ph.D.- orientador e presidente da banca**, **Prof. Dr. Paulo Henrique Mendes Maia**, ambos da Universidade Estadual do Ceará-UECE e o professor **Prof. Dr. Celso Gonçalves Camilo Júnior – Universidade Federal de Goiás -UFG**. Inicialmente o mestrando expôs seu trabalho e a seguir foi submetido à arguição pelos membros da Banca, dispondo cada membro de tempo para tal. Finalmente a Banca reuniu-se em separado e concluiu por considerar o mestrando APROVADO, por sua dissertação e sua defesa pública. Eu, Professor **Dr. Jerffeson Teixeira de Souza**, orientador da dissertação e presidente da Banca, lavro a presente Ata que será assinada por mim e demais membros da Banca. **Fortaleza, 12 de agosto de 2014.**


Prof. Dr. Jerffeson Teixeira de Souza, Ph.D.
Orientador – UECE


Prof. Dr. Paulo Henrique Mendes Maia
UECE


Prof. Dr. Celso Gonçalves Camilo Júnior
Universidade Federal de Goiás - UFG

MATHEUS HENRIQUE ESTEVES PAIXÃO

**UMA ABORDAGEM BASEADA EM OTIMIZAÇÃO ROBUSTA PARA O PROBLEMA
DO PRÓXIMO RELEASE NA PRESENÇA DE INCERTEZAS**

Dissertação submetida à Comissão Examinadora do Programa de Pós-Graduação Acadêmica em Ciência da Computação da Universidade Estadual do Ceará como requisito parcial para obtenção do grau de Mestre.

Aprovada em: 12/08/2014

BANCA EXAMINADORA

Prof. Dr. Jerffeson Teixeira de Souza
(Orientador)
Universidade Estadual do Ceará – UECE

Prof. Dr. Celso Gonçalves Camilo Júnior
Universidade Federal de Goiás – UFG

Prof. Dr. Paulo Henrique Mendes Maia
Universidade Estadual do Ceará – UECE

Essa dissertação é dedicada a você, leitor.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais Paulo e Salete por terem proporcionado um ambiente familiar necessário para um bom desenvolvimento dos meus estudos, desde a educação básica até hoje. Agradeço à toda a minha família em geral, pelo apoio e confiança depositados.

Agradeço meu orientador, prof. Dr. Jerffeson Teixeira de Souza, pela aceitação como orientando; pela apresentação da área de Engenharia de Software Baseada em Busca; por uma orientação sempre inteligente, profissional e pessoal; por todos os artigos escritos em parceria; por sempre tentar ao máximo encontrar recursos para custeio das minhas viagens para apresentações de trabalhos; por me mostrar um pouco da carreira que eu muito provavelmente vou seguir; por me auxiliar em todo o processo de seleção do doutorado; e obviamente, por todas as conversas sobre MMA realizadas.

Agradeço ao meu professor da graduação e orientador do TCC, prof. Dr. Cidley Teixeira de Souza, pelas conversas que me fizeram escolher a carreira acadêmica; pela indicação do Mestrado em Ciência da Computação da UECE; e pelas recomendações formais e informais para minha eventual aceitação.

Agradeço aos professores Dr. Celso Camilo e Dr. Paulo Henrique por aceitarem o convite de participação na minha banca avaliadora.

Agradeço a todos os professores com os quais tive contato e que de fato marcaram minha vida, seja na transferência do conhecimento, no despertar do conhecimento ou numa simples conversa pelos corredores. De forma mais recente, agradeço particularmente ao prof. Dr. Gustavo Campos pelos conhecimentos obtidos na disciplina e mais importante pelos conhecimentos práticos passados ao longo das conversas.

Agradeço a todos os colegas do MACC, por todas as conversas e trocas de experiências. Agradeço em particular aos amigos do GOES (Grupo de Otimização em Engenharia de Software) pelo companheirismo; Thiago Nascimento pelas inúmeras discussões, diferenças de opiniões e disputas pelo poder, Allysson Alex por me apresentar um novo significado para a palavra “esforço”, Italo Yeltsin por ter sido praticamente meu primeiro orientando, Átila Freitas e Alan Bandeira pelas horas de Steam. Não dá pra citar todos, mas de qualquer forma vocês sabem que somos todos GOES.

Ao final mas (definitivamente) não menos importante, acho que agradeço a você Mariana Maia, pelo melhor café de todos.

“You talkin’ to me? You talkin’ to me? You talkin’ to me? Then who the hell else are you talking... you talking to me? Well I’m the only one here. Who the fuck do you think you’re talking to? Oh yeah? OK”

Travis Bickle - Taxi Driver

LISTA DE PUBLICAÇÕES

Durante o período do mestrado alguns artigos científicos foram publicados em conferências e periódicos. A lista é dividida em dois grupos de publicações. As primeiras estão diretamente ligadas ao tema da dissertação e serviram de base para a escrita desse trabalho. O segundo grupo é composto por artigos desenvolvidos com o intuito de consolidar os estudos e conhecimentos relacionados à linha de pesquisa em geral.

Estudos Relacionados à Dissertação:

1. PAIXÃO M.; SOUZA J. A Robust Optimization Approach to the Next Release Problem in the Presence of Uncertainties. Aceito para publicação na próxima edição do Journal of Systems and Software – JSS.
2. PAIXÃO, M.; SOUZA, J. A scenario-based robust model for the next release problem. In: ACM. Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference, GECCO 2013. p. 1469–1476.
3. PAIXÃO, M.; SOUZA, J. A recoverable robust approach for the next release problem. In: Search Based Software Engineering. Springer, SSBSE 2013. p. 172–187.

Estudos Relacionados à Linha de Pesquisa em Geral:

1. PAIXÃO, M.; SOUZA J.; BRASIL M.; NEPOMUCENO T; Aplicando o algoritmo ant-q na priorização de requisitos de software com precedência. No III Workshop de Engenharia de Software Baseada em Busca - WESB 2012.
2. ARAÚJO, A. A.; PAIXAO, M. Machine learning for user modeling in an interactive genetic algorithm for the next release problem. In: Search Based Software Engineering - SSBSE'2014. Springer, 2014. p. 228–233.
3. BRUNO I.; PAIXÃO M.; SOUZA J. Uma Adaptação de Algoritmo Genético para o Problema do Próximo Release com Interdependências entre Requisitos. No IV Workshop de Engenharia de Software Baseada em Busca – WESB 2013.
4. ASSUNÇÃO W.; BARROS M.; COLANZI T.; NETO A.; PAIXÃO M.; SOUZA J.; VERGÍLIO S. Mapeamento da Comunidade Brasileira de SBSE. No IV Workshop de Engenharia de Software Baseada em Busca – WESB 2013.
5. ASSUNÇÃO W.; BARROS M.; COLANZI T.; NETO A.; PAIXÃO M.; SOUZA J.; VERGÍLIO S. A mapping study of the brazilian sbse community. Journal of Software Engineering Research and Development, Springer, v. 2, n. 1, p. 3, 2014.

RESUMO

O Problema do Próximo Release é uma importante tarefa no modelo de desenvolvimento de software iterativo e incremental, consistindo na seleção de um conjunto de requisitos para serem incluídos no próximo *release* do sistema. Atualmente, o ambiente de desenvolvimento de software é muito dinâmico e as incertezas relacionadas às variáveis de entrada desse problema devem ser levadas em consideração, principalmente quando abordagens baseadas em busca são utilizadas para resolução do problema. Sendo a *otimização robusta* uma linha de pesquisa que trata incertezas em problemas de otimização, esse trabalho apresenta uma formulação para o Problema do Próximo Release empregando técnicas da *otimização robusta*. Essa nova modelagem possibilita a geração de soluções robustas para o problema, ou seja, soluções que continuam válidas mesmo na presença de incertezas. Para medir e avaliar o “preço da robustez”, que é dado pela perda em qualidade da solução devido à robustez, um estudo empírico foi realizado, utilizando instâncias artificiais e reais do problema. Diferentes situações de planejamento do próximo *release* foram consideradas, incluindo diferentes números de requisitos, habilidades de estimativa e interdependências entre requisitos. Resultados da avaliação empírica mostram que a perda em qualidade da solução é relativamente pequena e que o comportamento do modelo proposto é estatisticamente o mesmo para todas as instâncias consideradas, o que evidencia a viabilidade da proposta.

Palavras-Chave: Problema do Próximo Release. Otimização Robusta. Engenharia de Software Baseada em Busca.

ABSTRACT

The Next Release Problem is a significant task in the iterative and incremental software development model, involving the selection of a set of requirements to be included in the next software release. Given the dynamic environment in which modern software development occurs, the uncertainties related to the input variables of this problem should be taken into account, especially when search based approaches are used to tackle the problem. Since the *robust optimization* is a modelling technique that handles uncertainties in generic optimization problems, this work presents a formulation to the next release problem considering the robust optimization framework. Such new formulation enables the production of robust solutions, i.e., solutions that are feasible even in the presence of noisy data. In order to measure the “price of robustness”, which is the loss in solution quality due to robustness, a large empirical evaluation was executed over synthetical and real-world instances. Several next release planning situations were considered, including different number of requirements, estimating skills and interdependencies between requirements. All empirical results are consistent to show that the penalization with regard to solution quality is relatively small. In addition, the proposed model’s behavior is statistically the same for all considered instances, which qualifies it to be applied even in large-scale real-world software projects.

Keywords: Next Release Problem. Robust Optimization. Search Based Software Engineering.

LISTA DE FIGURAS

Figura 1	Número de publicações em SBSE por ano entre 1976 e 2014. Fonte: (ZHANG, 2014)	22
Figura 2	Taxa de publicações em SBSE por país. Fonte: adaptado de (ZHANG, 2014).	23
Figura 3	Método de desenvolvimento em Cascata. Fonte: adaptado de (SOMMERVILLE, 2011).	25
Figura 4	Método de desenvolvimento Iterativo e Incremental. Fonte: adaptado de (SOMMERVILLE, 2011).	25
Figura 5	Incertezas em um sistema de otimização real. Fonte: adaptado de (BEYER; SENDHOFF, 2007).	30
Figura 6	Alguns tipos de Cruzamento para representação binária. Fonte: adaptado de (BURKE; KENDALL, 2005).	36
Figura 7	Mutação por Bit Flip. Fonte: (BURKE; KENDALL, 2005)	37
Figura 8	Valores de <i>fitness</i> encontrados pelo AG, TS e BA para algumas instâncias artificiais com variação de custo de 10% e sem relações de precedência. Fonte: Autor.	63
Figura 9	Fatores de redução encontrados pelo AG e TS para algumas instâncias artificiais com variação de custo de 10% e sem relações de precedência. Fonte: Autor.	65
Figura 10	Fatores de redução do AG para algumas instâncias artificiais com diferentes variações de custo e sem relações de precedência. Fonte: Autor.	69
Figura 11	Fatores de redução do AG para algumas instâncias artificiais com variações de custo de 10% e diferentes relações de precedência. Fonte: Autor.	71
Figura 12	Fatores de redução do AG para algumas instâncias reais com diferentes variações de custo. Fonte: Autor.	73

LISTA DE TABELAS

Tabela 1	Valores de <i>fitness</i> para as instâncias artificiais com variação de custo de 10% e sem precedência entre requisitos	62
Tabela 2	Fatores de redução para as instâncias artificiais com 10% de variação de custo e sem precedência entre requisitos	64
Tabela 3	Fatores de redução encontrados pelo AG para instâncias artificiais com diferentes variações de custo em sem relações de precedência	68
Tabela 4	Fatores de redução encontrados pelo AG para as instâncias artificiais com variação de custo de 10% e diferentes densidades de precedência	70
Tabela 5	Fatores de redução do AG para instâncias reais com diferentes variações de custo	73

LISTA DE ABREVIATURAS E SIGLAS

ACS	Ant Colony System.
AG	Algoritmo Genético.
BA	Busca Aleatória.
DAS	Dynamic Adaptive System.
ER	Engenharia de Requisitos.
ES	Engenharia de Software.
GRASP	Greedy Randomized Adaptive Search Procedure.
NRP	Next Release Problem.
PO	Pesquisa Operacional.
PPR	Problema da Priorização de Requisitos.
SBSE	Search Based Software Engineering.
SRL	Software Release Planning.
TS	Têmpera Simulada.

SUMÁRIO

1 INTRODUÇÃO	16
1.1 Motivação	16
1.2 Objetivos	19
1.3 Organização do Trabalho	19
2 FUNDAMENTAÇÃO TEÓRICA	21
2.1 Pesquisa Operacional	21
2.2 Engenharia de Software Baseada em Busca	22
2.3 Problemas da Engenharia de Requisitos Abordados pela SBSE	24
<i>2.3.1 Problema do Próximo Release</i>	24
<i>2.3.2 Problema do Planejamento de Releases</i>	27
<i>2.3.3 Problema da Priorização de Requisitos</i>	28
2.4 Otimização Robusta	29
2.5 Técnicas e Algoritmos de Busca	32
<i>2.5.1 Algoritmos Genéticos</i>	34
<i>2.5.2 Têmpera Simulada</i>	38
<i>2.5.3 Busca Aleatória</i>	40
2.6 Análise Estatística para Avaliação de Metaheurísticas em SBSE	41
2.7 Conclusões	42
3 TRABALHOS RELACIONADOS	43
3.1 Problema do Próximo Release	43
3.2 Tratamento de Incertezas em Problemas da Engenharia de Requisitos	45
3.3 Tratamento de Incertezas em outros Problemas da Engenharia de Software	47
3.4 Conclusões	48
4 PROPOSTA DE MODELAGEM ROBUSTA PARA O NRP	50
4.1 Modelo Matemático	50
4.2 Exemplo de Utilização	54
4.3 Conclusões	55
5 ESTUDO EMPÍRICO DA PROPOSTA	57

5.1 Configurações do Estudo Empírico	58
<i>5.1.1 Instâncias</i>	58
<i>5.1.2 Técnicas de Busca</i>	59
5.2 Resultados e Análises	61
<i>5.2.1 Preço da Robustez para diferentes níveis de robustez</i>	61
<i>5.2.2 Preço da Robustez para diferentes variações no custo dos requisitos</i>	67
<i>5.2.3 Preço da Robustez para instâncias com interdependências entre requisitos</i>	69
<i>5.2.4 Preço da Robustez para instâncias reais do problema</i>	72
5.3 Conclusões	74
6 CONSIDERAÇÕES FINAIS	76
6.1 Contribuições e Limitações da Pesquisa	76
6.2 Trabalhos Futuros	77
6.3 Conclusão	78
REFERÊNCIAS	79

1 INTRODUÇÃO

Nesse capítulo são apresentados os principais fatores que levaram ao desenvolvimento desse trabalho, assim como os objetivos a serem alcançados pelo mesmo. A estrutura do restante do trabalho é apresentada ao final.

1.1 Motivação

Em um processo de desenvolvimento de software iterativo e incremental, uma versão estável e executável do sistema entregue ao cliente é chamada de *release*. Apesar dos vários benefícios trazidos por esse modelo de desenvolvimento, como uma maior participação do cliente e facilidades na integração de mudanças, o mesmo incorpora uma maior complexidade ao gerenciamento do projeto. A escolha de quais requisitos serão incluídos no próximo *release* deve ser feita de forma a manter o cliente interessado e envolvido no projeto. O custo para desenvolver os requisitos selecionados para o próximo *release* deve respeitar um orçamento previamente definido. Além do mais, os requisitos normalmente apresentam interdependências entre si. Tais restrições também devem ser respeitadas. Na linha de pesquisa conhecida como Engenharia de Software Baseada em Busca, esse problema é conhecido como o Problema do Próximo Release, ou *Next Release Problem* (NRP).

A Engenharia de Software Baseada em Busca, ou *Search Based Software Engineering* (SBSE), sugere que os problemas da Engenharia de Software, devido à sua natureza complexa, podem ser modelados matematicamente como problemas de busca (HARMAN; JONES, 2001). Por complexos, podem ser identificados problemas que apresentam, dentre outras características: objetivos conflitantes, inconsistência nos dados e um grande número de possíveis soluções. A partir de uma modelagem matemática, torna-se possível uma resolução automática desses problemas com a aplicação de um algoritmo de otimização. Dentre as várias técnicas de busca existentes, os algoritmos evolucionários e as metaheurísticas destacam-se como as mais utilizadas em SBSE.

O NRP foi primeiramente modelado como um problema de otimização por Bagnall, Rayward-Smith e Whittle (2001). Nesse modelo, cada cliente tem uma certa importância para organização e solicita um subconjunto de requisitos para serem implementados no próximo *release*. O objetivo é selecionar um subconjunto de clientes para serem satisfeitos, de forma que a soma da importância de tais clientes seja máxima. Um cliente é considerado satisfeito quando todos os requisitos que ele solicitou são incluídos no *release*. Uma variação desse modelo foi proposta por Akker et al. (2005). Nesse outro modelo, ao invés do cliente, os próprios requisitos têm estimativas de importância para a empresa. Nesse caso, o objetivo é selecionar um subconjunto de requisitos onde a importância total seja maximizada.

Para aplicar alguma técnica de otimização na resolução do NRP, é necessário obter os valores de importância e custo de cada requisito. Como mostrado no estudo empírico por Cao e Ramesh (2008), empresas que utilizam um modelo iterativo e incremental de desenvolvimento de software, normalmente utilizam algum tipo de processo iterativo também na Engenharia de Requisitos (ER). Dentre a maioria das práticas iterativas de ER citadas no estudo, um ponto em comum entre elas é o fato do cliente indicar o valor de importância dos requisitos. Contudo, a partir da evolução e da utilização do sistema por parte do cliente, requisitos previamente considerados importantes podem perder prioridade, assim como requisitos não tão importantes podem assumir um caráter prioritário. Dessa forma, os valores de importância dos requisitos são na verdade estimativas que podem mudar ao longo do desenvolvimento do sistema.

Com relação ao custo dos requisitos em abordagens iterativas de ER, também é comum a utilização de estimativas (BOEHM; ABTS; CHULANI, 2000). Apesar de existirem várias técnicas para estimação de custo de software, a maioria segue o mesmo princípio. Um grupo de especialistas, normalmente a equipe de desenvolvimento, analisa cada requisito e faz uma projeção do trabalho necessário para desenvolvimento daquela funcionalidade. Tais estimativas são principalmente baseadas em experiências pessoais e conhecimentos obtidos em projetos passados (BOEHM; ABTS; CHULANI, 2000).

Tanto as estimativas de importância como as de custo dos requisitos podem ser consideravelmente difíceis de se elaborar devido ao ambiente dinâmico no qual o desenvolvimento de software está inserido. No trabalho por Harker, Eason e Dobson (1993), requisitos de software são classificados em seis diferentes tipos. Cada tipo ainda é especificado como ‘dinâmico’ ou ‘estático’. No primeiro caso algumas características do requisito podem mudar ao longo do desenvolvimento, e no segundo caso as características do requisito não mudam com o tempo. Dentre os seis tipos diagnosticados, cinco são identificados como ‘dinâmicos’ e apenas um como ‘estático’, o que ressalta a característica evolutiva dos requisitos. Nesse contexto, importância e custo dos requisitos estão entre os aspectos que podem mudar durante o desenvolvimento do *release*. De fato, o alto grau de incerteza relacionado às variáveis do NRP resulta em um contexto relativamente complicado, conforme ressaltado por Zhang, Finkelstein e Harman (2008):

“Problemas da Engenharia de Software são tipicamente ‘confusos’, onde a informação disponível é frequentemente incompleta, às vezes vaga e quase sempre sujeita a um alto grau de mudanças (inclusive mudanças imprevistas). Requisitos mudam constantemente e pequenas mudanças nos estágios iniciais normalmente levam a grandes mudanças nas soluções, tornando-as mais complexas e fazendo com que os resultados iniciais tornem-se potencialmente frágeis.”

Com relação a possíveis problemas ocasionados por erros nas estimativas de custo

dos requisitos, a análise de sensibilidade realizada por Harman et al. (2009) mostra o impacto que uma estimativa de custo incorreta pode ter na solução final do NRP. Diferentes situações foram simuladas, considerando tanto requisitos de alto como de baixo custo, com grandes ou pequenos erros de estimativa. Na maioria dos casos, quanto maior o custo dos requisitos, independente se o erro na estimativa é grande ou pequeno, maior será o impacto no resultado final. De forma similar, quanto maior for erro na estimativa, maior será o impacto, independente do requisito ter um alto ou um baixo custo de desenvolvimento.

Como pode ser visto, algumas abordagens utilizadas para estimar os valores de importância e custo dos requisitos no desenvolvimento de software incremental podem inserir um certo grau de incerteza nas variáveis do NRP. Tais incertezas, se não tratadas cuidadosamente, podem causar um grande impacto no conjunto de requisitos selecionados para o próximo *release*. Dessa forma, ao solucionar o NRP utilizando técnicas de otimização, as incertezas relacionadas com a importância e custo dos requisitos devem ser consideradas.

Admitindo que alguns aspectos do problema são incertos, a otimização robusta apresenta-se como um framework da pesquisa operacional para tratar incertezas em problemas de otimização (BEYER; SENDHOFF, 2007). Tal tratamento consiste em basicamente identificar cada incerteza do problema e quantificá-la da forma mais adequada. Ao se utilizar a otimização robusta para tratar uma determinada incerteza, podem ser definidos tanto o *nível de robustez* como a *variação da incerteza*.

Tomando as incertezas das estimativas de custo como exemplo, o *nível de robustez* está relacionado com a maior quantidade de requisitos que serão considerados como erroneamente estimados. Neste caso, o *nível de robustez* age, dentre o conjunto de todas as estimativas de custo, como um indicador da quantidade máxima de requisitos que estão estimados incorretamente. A *variação da incerteza* de custo, por sua vez, representa o quanto o custo de cada requisito pode variar com relação à estimativa original. Essa variação matemática pode ser tanto positiva, quando o custo real é maior que a estimativa original, como negativa, quando o custo do requisito se mostra na verdade menor do que previamente estimado. Baseado no *nível de robustez* e na *variação da incerteza*, a otimização robusta projeta modelos matemáticos que buscam por soluções robustas. Uma solução robusta é aquela que, mesmo utilizando dados possivelmente incertos, continua respeitando todas as restrições do problema.

Dessa forma, a otimização robusta pode ser utilizada para se desenvolver um modelo matemático para o NRP que considere as incertezas associadas a esse problema. Então, tal modelo robusto pode tratar as incertezas de importância e custo dos requisitos, permitindo a geração de soluções robustas para o NRP.

Ao se garantir robustez a um problema de otimização, uma certa perda em qualidade da solução é inevitável. Tal perda está altamente relacionada ao *nível de robustez* e a *variação da incerteza*. A medida de perda de qualidade da solução robusta quando comparada com a

solução não-robusta é conhecida na literatura como o “preço da robustez” (BERTSIMAS; SIM, 2004). Com relação ao NRP, tal valor deve ser medido e avaliado cuidadosamente, pois é a partir dele que o engenheiro de requisitos deve decidir o quanto será possível perder em qualidade do *release* para aumentar a confiança do mesmo com relação às incertezas.

1.2 Objetivos

Este trabalho tem como objetivo principal apresentar uma nova formulação para o Problema do Próximo Release utilizando a otimização robusta para tratar as incertezas relacionadas à importância e custo dos requisitos. Ao considerar as incertezas desse problema, o novo modelo torna-se mais próximo de uma situação real de planejamento de *releases*, possibilitando uma maior aceitação e utilização de SBSE em projetos reais de desenvolvimento de software.

De forma complementar ao objetivo geral do trabalho, foram definidos os seguintes objetivos específicos:

- A partir das diversas estratégias de quantificação de incertezas presentes na otimização robusta, identificar as mais adequadas para mensurar as incertezas de importância e custo dos requisitos.
- Adaptar e implementar diferentes técnicas de busca para resolução do modelo robusto para o NRP.
- Avaliar o “preço da robustez” da nova modelagem proposta. Em outras palavras, medir o quanto é perdido em qualidade da solução para se garantir robustez. Para possibilitar uma melhor análise desse importante objetivo específico, as seguintes análises serão desenvolvidas separadamente:
 - Análise do “preço da robustez” quando se usa diferentes níveis de robustez.
 - Análise do “preço da robustez” quando se usa diferentes variações no custo dos requisitos.
 - Análise do “preço da robustez” na presença de interdependências entre requisitos.
 - Análise do “preço da robustez” quando o modelo é aplicado a instâncias reais do problema.

1.3 Organização do Trabalho

O conteúdo deste trabalho está organizado em seis capítulos, incluindo a presente introdução. Cada capítulo é sucintamente apresentado a seguir:

Capítulo 2 - Fundamentação Teórica: Discorre sobre os conceitos de Pesquisa Operacional e da linha de pesquisa de Engenharia de Software Baseada em Busca, apresentando um pequeno histórico e seus conceitos básicos. Os problemas da Engenharia de Requisitos tratados pela SBSE também são apresentados, dando uma ênfase maior ao Problema do Próximo Release, por ser o objeto de estudo deste trabalho. A otimização robusta, como técnica de modelagem utilizada, também tem suas características exibidas. Finalmente, os algoritmos utilizados no estudo empírico são apresentados, bem como as ferramentas estatísticas usadas na validação dos resultados.

Capítulo 3 - Trabalhos Relacionados: Apresenta os principais trabalhos relacionados ao Problema do Próximo Release e tratamento de incertezas na resolução de problemas da Engenharia de Software a partir de técnicas de busca. São destacadas as principais características de tais trabalhos, bem como as principais diferenças relacionadas à presente pesquisa.

Capítulo 4 - Proposta de Modelagem Robusta para o NRP: Descreve em detalhes a modelagem robusta para o Problema do Próximo Release proposta nesse trabalho. A partir do modelo tradicional proposto por Akker et al. (2005), a construção do novo modelo é mostrada em etapas. Um exemplo do funcionamento do modelo é utilizado para facilitar o entendimento do mesmo.

Capítulo 5 - Estudo Empírico da Proposta: Relata o estudo empírico realizado para validação e avaliação da modelagem proposta. Apresenta primeiramente as instâncias utilizados no estudo, bem como os detalhes da adaptação de cada algoritmo de busca aplicado. Os respectivos resultados são analisados empirica e estatisticamente, chegando a conclusões acerca do comportamento do modelo proposto.

Capítulo 6 - Considerações Finais: Relaciona as principais contribuições e conclusões desse trabalho, bem como suas limitações. Oportunidades para trabalhos futuros também são apontadas.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo fornece uma fundamentação teórica sobre alguns dos tópicos apresentados neste trabalho, facilitando o entendimento do modelo robusto proposto e do estudo empírico realizado para validação e avaliação do mesmo. Primeiramente, os conceitos de Pesquisa Operacional e otimização matemática são brevemente introduzidos, promovendo uma melhor explicação da linha de pesquisa da Engenharia de Software Baseada em Busca. Posteriormente, os problemas relacionados a requisitos que são tratados pela SBSE são apresentados, com um foco maior no Problema do Próximo Release. Os conceitos básicos da Otimização Robusta são então mostrados, bem como as técnicas de busca utilizadas no estudo empírico. Finalmente, as ferramentas estatísticas utilizadas para avaliação dos resultados do estudo empírico também são mostradas.

2.1 Pesquisa Operacional

A Pesquisa Operacional (PO) é um ramo da computação e da matemática aplicada que procura solucionar de forma ótima, ou quase ótima, problemas complexos do mundo real (ANDRADE, 1998). Entende-se por solução ótima a melhor solução possível, ou seja, uma solução otimizada. A partir de modelos matemáticos do problema que está sendo abordado, essa solução ótima é encontrada com a aplicação de algoritmos de otimização, também chamados de algoritmos ou técnicas de busca. Um modelo matemático de otimização apresenta a seguinte forma geral:

$$\begin{aligned} &\text{minimizar } f(x) \\ &\text{sujeito a: } g_j(x) \geq 0, \quad j = \{1, 2, \dots, J\} \\ &\quad \quad \quad h_k(x) = 0, \quad k = \{1, 2, \dots, K\} \\ &\quad \quad \quad x \in \Omega \end{aligned}$$

Onde $f(x)$ é a função a ser otimizada, sendo possível tanto uma minimização (como no modelo acima) como uma maximização de $f(x)$. As inequações $g_j(x) \geq 0$ e as equações $h_k(x) = 0$ representam as restrições do problema que devem ser respeitadas ao otimizar $f(x)$. O conjunto Ω é formado por todos os possíveis valores de x que podem ser usados para valorar $f(x)$. Uma solução viável para o problema é aquela que respeita todas as restrições.

Vários ramos da engenharia moderna vêm utilizando a PO para solução de problemas. Notadamente podem ser vistos trabalhos nas áreas de engenharia elétrica (BIANCHI; BOLOGNANI, 1998) (JINTAO; LAI; YIHAN, 1995), engenharia química (JUDSON, 2007), engenharia civil (ABDULLAH; RICHARDSON; HANIF, 2001) dentre outras. A Engenharia de Software (ES) como é conhecida hoje, sendo uma disciplina de engenharia, pode e já está incorporando a PO no seu escopo de pesquisa, como pode ser visto na seção a seguir.

2.2 Engenharia de Software Baseada em Busca

A Engenharia de Software Baseada em Busca, ou *Search Based Software Engineering* (SBSE), é o termo criado por Harman e Jones (2001) para denominar a aplicação de técnicas de busca em ES. Trabalhos anteriores a este artigo já faziam uso da PO em ES, principalmente na área de testes, mas a partir desse trabalho é que a comunidade de ES realmente virou sua atenção para os algoritmos de otimização. O número de publicações em SBSE ao longo dos anos pode ser visto na Figura 1. A quantidade de publicações a cada ano foram retiradas do SEBASE (ZHANG, 2014), atualmente o maior repositório de artigos de SBSE.

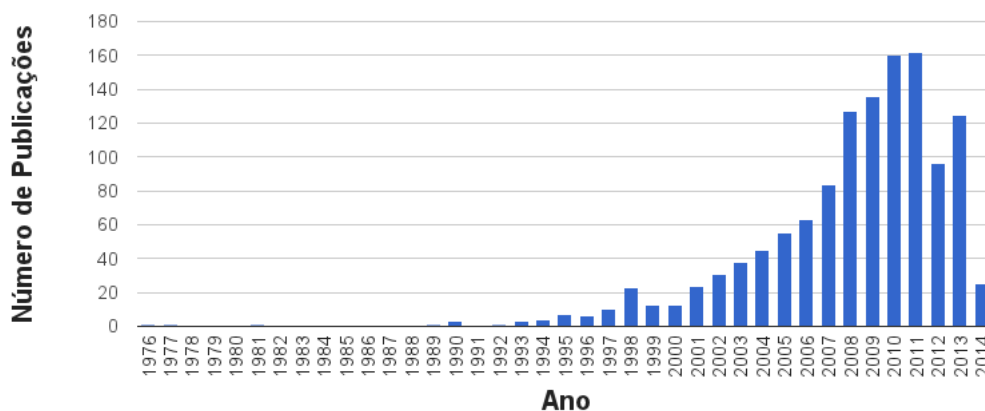


Figura 1 – Número de publicações em SBSE por ano entre 1976 e 2014. Fonte: (ZHANG, 2014)

Como dito anteriormente, o crescimento em publicações depois de 2001 é visível. Um grande salto ocorreu no ano de 2008, onde desse ano até 2013, a média de publicações anuais se manteve acima de 130 artigos. É válido destacar que em relação a 2014 foram contabilizados artigos publicados até Maio.

Uma divisão da quantidade de publicações entre países pode ser vista na Figura 2. Como pode ser visto, o Reino Unido e EUA são responsáveis por grande parte das publicações em SBSE a nível mundial, representando mais de 40% do total de artigos. O Brasil encontra-se atualmente contribuindo com 6% de todas as publicações de SBSE.

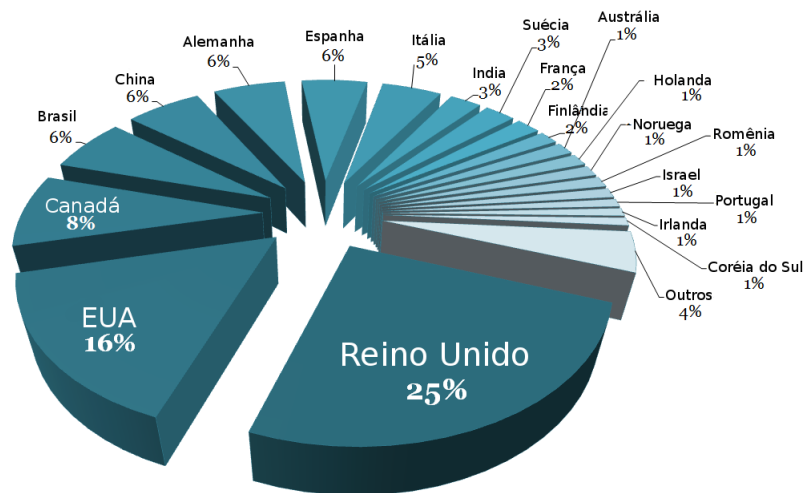


Figura 2 – Taxa de publicações em SBSE por país. Fonte: adaptado de (ZHANG, 2014).

Para aplicar uma técnica de otimização a um problema de ES, se faz necessária a adaptação, ou modelagem, do problema. Essa modelagem é constituída por basicamente dois pontos: representação da solução e função objetivo (HARMAN; JONES, 2001). O primeiro diz respeito a como uma possível solução para o problema será representada. Normalmente essa representação é feita através de vetores ou matrizes de números inteiros ou binários, sendo possível ainda a utilização de estruturas de dados mais complexas. A função objetivo, ou função de *fitness*, é responsável por medir a qualidade de uma possível solução. Basicamente atribui um valor numérico para determinada solução, fazendo com que várias soluções candidatas possam ser comparadas entre si. A modelagem do problema é um dos fatores mais importante na SBSE, sendo considerado o mais importante por vários autores (HARMAN, 2007). Segundo Harman e Clark (2004), para se conseguir bons resultados com SBSE, a modelagem do problema deve apresentar as seguintes características:

- **Grande Espaço de Busca:** A modelagem deve ser capaz de representar e diferenciar muitas, idealmente todas, as possíveis soluções do problema. Assim, as técnicas de busca são capazes de explorar o espaço de busca exaustivamente.
- **Baixa Complexidade Computacional:** Devido ao grande espaço de busca, muitas soluções devem ser avaliadas até que se encontrem as soluções ótimas e sub-ótimas. Isso resulta em milhares de avaliações de *fitness*, ou seja, várias execuções da função objetivo. Dessa forma, a complexidade da função objetivo tem uma grande influência na complexidade do processo de busca como um todo.
- **Continuidade da Função Objetivo:** A função objetivo não precisa ser necessariamente contínua para servir como um bom guia para a busca, mas um alto grau de descontinui-

dade na função pode resultar em uma má diferenciação entre soluções, dificultando o processo de busca. Quanto mais contínua a função objetivo, possivelmente serão encontrados melhores resultados.

- **Desconhecimento das Soluções Ótimas:** As possíveis soluções ótimas para o problema devem ser desconhecidas. Caso contrário uma abordagem baseada em busca não seria necessária.

Com a SBSE tomando forma e se concretizando como uma linha de pesquisa dentro da ES, alguns problemas já são considerados “clássicos” da área. Dentre os quais podem ser citados a seleção e priorização de casos de teste (ELBAUM; MALISHEVSKY; ROTHERMEL, 2002) (ROTHERMEL et al., 2001) e a otimização de estimativas de custo (DOLADO, 2000) (KIRSOPP; SHEPPERD; HART, 2002). Alguns dos problemas relacionados à Engenharia de Requisitos também já foram abordados pela SBSE. Tais problemas são mostrados na seção a seguir.

2.3 Problemas da Engenharia de Requisitos Abordados pela SBSE

Nesta seção serão apresentados com mais detalhes alguns problemas relacionados à Engenharia de Requisitos que já foram resolvidos utilizando técnicas baseadas em busca. Por ser objeto de estudo desse trabalho, o Problema do Próximo Release será abordado com mais detalhes.

2.3.1 Problema do Próximo Release

A Engenharia de Software é uma disciplina de engenharia relacionada a todos aspectos que envolvem a produção de um software, desde as etapas iniciais de especificação do sistema até a etapa de manutenção do produto já em utilização (SOMMERVILLE, 2011). Para tanto, foram desenvolvidas várias técnicas e metodologias ao longo dos anos. Um dos primeiros processos de desenvolvimento de software utilizado foi o método cascata, mostrado na Figura 3. Como pode ser observado, o sistema é construído em um único ciclo, sendo o produto entregue como um todo ao final do desenvolvimento. O contato do cliente com o software acontece somente na sua versão final. A maior deficiência desse modelo é exatamente a baixa participação do cliente durante o processo de desenvolvimento, causando problemas como: insatisfação do mesmo com relação a prazo e funcionalidades mal entendidas pelos desenvolvedores. Outro problema que merece ser mencionado é a dificuldade de incorporação de mudanças após o início do desenvolvimento do sistema.

O ciclo de vida iterativo e incremental surge em resposta às deficiências frequentemente apontadas pelo modelo em cascata. Apesar de atualmente ter se tornado popular por

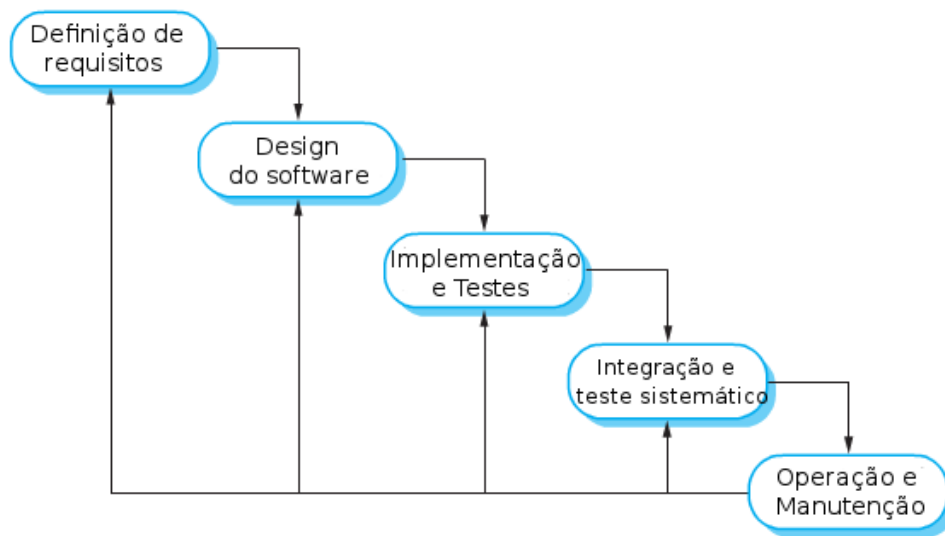


Figura 3 – Método de desenvolvimento em Cascata. Fonte: adaptado de (SOMMERVILLE, 2011).

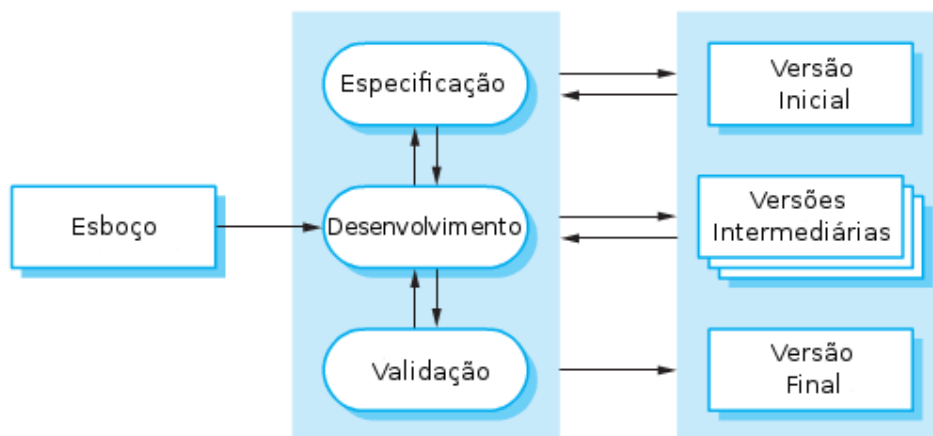


Figura 4 – Método de desenvolvimento Iterativo e Incremental. Fonte: adaptado de (SOMMERVILLE, 2011).

conta das metodologias ágeis, esse modelo de desenvolvimento já é utilizado desde a década de 70 (LARMAN; BASILI, 2003). Consiste em dividir o desenvolvimento de software em vários espaços de tempo, denominados de ciclos, ou iterações. Ao final de cada iteração, um subproduto executável do sistema é entregue ao cliente. É chamado de iterativo pois esses ciclos se repetem até o sistema estar completamente terminado. Esse ciclo de vida se encaixa de certa forma no modelo de desenvolvimento evolucionário, onde são adicionadas novas funcionalidades (módulos) ao sistema de forma contínua. O modelo iterativo e incremental é ilustrado na Figura 4.

O desenvolvimento de software incremental faz com que o cliente coopere com o processo de desenvolvimento. Essa participação ativa do usuário facilita o entendimento das

funcionalidades por parte dos desenvolvedores. A cada final de ciclo uma versão do sistema é entregue, aumentando a satisfação do cliente com relação a prazo. Essa versão cumpre a função de um protótipo, possibilitando que o cliente forneça *feedbacks* frequentes à equipe de desenvolvimento e conseqüentemente reduz a possibilidade de insatisfação em relação às funcionalidades desenvolvidas até então. Outra vantagem que deve ser citada diz respeito aos riscos do projeto. Caso o mesmo seja abortado, o cliente ainda poderá utilizar as funcionalidades implementadas até o momento da interrupção.

O método iterativo traz muitas vantagens para o desenvolvimento de grandes sistemas de software, mas também incorpora uma maior complexidade no gerenciamento das atividades por parte dos desenvolvedores e líderes de projetos. Ao final de cada iteração, as novas funcionalidades implementadas devem ser acopladas ao subproduto que o cliente está utilizando, gerando um novo subproduto. Essa questão é comumente chamada de integração. Testes de integração são muito abordados pela comunidade de ES, inclusive de SBSE, como pode ser visto em (BRIAND; FENG; LABICHE, 2002) e (ASSUNÇÃO et al., 2011). Podem ser apresentados ainda outros problemas provenientes do método incremental, mas o maior deles é a seleção de quais funcionalidades serão desenvolvidas a cada iteração.

A versão intermediária do produto ao final de cada iteração é chamada de *release*. Como o sistema será entregue ao longo de vários *releases*, torna-se claro que, no início de cada ciclo, deve ser decidido o que será incluído no *release*. A seleção de quais requisitos serão entregues no próximo *release* do sistema caracteriza o Problema do Próximo Release, ou NRP. Tal problema foi pela primeira vez abordado com técnicas de busca por Bagnall, Rayward-Smith e Whittley (2001).

A seleção dos requisitos para o próximo *release* do sistema é uma tarefa relativamente complicada, pois existem muitos aspectos a serem considerados. De forma direta, o principal objetivo de um *release* é satisfazer o cliente ao máximo, implementando os requisitos que o mesmo considera como mais importantes. No cenário em que o cliente é uma única pessoa ou somente uma organização, a tarefa é simples. Mas na maioria das vezes, mesmo dentro de uma mesma instituição, várias pessoas vão utilizar o sistema de forma diferente. Nesse panorama, existem vários clientes com prioridades de requisitos diferentes, em que um *release* vai atingir níveis de satisfação diferentes para cada cliente. Além do mais, os vários clientes ainda podem apresentar importâncias distintas para a empresa que está desenvolvendo o software.

Além de lidar com objetivos conflitantes no momento da seleção dos requisitos para o próximo *release*, o gerente do projeto e a equipe de desenvolvimento ainda têm de atentar para uma grande restrição: o orçamento disponível. Cada requisito apresenta um certo custo de desenvolvimento, bem como também existe um orçamento máximo para o *release*. Dessa forma, devem ser selecionados os requisitos que mais se adequam aos objetivos de forma a não ultrapassar o orçamento do *release*.

Outro fator importante a ser contemplado no planejamento do próximo *release* são os relacionamentos existentes entre requisitos. Requisitos normalmente se relacionam entre si de diferentes formas, sendo esses relacionamentos comumente chamados de interdependências (CARLSHAMRE et al., 2001). As interdependências entre requisitos podem ser classificadas em dois grupos: *interdependências funcionais* e *interdependências de valor* (SAGRADO; AGUILA; ORELLANA, 2011). As primeiras determinam relações onde a implementação de certo requisito depende diretamente da implementação de outro. Nas últimas, a implementação de um requisito pode influenciar nas características de outro requisito, como aumentar sua importância ou diminuir seu custo de desenvolvimento.

Apesar da complexidade dessa atividade, muitas empresas ainda realizam o planejamento do próximo *release* de forma *ad hoc* ou artesanal (RUHE; SALIU, 2005). A comparação entre os requisitos são baseadas totalmente em experiência e intuição dos desenvolvedores, com análises subjetivas da relevância de cada requisito bem como da importância de cada cliente. Como pode ser visto em (KARLSSON; WOHLIN; REGNELL, 1998), existem várias metodologias para a seleção e priorização de requisitos, mas a grande maioria delas é dependente de análises subjetivas humanas.

Torna-se visível que é necessária uma abordagem mais automatizada para o problema, uma técnica que diminua a necessidade de análises subjetivas e erros humanos. É relevante proporcionar aos engenheiros de software a possibilidade de se preocuparem somente com os objetivos que devem ser alcançados no próximo *release*, sem ocupar seu tempo com cálculos de orçamento e satisfação. Nesse contexto, a comunidade de SBSE, incluindo este trabalho, tem voltado suas atenções para solucionar o NRP através de modelagens matemáticas e algoritmos de busca.

2.3.2 Problema do Planejamento de Releases

Outro problema relacionado à escolha de quais requisitos serão implementados nas versões intermediárias do sistema é o Problema do Planejamento de Releases. Trabalhos na literatura se referem a esse problema simplesmente como *Software Release Planning* (SRL) (RUHE, 2005). O NRP, como dito anteriormente, tem como objetivo planejar somente o próximo *release* do sistema, sendo necessário repetir o mesmo processo para todos os ciclos de desenvolvimento. Diferentemente, o SRL tem como objetivo planejar todos os *releases* do sistema. Dado o conjunto de todos os requisitos a serem implementados no sistema, todos os *releases* são planejados de uma única vez, alocando os requisitos a cada *release*. As dificuldades citadas anteriormente para o NRP continuam para o SRL, como as restrições de orçamento para os *releases*, clientes com objetivos possivelmente conflitantes e interdependências entre requisitos.

No NRP, cada *release* é planejado para maximizar a satisfação do cliente. Assim, ao longo do desenvolvimento, é possível que as últimas entregas do sistema sejam compostas por funcionalidades não tão importantes, diminuindo o interesse do cliente com o projeto. Como no SRL os *releases* são todos planejados ao mesmo tempo, é possível organizar os requisitos mais importantes da forma mais adequada ao projeto. Pode-se, por exemplo, balancear as importâncias dos *releases*, mantendo um nível de satisfação constante do cliente ao longo do projeto.

A grande limitação do SRL está relacionada com possíveis adições ou remoções de requisitos ao longo do desenvolvimento do sistema. Mudanças no conjunto de requisitos resultam em um replanejamento dos *releases*. Como os mesmos foram planejados de forma ótima ou quase ótima, uma realocação manual de requisitos pode inviabilizar o planejamento anterior, enquanto uma realocação automática caracteriza um outro problema de otimização. O NRP planeja o novo *release* a cada ciclo, portanto, ao se adicionar um novo requisito ao projeto, este entrará no processo de seleção já no próximo ciclo. Dessa forma, percebe-se que a decisão de se utilizar o NRP ou o SRL no planejamento dos *releases* do sistema deve ser feita levando em consideração principalmente a possibilidade de mudanças no conjunto de requisitos ao longo do projeto.

Mais detalhes sobre o Problema do Planejamento de Releases podem ser encontrados em (GREER; RUHE, 2004), (RUHE; SALIU, 2005), (COLARES et al., 2009) e (SOUZA et al., 2011).

2.3.3 Problema da Priorização de Requisitos

Tanto o NRP como o SRL estão relacionados com o planejamento dos *releases* de um projeto de software quando este segue o ciclo de vida iterativo e incremental. Vários fatores são levados em consideração no momento do planejamento, como o orçamento disponível para os *releases*, interdependências entre requisitos, entre outros. Do ponto de vista de Engenharia de Software, tais modelagens são atualmente as que mais se aproximam de um ambiente real de planejamento de *releases*, o que também se reflete na complexidade computacional desses modelos matemáticos. O NRP e o SRL se assemelham aos clássicos Problema da Mochila e Problema das Múltiplas Mochilas, respectivamente. Ambos são problemas NP-completos, ou seja, ainda não existe um algoritmo conhecido que resolva esses problemas em tempo polinomial.

Apesar da reconhecida qualidade do NRP e do SRL, alguns trabalhos tratam o planejamento de requisitos de forma um tanto mais simples, através do Problema da Priorização de Requisitos (PPR). Essa modelagem consiste basicamente em, dado o conjunto de requisitos a serem implementados, definir uma ordem de implementação. Diferentes trabalhos levam em

consideração diferentes critérios para essa priorização, como importância e riscos dos requisitos. Alguns ainda consideram alguns tipos de interdependências entre requisitos. De forma geral, os modelos são muito simples e não refletem exatamente um contexto real de priorização de requisitos, sendo muitas vezes resumidos, no contexto de otimização, a simples problemas de ordenação.

De qualquer forma, o PPR ainda apresenta uma vantagem importante. Enquanto o NRP e o SRL estão diretamente ligados ao modelo de desenvolvimento iterativo e incremental, essa ordem de implementação gerada pelo PPR é independente do modelo de desenvolvimento utilizado. Mais detalhes sobre o Problema da Priorização de Requisitos podem ser encontrados em (LIMA et al., 2011), (BRASIL et al., 2011) e (PAIXAO et al., 2012).

2.4 Otimização Robusta

Como foi apresentado anteriormente, a PO procura solucionar problemas do mundo real de forma ótima. Essa solução é encontrada a partir de uma modelagem matemática do problema juntamente com um algoritmo de resolução. A maioria das modelagens matemáticas de problemas reais parte do pressuposto que os dados a serem processados são determinísticos e livres de ruído (MULVEY; VANDERBEI; ZENIOS, 1995). É de conhecimento geral que uma modelagem matemática ideal, ou seja, que consiga mapear e modelar o ambiente ou o sistema de forma perfeita, é impossível de ser desenvolvida. Os dados de entrada, sejam obtidos a partir de simulações ou de estimativas, estão sempre sujeitos a falhas. Dessa forma, é chamada de incerteza ou imprecisão qualquer aspecto do sistema que possa assumir um certo valor diferente do esperado ou estimado.

A primeira proposta de tratamento de incertezas em problemas de otimização é mostrada em (TAGUCHI, 1986), onde é introduzido o que o autor chama de “engenharia de qualidade”. Ele defende que é necessário atingir um certo nível de “qualidade” na modelagem e na resolução de problemas de otimização levando em consideração as várias possíveis fontes de erros dos problemas. Algumas de suas teorias e ideias foram questionadas, ou ainda desconsideradas, após alguns anos, mas ele é de fato reconhecido como o criador desse conceito referente a análise de incertezas em problemas de otimização.

Apesar dessas primeiras abordagens por Taguchi (1986), a otimização robusta ganhou mais visibilidade após os trabalhos por Mulvey, Vanderbei e Zenios (1995) e Bai, Carpenter e Mulvey (1997). Sendo o primeiro responsável por apresentar os conceitos fundamentais do que se conhece atualmente como Otimização Robusta e o segundo responsável por chamar a atenção para a importância do tratamento de incertezas. Desde então, a Otimização Robusta já foi aplicada com sucesso em diversas disciplinas de engenharia incluindo, mas não se limitando a, produção (LEUNG et al., 2007), aeronáutica (DU; WANG; CHEN, 2000), eletrônica (MAL-

COLM; ZENIOS, 1994), mecânica (LI; AZARM, 2008), química (WANG; RONG, 2009) e metalúrgica (DULIKRAVICH; EGOROV-YEGOROV, 2005).

A Otimização Robusta pode ser considerada um *framework*, pois a aplicação da mesma a um certo problema de otimização consiste em uma série de passos sequenciais e bem definidos, onde existem diversas técnicas disponíveis para realização de cada passo. Assim, o *framework* da Otimização Robusta consiste em basicamente três passos:

1. Identificar e quantificar as incertezas do problema;
2. Desenvolver um modelo matemático de otimização capaz de gerar soluções robustas para o problema;
3. Aplicar um determinado algoritmo de otimização para resolução do modelo robusto desenvolvido;

Com relação ao primeiro passo, os diferentes tipos de incerteza em um sistema de otimização real são mostrados na Figura 5 e explicados a seguir (BEYER; SENDHOFF, 2007).

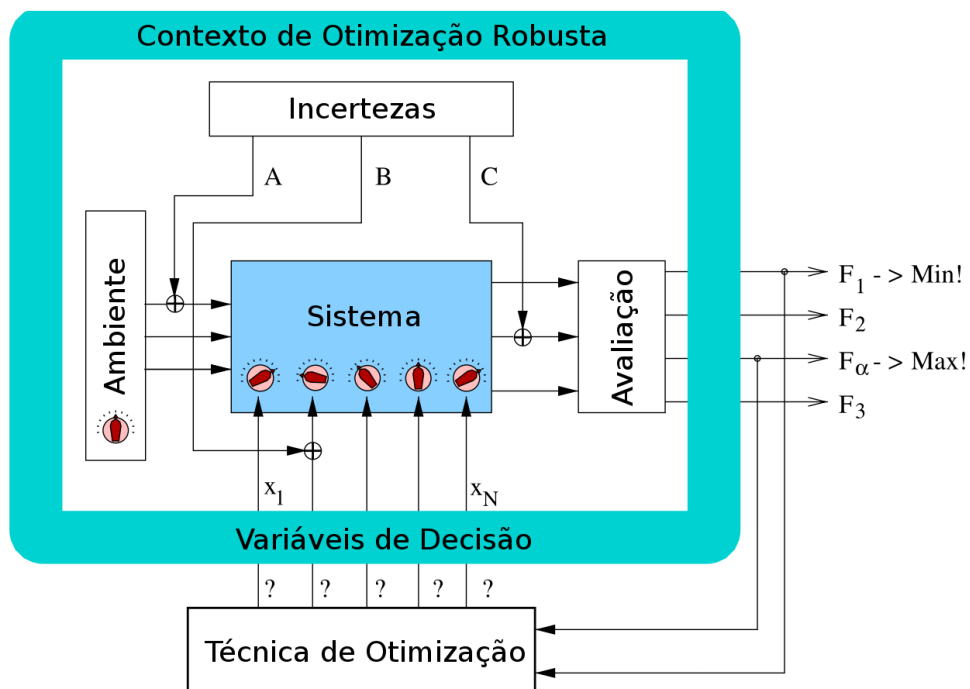


Figura 5 – Incertezas em um sistema de otimização real. Fonte: adaptado de (BEYER; SENDHOFF, 2007).

- (A) **Mudanças ambientais e de condições de operação:** os modelos matemáticos são desenvolvidos a partir de uma certa configuração fixa das características do ambiente de operação, sendo que um ambiente real muda constantemente.

- (B) **Imprecisão nas variáveis de decisão:** as variáveis de decisão $\{x_1 \dots x_N\}$ atuam no sistema a ser otimizado a partir de atuadores, como motores, reguladores etc. Equipamentos de alta precisão apresentam um custo relativamente alto, sendo que na maioria das vezes se utiliza maquinário com um certo grau de erro.
- (C) **Incerteza de resultados:** relacionada com o uso de simulações. Quando se aplica a solução encontrada pelo algoritmo de otimização no ambiente real de operação, os resultados reais podem ser diferentes dos resultados simulados.
- (D) **Incerteza no cumprimento das restrições:** relacionada às restrições do problema. É uma incerteza que surge quando as demais incertezas (A), (B) e (C) são consideradas, por isso não é mostrada na Figura 5 acima. Quando as restrições são definidas a partir de estimativas e simulações, é necessário garantir que a solução gerada continue respeitando todas essas restrições do problema.

Os tipos de incertezas mostrados acima também podem ser chamados de *pontos de falha* (ROY, 2010), representando as fontes de incertezas do problema de otimização. Após a identificação das incertezas pertencentes ao problema, as mesmas devem ser quantificadas e inseridas na formulação matemática. Os tipos de incerteza apresentados acima (A-D), podem ser quantificados de forma discreta ou contínua (AISSI; BAZGAN; VANDERPOOTEN, 2009), seguindo uma das seguintes estratégias (BEYER; SENDHOFF, 2007):

- (1) **Determinística:** define domínios específicos em que os valores incertos podem variar. Normalmente representados por conjuntos finitos.
- (2) **Probabilística:** define medidas de probabilidade de uma incerteza assumir um certo valor. A distribuição de probabilidade que a incerteza segue pode ser tanto definida manualmente como a partir de distribuições previamente conhecidas.
- (3) **Possibilística:** utiliza lógica *fuzzy* e valores nominais para lidar com incertezas subjetivas, indicando a possibilidade de uma incerteza assumir um certo valor.

A escolha de um conjunto específico de estratégias para quantificar cada uma das incertezas do problema caracteriza o que é chamado na literatura de *versão* do modelo robusto (ROY, 2008).

Após a identificação e quantificação das incertezas do problema, o próximo passo do *framework* da Otimização Robusta é o desenvolvimento do modelo de otimização que considere as incertezas e gere soluções robustas. Um modelo de otimização robusto é similar a um modelo de otimização ‘tradicional’, como o mostrado na seção 2.1, sendo composto por um conjunto de funções objetivo e um conjunto de restrições. Diferentemente, um modelo robusto deve ser desenvolvido considerando as incertezas do problema. Dessa maneira, considere

S como o conjunto de todos os valores que as incertezas do problema podem assumir. Um modelo de otimização robusto apresenta a seguinte forma geral:

$$\begin{array}{ll}
 \text{minimizar } f_s(x) & \forall s \in S \\
 \text{sujeito a: } g_{j,s}(x) \geq 0, & j = \{1, 2, \dots, J\}, \quad \forall s \in S \\
 & h_{k,s}(x) = 0, \quad k = \{1, 2, \dots, K\}, \quad \forall s \in S \\
 & x \in \Omega
 \end{array}$$

De forma análoga ao modelo tradicional, além de uma minimização, também é possível uma maximização de $f(x)$. Percebe-se que os possíveis valores para as incertezas são considerados tanto na função objetivo como nas restrições. Dessa maneira, uma solução ótima x deve ser ótima para todos os possíveis valores de S e ainda respeitar todas as restrições para todos os possíveis valores de S .

A partir do desenvolvimento do modelo robusto, basta seguir para o último passo do *framework* da Otimização Robusta, que é a aplicação de uma técnica de busca para resolução do modelo robusto. Essa parte pode ser considerada a mais simples, no sentido de que não há mais nada de ‘novo’. Qualquer técnica de busca ‘tradicional’ pode ser aplicada, incluindo programação matemática, metaheurísticas e algoritmos evolucionários. Na próxima seção são apresentadas as técnicas de busca utilizadas neste trabalho.

2.5 Técnicas e Algoritmos de Busca

Uma técnica de busca é utilizada para solucionar um modelo matemático de otimização. “Algoritmo de busca” e “algoritmo de otimização” são termos intercambiáveis a “técnica de busca”. A literatura relacionada a otimização e algoritmos de busca é extensa, e a quantidade de técnicas de busca disponíveis é igualmente grande. De forma geral, técnicas de busca podem ser divididas entre algoritmos exatos e metaheurísticas.

De um lado mais ‘tradicional’ da Pesquisa Operacional, algoritmos exatos são os mais utilizados. Tais algoritmos são considerados ‘exatos’ pois são determinísticos e sempre encontram a solução ótima para o problema, ou seja, a melhor solução possível. Nesse sentido, podem ser citadas técnicas de programação matemática, como os algoritmos *Simplex* e *Branch and Bound*.

Apesar de sempre encontrarem a solução ótima, algoritmos exatos apresentam, em geral, um grande custo computacional. Apresentando, normalmente, uma ordem de comple-

xidade exponencial, algoritmos exatos sofrem do que é chamado na literatura de “explosão combinatorial”. Quando aplicados a pequenas instâncias do problema, a solução ótima é rapidamente encontrada, mas o aumento no tamanho da instância causa um aumento considerável no tempo de resposta, chegando a tempos computacionais não aplicáveis.

A despeito dessa desvantagem, alguns trabalhos de SBSE têm se voltado a aplicar técnicas exatas, inclusive a problemas da Engenharia de Requisitos (FREITAS; SOUZA, 2011)(FREITAS; COUTINHO; SOUZA, 2011). O trabalho em Freitas et al. (2011) discute ainda as situações em que técnicas exatas são aplicáveis em SBSE. Contudo, as metaheurísticas ainda são as técnicas de busca mais utilizadas em SBSE (HARMAN, 2007).

Metaheurísticas são técnicas de busca de propósito geral, não determinísticas e de caráter estocástico. Diferentemente das heurísticas, que são algoritmos desenvolvidos exclusivamente para resolução de um problema específico, as metaheurísticas têm a característica de serem aplicáveis a qualquer problema a partir de uma simples adaptação. São algoritmos não determinísticos pois não é garantido que a solução ótima será encontrada, onde inclusive, duas execuções para uma mesma instância podem retornar soluções diferentes. Por caráter estocástico, entende-se que o processo de busca é parcialmente guiado por eventos aleatórios.

Diferentemente dos algoritmos exatos, as metaheurísticas não apresentam uma garantia de encontrar a solução ótima do problema. Em compensação, elas são muito mais escaláveis computacionalmente. Para instâncias mais complexas de um certo problema, onde técnicas exatas levariam um tempo computacional intangível para chegar na solução ótima, metaheurísticas conseguiriam em tempo hábil alcançar soluções de boa qualidade (sub-ótimas). Assim, metaheurísticas são técnicas de busca interessantes para o cenário no qual é melhor obter uma boa solução do que nenhuma.

Problemas da Engenharia de Software são, de forma geral, problemas complexos, com muitas restrições, onde soluções ótimas não são previamente conhecidas (HARMAN; JONES, 2001). A quantidade de possíveis soluções, ou espaço de busca, normalmente é consideravelmente grande. Centenas de casos de teste a serem priorizados, centenas de requisitos a serem selecionados, milhares de linhas de código a serem otimizadas. Essa complexidade se reflete nas já existentes modelagens matemáticas para certos problemas, como o Problema do Próximo Release e o Problema do Planejamento de Releases, por exemplo, ambos problemas NP-completo. Dessa forma, a opção pelas metaheurísticas em SBSE foi natural. Além da escalabilidade dessas técnicas de busca, a facilidade de adaptação de tais algoritmos a um certo problema também se apresenta como um grande motivo para a aplicação de metaheurísticas em problemas de ES.

As metaheurísticas utilizadas no estudo empírico desse trabalho foram **Algoritmos Genéticos** e **Têmpera Simulada**. Como o objetivo desse trabalho é propor um modelo robusto para o NRP e avaliar seu comportamento, tais metaheurísticas foram escolhidas por serem duas

das técnicas de busca mais utilizadas em SBSE (HARMAN, 2007). Ainda segundo (HARMAN, 2007), uma técnica de **Busca Aleatória** também é aplicada para servir como um teste de sanidade dos demais algoritmos, ou seja, mostrar que algoritmos especializados de fato apresentam melhores resultados.

Uma descrição mais detalhada das técnicas de busca utilizadas neste trabalho são mostradas a seguir.

2.5.1 Algoritmos Genéticos

Algoritmos Genéticos (AGs) são uma técnica de busca inspirada em genética e na teoria da seleção natural de Darwin (ERASER, 1957)(HOLLAND, 1975). Apesar de ser aplicado a diversos problemas de otimização em diversas áreas de conhecimento, seu estudo teórico faz parte da linha de pesquisa nomeada como Computação Evolucionária (EIBEN; SMITH, 2003). Caracterizado como uma metaheurística, é um algoritmo de otimização não determinístico e de comportamento estocástico.

Pelo fato de ser uma técnica já muito estudada e em constante evolução, a quantidade de variações e adaptações desse algoritmo é relativamente grande. Dessa forma, o AG implementado para esse trabalho bem como a maioria dos conceitos básicos explicados nessa seção são baseados no livro (BURKE; KENDALL, 2005).

O AG consiste basicamente em manter uma certa população de indivíduos e evoluir essa população ao longo da execução do algoritmo. Cada indivíduo na população representa uma possível solução para problema e evoluir a população consiste em melhorar a qualidade de seus indivíduos. A qualidade de um indivíduo, também chamado de valor de aptidão ou valor de *fitness*, é dado pela função objetivo, ou função de *fitness*, do problema. A evolução da população é atingida a partir de operadores genéticos aplicados nos indivíduos, como **cruzamento**, **mutação** e **reparação**. O pseudo-código do Algoritmo Genético implementado neste trabalho é mostrado no Algoritmo 1.

A forma como os operadores genéticos vão agir sob os indivíduos, bem como alguns outros procedimentos do AG, são totalmente dependentes da codificação do indivíduo, ou seja, em como uma possível solução para o problema é representada. Como dito na seção 2.2, a representação da solução é um dos aspectos mais importantes na aplicação de uma técnica de busca, podendo ser feita de inúmeras formas. Para explicação dos operadores genéticos nesta seção, será utilizada a representação de solução mais comum na literatura e também empregada neste trabalho, que é a representação através de um vetor binário. Nessa representação, cada gene do indivíduo é um bit.

Primeiramente, é necessário criar a população inicial do AG. De forma geral, o mais comum é que a população inicial seja gerada de forma aleatória, o que garante uma boa

Algoritmo 1 Pseudo-código do Algoritmo Genético

```

População ← criarPopulaçãoInicial()
Filhos ← vazio
while Critério de Parada do
  calcularFitnessIndivíduos(População)
  for each indivíduo na População do
    pais ← selecionarPais(População)
    filho ← cruzamento(pais)
    mutação(filho)
    if filho é inválido then
      Reparar(indivíduo)
    end if
    Filhos ← filho
  end for
  População ← Filhos
end while
return melhor indivíduo na População

```

cobertura inicial do espaço de busca. Entretanto, uma geração mais sofisticada, utilizando informações específicas do domínio, é simples de ser implementada.

O AG continua a evolução da população até um certo critério de parada ser atingido. Dentre os critérios mais utilizados estão a definição de um número máximo de iterações ou um número máximo de avaliações de *fitness*. De forma semelhante à criação da população inicial, também são possíveis critérios de parada mais sofisticados, como critérios de convergência, por exemplo.

No início de cada iteração do AG, também chamada de geração, o valor de *fitness* de cada indivíduo na população deve ser calculado. Esse valor de *fitness* é calculado com base na função objetivo do problema e indica a qualidade de cada indivíduo, sendo utilizado para guiar todo o processo de busca do AG.

A evolução da população consiste em, a partir da população já existente, criar uma nova população que seja melhor que a atual. Os novos indivíduos da população (filhos) serão criados a partir dos indivíduos atuais (pais). Dessa forma, um certo conjunto de pais será selecionado para geração de cada novo filho. A teoria da seleção natural de Darwin é aplicada nesse exato momento, onde os indivíduos mais adaptados (com um maior valor de *fitness*) terão uma maior probabilidade de serem selecionados como pais. A ideia básica é que bons pais provavelmente gerarão bons filhos. Um estudo sobre diferentes formas para seleção dos pais pode ser vista em (BLICKLE; THIELE, 1996).

Após a seleção dos pais, o filho é criado a partir do operador genético de **cruzamento**. Esse operador é responsável por combinar o material genético dos pais (na maioria das vezes dois) para gerar um ou mais filhos, dependendo do tipo de cruzamento. Alguns tipos de

cruzamento para representação binária podem ser vistos na Figura 6.

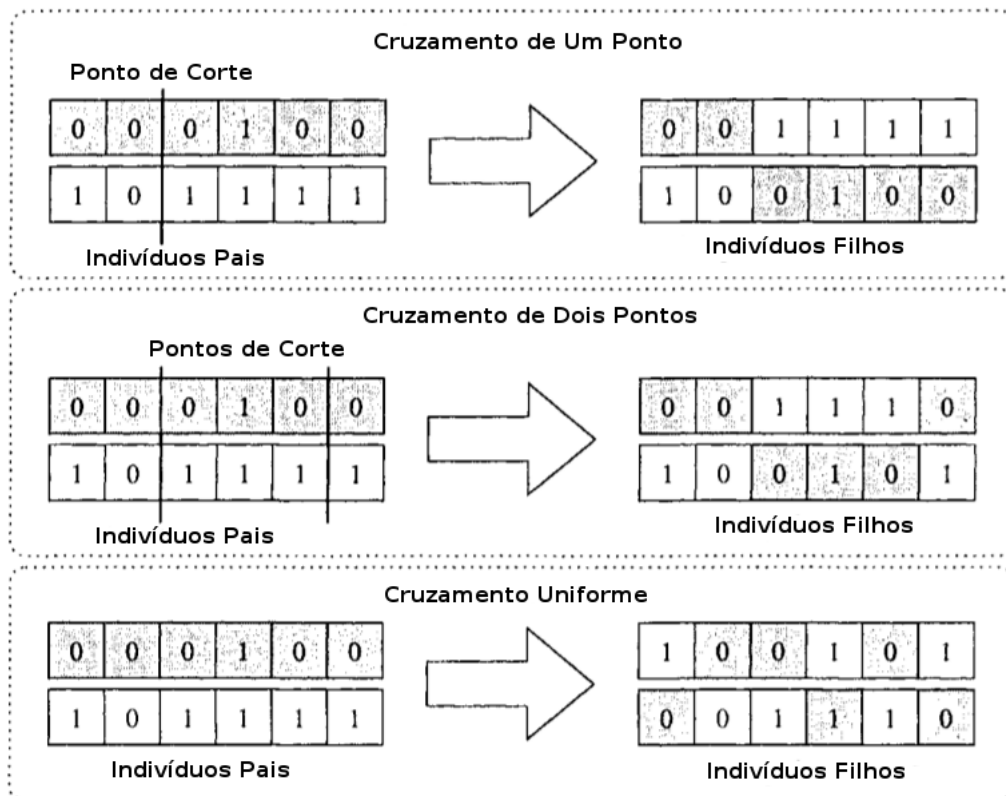


Figura 6 – Alguns tipos de Cruzamento para representação binária. Fonte: adaptado de (BURKE; KENDALL, 2005).

Todos os tipos de cruzamento mostrados na figura são realizados a partir de dois pais e geram dois filhos. Em mais detalhes, o **cruzamento de um ponto** funciona da seguinte forma: um ponto de corte é definido de forma aleatória. O ponto de corte divide os pais entre cabeça e cauda. O primeiro filho é então formado pela combinação da cabeça do primeiro pai com a cauda do segundo, e o segundo filho é formado pela cabeça do segundo pai e a cauda do primeiro. O **cruzamento de dois pontos** age de forma similar. Mas nesse caso são definidos dois pontos de corte, dividindo os pais entre cabeça, corpo e cauda. No **cruzamento uniforme** cada filho é formado por metade dos genes de cada pai, definidos aleatoriamente. Vale ressaltar ainda que a operação de **cruzamento** é condicionada a uma certa **probabilidade de cruzamento**. Caso essa probabilidade não seja atingida, os pais selecionados são simplesmente usados como filhos. De forma geral, a **probabilidade de cruzamento** é alta.

Após o filho ser gerado a partir do **cruzamento**, o mesmo passa pelo processo de **mutação**. Como o filho é gerado a partir de uma combinação dos pais, ele fica limitado ao material genético que já está presente na população. No sentido de otimização, a população cobre uma parcela do espaço de busca, os filhos gerados a partir dessa população estarão necessariamente dentro dessa mesma parcela do espaço de busca.

Assim, o processo de **mutação** tem o objetivo de modificar o filho, possibilitando

uma maior cobertura do espaço de busca. Como no processo de mutação descrito por Darwin, a **mutação** no AG consiste em uma pequena mudança no material genético do indivíduo. De forma semelhante ao **cruzamento**, existem diversas estratégias de mutação para representação binária. A **mutação por bit flip** é a mais comum e é mostrada na Figura 7.



Figura 7 – Mutação por Bit Flip. Fonte: (BURKE; KENDALL, 2005)

Tal estratégia de **mutação** é aplicada a cada bit do indivíduo seguindo uma certa **probabilidade de mutação**. No caso de um certo bit sofrer mutação, ele será alterado de 0 para 1 ou de 1 para 0. Ainda inspirado na teoria de Darwin, a **mutação** no AG não deve ocorrer com frequência. Um valor comum para a **probabilidade de mutação** é de $1/N$, onde N é a quantidade de bits no indivíduo. Dessa forma, em média, pelo menos um bit do indivíduo sofrerá mutação, como mostrado na Figura 7.

Ao aplicar o AG a um problema de otimização com restrições, os operadores de **cruzamento** e **mutação** podem gerar indivíduos inválidos. Apesar de alguns AGs manterem indivíduos inválidos na população, o mais comum é que se desenvolva um operador de **reparação**. Tal operador deve ser desenvolvido de forma específica para o problema e consiste em transformar um indivíduo inválido em um indivíduo que respeite todas as restrições.

Ao final de cada geração do AG, os filhos criados são copiados para a população, se tornando os novos pais da próxima geração e continuando o processo de evolução da população.

A execução do AG é baseada em eventos aleatórios. Assim, dependendo de certos eventos ao longo de uma certa geração, como uma má seleção dos pais, ou a escolha de um mau ponto de corte no cruzamento, é possível que bons indivíduos acabem se perdendo. Para evitar uma grande perda de informação entre gerações, pode ser utilizada uma estratégia de **elitismo**. Nessa técnica, uma certa porcentagem dos melhores indivíduos da população (elite) é automaticamente copiada para a próxima geração. Isso garante que o melhor indivíduo da próxima geração não será pior que o melhor indivíduo da geração passada, garantindo que a população nunca vai regredir em qualidade. A porcentagem de indivíduos que são automaticamente copiados é chamada de **taxa de elitismo**.

Ao final do algoritmo, o melhor indivíduo da população é retornado como solução final encontrada.

2.5.2 *Têmpera Simulada*

A Têmpera Simulada (TS) é uma metaheurística inspirada na técnica de têmpera, ou arrefecimento, processo da termodinâmica para manuseio e modelagem de metais (KIRKPATRICK; VECCHI; GELATT, 1983). De forma similar ao AG, a TS já foi estudada e aplicada a vários problemas de otimização, existindo diversas variações e adaptações desse algoritmo. O algoritmo de TS implementado neste trabalho e mostrado a seguir foi principalmente baseado no livro (BURKE; KENDALL, 2005).

A técnica de têmpera é utilizada principalmente para o manuseio e modelagem de metais. Consiste em esquentar o metal até uma certa temperatura em que o mesmo se encontre numa consistência em que seja possível sua modelagem. Após se chegar à forma desejada, o metal é esfriado lentamente, para voltar à consistência sólida sem perder a forma previamente definida. Quanto mais quente o metal, menos sólida será a sua consistência e mais fácil será a modelagem. A técnica de têmpera também é utilizada na modelagem de vidros.

O algoritmo da TS é inspirado na técnica de têmpera e foi desenvolvido como uma evolução de algoritmos de busca local baseados em vizinhança, como o *Hill Climbing*. Nesse tipo de algoritmo, após definida a representação da solução, é definido o conceito de solução vizinha. De maneira geral, uma solução vizinha é aquela que é parecida com a solução original segundo algum critério. A partir de uma solução inicial aleatória, o *Hill Climbing* gera vizinhos dessa solução inicial e escolhe o melhor deles para se tornar a solução corrente. Os vizinhos da solução corrente são avaliados e o melhor vizinho é escolhido. Esse processo se repete até não haver mais vizinhos melhores que a solução corrente. Percebe-se que esse é um algoritmo simples que é fortemente sujeito a ficar presos em ótimos locais, ou seja, soluções ótimas na sua vizinhança mas que são possivelmente piores que soluções em outra vizinhança.

A TS consiste em simular o processo de têmpera, onde a solução corrente é o objeto a ser modelado. Nesse algoritmo, soluções piores que a solução corrente podem ser aceitas como nova solução, possibilitando assim uma certa fuga de um ótimo local. A aceitação de uma solução pior que a corrente é probabilística e em função da temperatura do 'sistema'. No início do algoritmo a temperatura é alta e a solução ainda não é tão boa, então o algoritmo está mais propenso a aceitar uma solução pior. Ao final, com uma temperatura baixa e solução corrente possivelmente boa, a probabilidade de aceitar uma solução pior é pequena. O pseudocódigo da TS implementada nesse trabalho pode ser visto no Algoritmo 2.

Primeiramente, deve-se criar a solução inicial de forma aleatória e inicializar a temperatura com um certo valor de temperatura inicial. A TS pode, em alguns momentos, trocar a solução corrente por uma solução pior, sendo possível que essa solução melhor não seja mais encontrada ou superada. Dessa forma, a melhor solução encontrada até o momento é sempre guardada.

Algoritmo 2 Pseudo-código da Têmpera Simulada

```

soluçãoCorrente ← criaSoluçãoInicial()
melhorSolução ← soluçãoCorrente
temperatura ← temperaturaInicial
while temperatura > temperaturaFinal do
  for quantidade de vizinhos avaliados por iteração do
    vizinho ← soluçãoVizinha(soluçãoCorrente)
    if fitness(vizinho) < fitness(soluçãoCorrente) then
      soluçãoCorrente ← vizinho
      if fitness(soluçãoCorrente) < fitness(melhorSolução) then
        melhorSolução ← soluçãoCorrente
      end if
    else
      if  $\exp\left(\frac{\text{fitness}(\text{soluçãoCorrente}) - \text{fitness}(\text{vizinho})}{\text{temperatura}}\right) > \text{random}[0, 1)$  then
        soluçãoCorrente ← vizinho
      end if
    end if
  end for
  temperatura ← temperatura × taxaResfriamento
end while
return melhorSolução

```

Como na técnica de têmpera, o algoritmo de TS ‘resfria o sistema’ lentamente, diminuindo a temperatura a cada iteração. O critério de parada da TS está relacionado com essa diminuição da temperatura. O algoritmo para quando a temperatura atinge um valor abaixo da temperatura final previamente estipulada.

A cada iteração uma certa quantidade de vizinhos é avaliada. Os vizinhos de uma mesma iteração são avaliados utilizando a mesma temperatura, quanto mais vizinhos avaliados, maior será a cobertura do espaço de busca. Como dito anteriormente, o conceito de solução vizinha é diretamente ligado à representação da solução. Para uma representação binária, pode ser considerada uma solução vizinha aquela que for diferente da solução original em somente um bit.

A solução vizinha é gerada a partir da solução corrente e as duas são comparadas em valor de *fitness*. Quando a solução vizinha é melhor que a solução corrente, a primeira assume o posto de solução corrente automaticamente. Caso seu valor de *fitness* também seja melhor que o da melhor solução encontrada até então, a nova solução corrente torna-se a melhor solução. Vale ressaltar que no Algoritmo 2, uma solução melhor é a que apresenta *fitness* menor, caracterizando um problema de minimização. No caso de um problema de maximização, uma solução melhor é a que apresenta *fitness* maior.

No caso em que a solução vizinha apresenta um valor de *fitness* pior que a solução corrente, a probabilidade do algoritmo aceitar essa solução vizinha como solução corrente é

dada por $\exp\left(\frac{\text{fitness}(\text{soluçãoCorrente}) - \text{fitness}(\text{vizinho})}{\text{temperatura}}\right)$. Percebe-se que, quanto maior a temperatura, maior será a probabilidade da solução vizinha ser aceita. De forma semelhante, quanto menor a diferença entre as soluções, maior a probabilidade da solução vizinha se tornar a solução corrente. Vale ressaltar novamente que essa probabilidade mostrada no pseudo-código é para problemas de minimização, no caso de problemas de maximização, deve ser utilizada a probabilidade $\exp\left(\frac{\text{fitness}(\text{vizinho}) - \text{fitness}(\text{soluçãoCorrente})}{\text{temperatura}}\right)$.

Ao final de cada iteração, a temperatura é diminuída. Essa redução é feita em função da **taxa de resfriamento**. Naturalmente, a **taxa de resfriamento** assume valores no intervalo entre 0 e 1. De forma similar à **probabilidade de mutação** no AG, a **taxa de resfriamento** deve ser pequena, proporcionando assim, uma lenta resfrição. Ao final do algoritmo, a melhor solução encontrada até então é retornada.

2.5.3 Busca Aleatória

A Busca Aleatória consiste em gerar um certo número de soluções aleatórias para o problema e retornar a melhor solução encontrada. É uma técnica de busca muito simples que, apesar de apresentar resultados interessantes para alguns problemas de SBSE, como testes de software (WEGENER; MUELLER, 2001), normalmente é utilizada como ‘teste de sanidade’ para outros algoritmos (HARMAN, 2007). Os resultados da busca aleatória são comparados com os resultados encontrados por algoritmos de otimização mais sofisticados para se provar a complexidade do modelo e a necessidade da aplicação de melhores técnicas de busca.

O pseudo-código da Busca Aleatória implementada nesse trabalho é mostrado no Algoritmo 3.

Algoritmo 3 Pseudo-código da Busca Aleatória

```

melhorSolução ← criaSoluçãoAleatória()
for número de soluções a serem geradas do
  soluçãoAleatória ← criaSoluçãoAleatória()
  if fitness(soluçãoAleatória) < fitness(melhorSolução) then
    melhorSolução ← soluçãoAleatória
  end if
end for
return melhorSolução

```

A avaliação da solução aleatória mostrada no algoritmo caracteriza um problema de minimização. Para problemas de maximização, uma solução aleatória será melhor quando apresentar um valor de *fitness* superior.

2.6 Análise Estatística para Avaliação de Metaheurísticas em SBSE

Metaheurísticas, como dito anteriormente, são algoritmos não determinísticos e de caráter estocástico. Diferentes execuções de um mesmo algoritmo, com as mesmas configurações, para uma mesma instância, podem não produzir o mesmo resultado. Ainda assim, metaheurísticas são as técnicas de busca mais utilizadas em SBSE devido à sua capacidade de encontrar boas soluções para problemas complexos em um tempo hábil.

Devido ao aspecto não determinístico das metaheurísticas, avaliações e comparações dessas técnicas de busca devem ser realizadas considerando a sua *distribuição de probabilidade* e vários outros aspectos (ARCURI; BRIAND, 2011). Entretanto, derivar a *distribuição de probabilidade* de uma técnica de busca estocástica é uma tarefa complexa, sendo necessário um estudo teórico aprofundado. Além do mais, o estudo da *distribuição de probabilidade* de uma certa metaheurística varia de acordo com a sua adaptação ao problema e implementação. Dessa forma, grande parte dos estudos empíricos que aplicam metaheurísticas a problemas da ES realizam análises baseadas em métricas da Estatística Descritiva, como média, mediana, desvio padrão etc (BARROS; DIAS-NETO, 2011).

No trabalho proposto por Arcuri e Briand (2011) é mostrado um guia para realização de análises e testes estatísticos na avaliação de metaheurísticas em SBSE. De forma geral, devido ao caráter não determinísticos dos algoritmos, recomenda-se a execução de uma mesma técnica de busca para a mesma instância várias vezes. O número necessário de execuções não é estipulado, sabe-se que quanto mais execuções melhor, mas um número de execuções igual a 30 é aceitável. O resultado da execução de certo algoritmo para uma certa instância não é um valor único, mas sim esse conjunto de valores provenientes das diversas execuções. Esse conjunto de resultados pode ser chamado de amostra.

A partir dessa amostra, vários dados estatísticos podem ser retirados, sendo média e desvio padrão os mais utilizados. Apesar de úteis na análise de uma técnica de busca específica, esses atributos não são os mais adequados na comparação entre resultados, seja de diferentes técnicas de busca ou de uma mesma técnica de busca com diferentes parâmetros. Ao realizar comparações, recomenda-se a utilização de testes estatísticos que levem em consideração toda a amostra, e não somente os valores de média (ARCURI; BRIAND, 2011).

A escolha de qual teste estatístico utilizar deve ser feito baseado nas amostras. Se as amostras forem uniformemente distribuídas, testes paramétricos podem ser utilizados, caso contrário, devem ser utilizados testes estatísticos não paramétricos. No primeiro caso recomenda-se o teste de Fisher, enquanto que no segundo caso recomenda-se o teste de Wilcoxon.

Ambos os testes recebem duas amostras como entrada e retornam como resultado um valor chamado de *p-value*. De forma geral, os testes assumem que as duas amostras são diferentes e o *p-value* indica a probabilidade das amostras serem na verdade iguais. Considere

duas amostras a e b . Um p -value de $p_{a,b} = 0.01$, por exemplo, indica que existe uma probabilidade de 1% de a e b serem iguais, ou seja, uma alta probabilidade que as amostras são de fato diferentes.

Toda a análise estatística desse trabalho foi realizada através da ferramenta de computação estatística R (R-PROJECT, 2014), como também recomendado em (ARCURI; BRIAND, 2011).

2.7 Conclusões

Este capítulo procurou apresentar uma fundamentação teórica sobre os assuntos necessários para o completo entendimento deste trabalho. Foi mostrado primeiramente os conceitos básicos de Pesquisa Operacional e otimização matemática, mostrando a forma geral de um modelo matemático de otimização. Em seguida, foi mostrado como a linha de pesquisa de Engenharia de Software Baseada em Busca se propõe a utilizar a PO para solucionar de forma automatizada problemas complexos da Engenharia de Software. Foi mostrado um pequeno histórico sobre a SBSE bem como os conceitos básicos de como aplicar uma técnica de busca a um problema de ES.

Em seguida, foram apresentados alguns problemas oriundos da Engenharia de Requisitos já abordados pela SBSE, com um foco principal no Problema do Próximo Release. A Otimização Robusta também foi discutida, sendo mostrados seus conceitos fundamentais e a forma geral de um modelo robusto de otimização. Uma breve discussão sobre técnicas de busca foi conduzida, objetivando diferenciar algoritmos exatos e metaheurísticas. As técnicas de busca utilizadas neste trabalho também foram discutidas, apresentando-se tanto as inspirações para desenvolvimento dos algoritmos como seus respectivos pseudo-códigos. Finalmente, as ferramentas estatísticas que foram utilizadas neste trabalho para avaliação dos resultados alcançados pelas metaheurísticas foram devidamente apresentadas.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta alguns dos trabalhos relacionados à esta pesquisa. Primeiramente, são apresentados os trabalhos diretamente relacionados com o Problema do Próximo Release, nas diferentes formas que esse problema foi modelado e nas diferentes técnicas de busca aplicadas. A seguir, são apresentados trabalhos de SBSE que de alguma forma trataram incertezas em problemas relacionados a requisitos. Finalmente, são mostrados os trabalhos de SBSE que tratam incertezas em outros problemas de ES.

3.1 Problema do Próximo Release

A primeira modelagem matemática formal do NRP é mostrada em (BAGNALL; RAYWARD-SMITH; WHITTLEY, 2001). Nessa abordagem inicial, são considerados que existem vários clientes onde cada um tem uma importância diferente para a organização. Cada cliente aponta quais requisitos ele deseja que sejam incluídos no próximo *release*. Para satisfazer um cliente, todos os requisitos indicados por ele devem ser implementados. O objetivo é selecionar um conjunto de clientes cuja soma das respectivas importâncias seja maximizada, de forma que a soma dos custos dos requisitos de cada cliente não ultrapasse um certo orçamento disponível para o *release*. Para solução do problema foram utilizadas técnicas de programação linear, heurísticas de vizinhança como *Hill Climbing* e a única metaheurística aplicada foi a TS.

Ainda seguindo essa modelagem inicial, de selecionar um conjunto de clientes que tenham sua importância maximizada, Jiang et al. (2010) apresenta um algoritmo de colônia de formigas híbrido para a resolução do problema. A construção da solução é realizada utilizando o *Ant System* tradicional e após cada formiga obter a sua solução, o *Hill Climbing* é utilizado como operador de busca local para melhorar a solução encontrada. Comparações com o *Ant System* sem busca local e TS são realizadas, onde o algoritmo proposto alcançou os melhores resultados.

Considerando essa modelagem de seleção de clientes, uma abordagem utilizando uma adaptação da metaheurística Greedy Randomized Adaptive Search Procedure (GRASP), chamada de GRASP Reativo, é mostrada em (LINHARES et al., 2010). Constata-se que esse algoritmo é adequado ao problema quando é comparado com a TS, AG e GRASP tradicional.

O trabalho por Jiang, Xuan e Ren (2010) é dedicado à resolução de grandes instâncias do NRP com seleção de clientes. Para tanto, desenvolve um algoritmo de espinha dorsal (tradução livre para *Backbone Algorithm*). Essa técnica reduz constantemente a instância através da detecção de partes comuns de vários ótimos locais. Essas partes comuns formam a espinha dorsal da solução final. Após um certo grau de diminuição, a instância reduzida é solucionada utilizando programação linear. Resultados são comparados com a LMSA (variação da

TS), algoritmo este que vinha encontrando os melhores resultados até então.

O trabalho em Akker et al. (2005) modela a seleção dos requisitos no NRP de forma diferente da primeira modelagem em Bagnall, Rayward-Smith e Whittley (2005). Cada requisito tem uma certa importância, dada a partir dos clientes, e o objetivo é selecionar um subconjunto de requisitos que maximize a importância total do *release*. Cada cliente ainda tem um certo valor para a organização e atribui uma certa importância para cada requisito. A importância global do requisito é calculada a partir de uma soma ponderada das importâncias atribuídas por cada cliente. Esse novo modelo é resolvido a partir da utilização de programação linear inteira.

Utilizando essa mesma modelagem de selecionar os requisitos de forma direta, a partir da importância dos mesmos, Sagrado, Aguila e Orellana (2010) apresenta uma adaptação do *Ant Colony System* (ACS) para esse modelo. A partir de comparações com a TS e AG, é mostrada a grande superioridade do ACS, tanto em termos da qualidade da solução como de velocidade de convergência.

Uma dificuldade encontrada nas pesquisas direcionadas ao NRP é a não utilização de dados reais para os estudos empíricos. Nas empresas de desenvolvimento de software em geral, informações relacionadas a clientes e requisitos são sigilosas, fazendo com que a maioria dos trabalhos realize seus estudos e análises em instâncias geradas de forma aleatória. A pesquisa em Baker et al. (2006) torna-se relevante então, ao realizar os experimentos em bases de dados reais de uma grande empresa de telecomunicações. As comparações são feitas entre um algoritmo guloso simples, TS e soluções geradas por especialistas dentro da empresa. Os dois algoritmos superaram as soluções desenvolvidas pelos especialistas, com a TS chegando a ser 50% melhor em alguns pontos.

Os trabalhos mais recentes estão mostrando uma preocupação cada vez maior em tornar a modelagem matemática do NRP cada vez mais próxima de um ambiente real de planejamento de *releases*. Nesse sentido, uma grande ênfase é dada às relações, ou interdependências, entre requisitos. A pesquisa em Sagrado, Aguila e Orellana (2011), considera as interdependências funcionais entre requisitos no momento da seleção. Exemplos de relações funcionais abordadas são: precedência, quando um requisito depende de outro para poder ser implementado; combinação, quando dois requisitos devem ser implementados no mesmo *release*; e exclusão, quando a seleção de um requisito implica na exclusão de outro e vice-versa. Para resolução do modelo utiliza ACS, GRASP e AG. O ACS novamente atingiu os melhores resultados.

Ainda com relação às interdependências, o trabalho em Ferreira e Souza (2012) trata não somente as relações funcionais, mas também as interdependências entre requisitos que alteram as importâncias e custos de implementação dos mesmos. Apresenta uma adaptação do ACS para solucionar o NRP com essas várias interdependências. Resultados do ACS mostraram-se novamente melhores que os resultados da TS e do AG.

No trabalho em (BRASIL et al., 2012), é abordada a relação de similaridade entre requisitos, onde a seleção de requisitos semelhantes entre si para o próximo *release* causa uma certa penalidade para a solução. Assim, o objetivo é selecionar os requisitos que, além de maximizar a importância do *release*, não sejam similares entre si. Utiliza a TS e um AG para realização das análises.

Nenhum dos trabalhos citados acima trata ou considera nenhuma das incertezas presentes nos requisitos ou em qualquer outro aspecto do NRP.

3.2 Tratamento de Incertezas em Problemas da Engenharia de Requisitos

Todos os trabalhos citados na seção acima utilizam uma formulação mono-objetivo e podem ser considerados ferramentas de tomada de decisão. O engenheiro de software entraria com os dados dos requisitos e a ferramenta retornaria um conjunto de requisitos para serem implementados no próximo *release*. Apesar de alguns trabalhos mostrarem que soluções encontradas por metaheurísticas normalmente superam soluções desenvolvidas por especialistas (SOUZA et al., 2010)(BAKER et al., 2006), ainda existe uma certa relutância por conta dos profissionais em aceitar soluções totalmente automatizadas. Com o intuito de prover uma ferramenta de suporte a decisão, permitindo que o engenheiro de software tome decisões baseadas em diferentes preferências e prioridades, uma versão multiobjetivo do NRP foi proposta por Zhang, Harman e Mansouri (2007). Apesar de não ser o foco principal do trabalho, essa modelagem multiobjetivo permite um certo tratamento de incertezas no NRP, principalmente com relação às incertezas de custo dos requisitos.

Nessa formulação multiobjetivo do NRP, o custo total do *release* é tratado como um objetivo ao invés de uma restrição, como é o caso na versão mono-objetivo. Em uma abordagem multiobjetivo não existe um único *release* ótimo, mas sim um conjunto de *releases* igualmente bons entre si. Tal conjunto é chamado de *frente de pareto*. Nesse caso a técnica de busca retorna uma *frente de pareto* e o engenheiro de software deve escolher, dentre esse conjunto de *releases*, qual deles será de fato implementado. Tal abordagem pode ser considerada uma ferramenta de apoio a decisão, pois é o engenheiro de software que tem a decisão final de qual *release* será implementado.

A partir desse conjunto de *releases*, podem ser feitas análises com relação às possíveis incertezas no custo dos requisitos, como: “quanto seria perdido na importância total do *release* se o custo total fosse 20% maior do que o estimado?”, ou “qual seria o ganho na importância se o custo dos requisitos fosse 10% menor?”. Esse modelo multiobjetivo ainda foi estudado mais a fundo, sendo aplicado a um conjunto maior de instâncias e realizando-se um conjunto maior de análises com relação ao seu comportamento (DURILLO et al., 2009)(DURILLO et al., 2011). Entretanto, essa abordagem não considera incertezas na importância dos

requisitos e esse tipo de análise da incerteza do custo só pode ser realizada com base no custo total e final do *release*.

Como forma de tratar as incertezas nas estimativas de importância dos requisitos, essas possíveis mudanças durante o ciclo de vida do software foram abordadas por Zhang et al. (2010). Nesse trabalho, cada requisito recebe um valor de importância chamado ‘today value’, que representa a importância daquele requisito no momento do planejamento do *release*. Assume-se que esse valor de importância, em algum momento após o desenvolvimento do *release*, mudará para um certo ‘future value’, representando a importância futura do requisito. Assim, a abordagem procura balancear as necessidades atuais e futuras da organização considerando esses dois valores de importância, juntamente com o custo total do *release*, através de uma formulação multiobjetivo. Considerando que o software será utilizado durante uma quantidade de tempo considerável, é improvável que os requisitos terão somente uma única importância futura. Tais valores de importância podem mudar constantemente durante a vida útil do software, configurando vários possíveis valores para a importância dos requisitos. Nessa situação, considerar vários valores de importância através de uma abordagem multiobjetivo, onde cada valor de importância caracteriza um objetivo diferente, mostra-se inviável tanto no âmbito computacional como no âmbito de tomada de decisão.

No sentido de analisar o impacto de possíveis incertezas nas estimativas de custos dos requisitos, o trabalho em Harman et al. (2009) realiza uma análise de sensibilidade do NRP. Tanto a modelagem mono-objetivo como a modelagem multiobjetivo do NRP são analisadas com relação às incertezas nos custos. Cada instância é primeiramente solucionada através de uma certa técnica de busca. A seguir, os dados dos custos dos requisitos dessa mesma instância são alterados, simulando possíveis incertezas. A instância alterada é então solucionada utilizando a mesma técnica de busca. Resultados provenientes de diversas variações na mesma instância são então analisados para avaliar o impacto que as incertezas causam na qualidade da solução final. Como esperado, quanto maior o erro na estimativa de custo, menor será a qualidade da solução. Tanto para requisitos de baixo custo como para requisitos de alto custo de desenvolvimento, um grande erro de estimativa tem um grande impacto. Para requisitos de alto custo, qualquer erro de estimativa já ocasiona uma perda considerável de qualidade, independente se esse erro é grande ou pequeno. Vale ressaltar que esse trabalho faz somente uma análise das incertezas, não apresenta nenhuma proposta para tratamento das mesmas num contexto de SBSE.

Os artigos citados acima são os únicos trabalhos de SBSE que consideram incertezas em problemas da Engenharia de Requisitos, seja a nível de análise ou de tratamento. Com relação à literatura de Engenharia de Software em geral, possíveis incertezas nos requisitos são amplamente estudadas, principalmente na questão de estimativas de custo de desenvolvimento, sendo impossível citar todos os trabalhos que abordam essa temática. Dessa forma,

uma revisão sistemática sobre a avaliação de incertezas em estimativas de custo de requisitos de software pode ser vista em (JORGENSEN, 2005). Esse trabalho também apresenta um guia genérico para prever os possíveis valores incertos nas estimativas de custo. De forma geral, dentre as várias estratégias mostradas, recomenda-se que não se siga somente intuição para prever os possíveis erros mas que também não se substitua de forma completa a opinião de especialistas. Além disso, é sugerido que o processo de previsão das incertezas seja acompanhado e validado por profissionais que não estão diretamente envolvidos no projeto.

3.3 Tratamento de Incertezas em outros Problemas da Engenharia de Software

Com relação ao tratamento de incertezas em outros problemas da ES, o trabalho por Antoniol, Penta e Harman (2004) propõe uma abordagem baseada em busca para o gerenciamento de projetos na presença de incertezas. As atividades do projeto são chamadas de ‘pacotes de trabalho’, que devem ser alocados a um grupo de funcionários para realizar a tarefa. A proposta consiste em dois algoritmos genéticos utilizados em sequência. O primeiro AG procura pela melhor sequência para execução dos pacotes de trabalho. A partir da ordem de execução dos pacotes de trabalho, o segundo AG busca a melhor alocação possível de funcionários para as respectivas atividades. O tratamento de incertezas é na verdade uma análise de sensibilidade do que aconteceria se alguns aspectos do problema fossem mal estimados. Dentre esses aspectos estão a possibilidade de saída de funcionários durante o projeto e erros de estimativa de custo dos pacotes de trabalho. Contudo, nenhuma abordagem para tratamento de tais situações é proposta.

Ainda no contexto de gerenciamento de projetos, o trabalho em Gueorguiev, Harman e Antoniol (2009) propõe uma abordagem baseada em otimização multiobjetivo para balancear robustez e tempo de entrega do software. Esse trabalho continua chamando as atividades do projeto de pacotes de trabalho e defende que robustez e tempo de entrega são objetivos conflitantes em um projeto de software. Um projeto é dito robusto quando consegue executar todas as atividades planejadas dentro do limite de tempo previamente estipulado. Sabendo que o tempo de execução de alguns pacotes de trabalho podem ser maiores que o estimado e que novos pacotes de trabalho podem ser incluídos durante o projeto, os gerentes muitas vezes estipulam o tempo do projeto para conseguir ‘encaixar’ essas atividades imprevistas. Quanto mais folgas são planejadas para tratar das incertezas, mais robusto é o projeto, mas ao mesmo tempo maior será o tempo de entrega, caracterizando os objetivos conflitantes. O AG multiobjetivo SPEAII é utilizado para gerar a *frente de pareto*, proporcionando para o gerente de projeto uma análise do balanceamento entre a robustez e tempo de entrega. Resultados demonstram que existem os chamados *sweet points* no espaço de busca, ou seja, pontos onde é possível melhorar muito um certo objetivo perdendo pouco do outro objetivo.

Ambos trabalhos citados acima estão relacionados ao gerenciamento de projetos de

software, no qual o planejamento de requisitos é intrínseco e é uma das atividades realizadas. De fato, Gueorguiev, Harman e Antoniol (2009) defende que os problemas de SBSE que tratam de requisitos, como o NRP, deveriam ser incorporados a abordagens mais genéricas para o gerenciamento do projeto como um todo. Este trabalho de dissertação não entra no mérito dessa discussão, concorda que são atividades pelo menos semelhantes, mas prefere citar esses trabalhos relacionados de forma separada por conta dos mesmos não indicarem de forma explícita que estão abordando temas da Engenharia de Requisitos.

Uma abordagem que leva em consideração incertezas para o refatoramento de código utilizando técnicas de busca pode ser visto em (MKAOUER et al., 2014). A refatoração de código tem como objetivo melhorar a manutenibilidade, legibilidade e alguns outros aspectos de qualidade do software. A detecção de pontos de refatoração em um sistema é feita, na maioria das vezes, a partir de *bad smells*. Esse termo é utilizado para denominar um conjunto de métricas que indicam o quanto uma classe ou conjunto de classes não está bem implementada. O trabalho defende que alguns aspectos dos *bad smells* são incertos, que podem mudar ao longo do próprio processo de refatoração. De forma específica, o trabalho trata possíveis mudanças na gravidade atribuída a cada *bad smell* e na importância dada a cada classe que contém os *bad smells*. A partir de uma adaptação do AG multiobjetivo NSGAI, o trabalho procura encontrar soluções de refatoração que apresentem um balanceamento ótimo entre os vários possíveis valores dessas incertezas.

Os sistemas denominados Dynamic Adaptive Systems (DAS) são aqueles que estão altamente sujeitos a incertezas ambientais e de condições de operação. Tais sistemas devem ser capazes de adaptar seu comportamento diante de uma série de situações inesperadas. No artigo por Ramirez et al. (2012), é proposta uma abordagem baseada em busca para automaticamente definir a relaxação das regras de adaptação de um DAS. Relaxação das regras de adaptação consiste na definição de regras *fuzzy* que especificam um certo limite para o qual um objetivo pode ser temporariamente não satisfeito e o sistema ainda apresentar um comportamento aceitável. A abordagem gerou regras capazes de diminuir a quantidade de adaptações necessárias e ainda modelos de objetivos de igual ou maior qualidade que os criados manualmente por engenheiros de requisitos.

3.4 Conclusões

Este capítulo procurou mostrar alguns dos trabalhos relacionados a essa pesquisa. Tais trabalhos foram divididos entre relacionados diretamente ao NRP, tratamento de incertezas em problemas da Engenharia de Requisitos e tratamento de incertezas em problemas da Engenharia de Software em geral.

Com relação ao NRP, foram mostrados trabalhos que tratam esse problema de di-

ferentes formas, com diferentes modelagens e diferentes técnicas de busca. Mostrou-se que existem, de forma geral, duas modelagens diferentes para o problema. A primeira seleciona clientes para serem satisfeitos, incluindo os requisitos desses clientes no próximo *release*. A segunda obtém valores de importância para os requisitos através dos clientes e seleciona os requisitos para o próximo *release* diretamente. Várias técnicas de busca já foram aplicadas a ambas modelagens, sendo as metaheurísticas Têmpera Simulada, Algoritmos Genéticos e Otimização por Colônia de Formigas as mais utilizadas. Percebe-se também uma tendência a tornar o modelo matemático cada vez mais próximo de um ambiente real de planejamento de *releases*, considerando as várias possíveis interdependências entre requisitos.

Foram poucos os trabalhos de SBSE que de alguma forma consideraram as incertezas relacionadas aos requisitos. Sendo que a maioria dos trabalhos citados não trata de fato as incertezas, mas somente realiza ou possibilita uma análise com relação ao impacto das mesmas na solução final. Com exceção de um trabalho, todos os demais citados nessa seção utilizam algum tipo de abordagem multiobjetivo para considerar as incertezas dos requisitos. Nenhum deles utiliza os conceitos e técnicas da Otimização Robusta para esse tratamento das incertezas. Apesar de serem poucos os trabalhos de SBSE que tratam incertezas nos requisitos, esse é um tópico bastante estudado na literatura de Engenharia de Software em geral, como pode ser visto a partir da revisão sistemática citada.

Com relação ao tratamento de incertezas por trabalhos de SBSE em outros problemas da Engenharia de Software, o gerenciamento de projetos foi abordado em duas oportunidades. Tanto de forma inicial, a partir de uma simples análise de sensibilidade, como de forma mais concreta, ao definir o conceito de projeto robusto e tratar essa robustez a partir de uma modelagem multiobjetivo. Possíveis incertezas nas métricas de *bad smells* para refatoração de software também foram tratadas. Novamente a partir de uma modelagem multiobjetivo, com o intuito de balancear objetivos de refatoração conflitantes.

No próximo capítulo, será apresentada de forma detalhada a proposta de tratamento das incertezas do Problema do Próximo Release utilizando os conceitos e técnicas da Otimização Robusta.

4 PROPOSTA DE MODELAGEM ROBUSTA PARA O NRP

Esse capítulo apresenta a proposta de modelagem matemática robusta para o NRP. É dividido em duas seções. A primeira apresenta o modelo propriamente dito e a segunda mostra um pequeno exemplo de utilização do mesmo, facilitando assim, o entendimento.

4.1 Modelo Matemático

Considere N como o número de requisitos disponíveis para serem selecionados para o próximo *release*. Dado um certo conjunto de requisitos $R = \{r_1, r_2, \dots, r_N\}$, os valores de importância dos requisitos são representados pelo conjunto $V = \{v_1, v_2, \dots, v_N\}$, onde v_i indica a importância do requisito r_i . De forma similar, os custos de desenvolvimento dos requisitos são representados pelo conjunto $C = \{c_1, c_2, \dots, c_N\}$, onde o custo do requisito r_i é indicado por c_i . Uma formulação básica para o NRP é mostrada a seguir:

$$\text{maximizar } \sum_{i=1}^N v_i \cdot x_i \quad (4.1)$$

$$\text{sujeito a } \sum_{i=1}^N c_i \cdot x_i \leq b \quad (4.2)$$

onde b é o orçamento do *release*. Nesse modelo, a função objetivo é o somatório das importâncias dos requisitos selecionados para o *release*, representando a importância total do *release*. O modelo ainda apresenta uma restrição, que é exatamente o fato da soma dos custos de desenvolvimento dos requisitos selecionados não ultrapassar o orçamento disponível. É utilizada uma representação de solução binária, onde um *release* é representado pelas variáveis de decisão $X = \{x_1, x_2, \dots, x_N\}$, onde $x_i \in \{0, 1\}$. Uma variável $x_i = 1$ indica que o requisito r_i está incluído no *release* e $x_i = 0$ o contrário.

Como indicado por Harker, Eason e Dobson (1993), a ocorrência de certos eventos pode alterar a importância dos requisitos durante o desenvolvimento do *release*. A incerteza na importância dos requisitos pode então ser classificada como uma incerteza do tipo **Mudanças ambientais e de condições de operação** (tipo A), mostrada na seção 2.4. Considere um projeto para desenvolvimento de um ‘software de prateleira’, por exemplo. Uma funcionalidade com potencial inovador seria possivelmente tratada como um requisito muito importante e seria desenvolvida o mais cedo possível. Entretanto, no caso de uma empresa concorrente lançar um software com uma funcionalidade igual ou parecida antes da primeira empresa lançar o seu produto, a importância de tal requisito diminuirá. Em tal situação, a primeira empresa teria desperdiçado recursos ao desenvolver esse requisito com uma prioridade tão alta. Como é possível

constatar, a ocorrência de um certo evento (lançamento de funcionalidade parecida por outra empresa) alterou o valor de importância do requisito. Assim, a quantidade de valores que a importância de um requisito pode assumir é discreta e depende do conjunto de eventos que de fato têm alguma influência na importância de tal requisito.

Dessa forma, a incerteza na importância dos requisitos mostra-se adequada para ser quantificada utilizando o conceito de cenários (YU, 1996). Um cenário pode ser definido como um conjunto de valores que representam diferentes contextos devido à ocorrência de certos eventos, como no exemplo mostrado acima. Portanto, é definido um conjunto de M cenários $S = \{s_1, s_2, \dots, s_M\}$, onde M é a quantidade de cenários, ou seja, M eventos que alteram o valor de importância dos requisitos. Cada cenário é representado por $s_j \subset S | s_j = \{v_1^j, v_2^j, \dots, v_N^j\}$, com v_i^j indicando a importância do requisito r_i no cenário s_j . Para considerar todos os possíveis valores de importância dos requisitos, e assim tratar essa incerteza, devem ser definidas ainda as probabilidades de ocorrência de cada cenário. Assim, para cada cenário s_j , é definida a priori uma probabilidade de ocorrência p_j , de forma que $\sum_{j=1}^M p_j = 1$. Finalmente, a importância v_i de um requisito no modelo robusto proposto é calculado por:

$$v_i = \sum_{j=1}^M v_i^j \cdot p_j \quad (4.3)$$

A formulação robusta para a importância dos requisitos mostrada acima considera todos os possíveis cenários, ponderando cada valor de importância pela probabilidade do respectivo cenário. Na situação em que as probabilidades dos cenários são difíceis de se definir, pode-se considerar todos os cenários com a mesma probabilidade de ocorrência. A importância do requisito proposta neste trabalho é na verdade uma generalização da importância na formulação básica do NRP. Ao se considerar a existência de um único cenário, ou seja, $M = 1$ e $S = \{s_1\}$ com probabilidade $p_1 = 1$, a importância do requisito será a mesma como mostrada na Equação 4.1.

A incerteza relacionada com o custo dos requisitos é diferente. Ao se considerar as incertezas de custo, o modelo deve garantir que a solução continuará respeitando a restrição de orçamento. Tal incerteza pode ser identificada como uma **Incerteza no cumprimento das restrições** (tipo D), também mostrada na seção 2.4. Com relação à quantificação, devido ao fato de que os custos normalmente variam de forma independente e não discreta, definir um conjunto de cenários baseado na ocorrência de eventos não é uma tarefa simples. Além do mais, incertezas nos custos dos requisitos são variáveis aleatórias contínuas e normalmente quantificadas utilizando intervalos de variação (YANG; HU; JIA, 2008).

Assim, a incerteza de custo dos requisitos será quantificada como mostrado a seguir. Considere c_i como sendo a estimativa de custo do requisito r_i , obtida de forma tradicional a

partir de alguma técnica de estimativa de custo. É definido ainda um valor \hat{c}_i que indica a variação máxima esperada do custo. O custo do requisito ao final do *release* é representado por \bar{c}_i e é calculado em função da sua própria estimativa e variação esperada, de forma que $c_i - \hat{c}_i \leq \bar{c}_i \leq c_i + \hat{c}_i$. Em outras palavras, o custo real do requisito durante a fase de planejamento do *release* é desconhecido, mas estará necessariamente dentro do intervalo definido pela sua estimativa de custo e variação esperada.

Uma possível formulação robusta para o custo total do *release* é mostrada a seguir:

$$\sum_{i=1}^N c_i \cdot x_i + \sum_{i=1}^N \hat{c}_i \cdot x_i \quad (4.4)$$

No caso acima, além da soma dos custos de todos os requisitos selecionados para o próximo *release*, o custo total ainda considera a soma de todas as respectivas variações de custo \hat{c}_i . Essa abordagem garante que, mesmo no pior caso, onde os custos de todos os requisitos serão máximos (dado por $c_i + \hat{c}_i$), o orçamento do *release* será satisfeito. Essa é, claramente, uma abordagem muito conservadora, pois assume que todas as estimativas de custo estão erradas e ainda para o máximo valor possível.

Em projetos de desenvolvimento de software reais, no entanto, diferentes equipes de desenvolvimento têm distintas habilidades de estimativa, normalmente relacionadas com a experiência da equipe. Tais habilidades podem ser mensuradas a partir de análises de projetos passados. Para uma certa equipe de desenvolvimento, pode ser extraído, em média, quantas estimativas de custo foram de fato corretas. A fim de considerar essa informação e tornar o modelo robusto mais realista, é definido um parâmetro de controle Γ (BERTSIMAS; SIM, 2004), que indica o nível esperado de falhas nas estimativas de custo. Assim, em uma situação onde as estimativas da equipe são 30% incorretas, em um projeto com $N = 50$ requisitos, por exemplo, o parâmetro de controle seria $\Gamma = 15$. Tal valor indica que existe uma expectativa de que 15 requisitos terão custos de desenvolvimento diferentes dos que foram estimados.

Utilizando esse novo parâmetro de controle, o custo total do *release* no modelo robusto proposto neste trabalho é calculado por:

$$\begin{cases} \sum_{i=1}^N c_i \cdot x_i + \max_{W \subseteq R, |W| \leq \Gamma} \sum_{i \in W} \hat{c}_i \cdot x_i, & \text{se } \Gamma \neq 0 \\ \sum_{i=1}^N c_i \cdot x_i, & \text{se } \Gamma = 0 \end{cases} \quad (4.5)$$

O custo total do *release* é composto pela soma das estimativas de custo c_i e um

segundo fator, que foi adicionado para garantir um certo nível de robustez controlado pelo fator Γ , como explicado a seguir. Considerando que existe uma expectativa que Γ requisitos terão seus custos estimados incorretamente, a formulação busca por um subconjunto $W \subseteq R$ com cardinalidade $|W| \leq \Gamma$, onde a soma das variações de custo \hat{c}_i seja máxima. Os requisitos dentro de W são considerados como erroneamente estimados, e seu custo de desenvolvimento será dado por $\bar{c}_i = c_i + \hat{c}_i$, enquanto os demais terão custo $\bar{c}_i = c_i$. Em outras palavras, como não é possível saber antecipadamente quais requisitos têm estimativas de custo erradas, o modelo garante que, mesmo se a equipe de desenvolvimento errar as estimativas dos requisitos com maior variação, a solução ainda será válida.

A partir da formulação robusta acima, é possível novamente chegar na formulação básica do NRP ou na abordagem mais conservadora. Utilizando $\Gamma = 0$, assume-se que a equipe não errou nenhuma estimativa. Nesse caso, $W = \emptyset$ e o custo total do *release* na Equação 4.5 será o mesmo da Equação 4.2, o modelo clássico do NRP. Além disso, todas as variações de custo serão consideradas quando $\Gamma = N$, o que leva a formulação de volta para a abordagem conservadora mostrada na Equação 4.4.

Finalmente, o modelo robusto para o Problema do Próximo Release proposto neste trabalho é formalmente apresentado a seguir:

$$\begin{aligned} & \text{maximizar} \quad \sum_{i=1}^N \sum_{j=1}^M v_i^j \cdot p_j \cdot x_i \\ & \text{sujeito a} \quad \begin{cases} \sum_{i=1}^N c_i \cdot x_i + \max_{W \subseteq R, |W| \leq \Gamma} \sum_{i \in W} \hat{c}_i \cdot x_i \leq b, & \text{se } \Gamma \neq 0 \\ \sum_{i=1}^N c_i \cdot x_i \leq b, & \text{se } \Gamma = 0 \end{cases} \end{aligned}$$

onde, $x_i \in \{0, 1\}$

R é o conjunto de requisitos

N é o número de requisitos

M é o número de cenários

v_i^j é a importância do requisito r_i no cenário s_j

p_j é a probabilidade de ocorrência do cenário s_j

c_i é a estimativa de custo do requisito r_i

\hat{c}_i é a variação máxima esperada do custo do requisito r_i

Γ é o parâmetro de controle da robustez

b é o orçamento do *release*

4.2 Exemplo de Utilização

O comportamento do modelo robusto proposto neste trabalho é delineado no exemplo a seguir.

Considere uma instância do NRP com seis requisitos $R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$. O cliente indicou dois possíveis cenários para os valores de importância dos requisitos. Eles são representados por $s_1 = \{4, 1, 9, 3, 8, 5\}$ e $s_2 = \{6, 5, 8, 2, 3, 5\}$. A probabilidade de ocorrência de cada cenário é $p_1 = 0.7$ e $p_2 = 0.3$. A importância de cada requisito é então calculada através da Equação 4.3, gerando o conjunto de importâncias dos requisitos $V = \{4.6, 2.2, 8.7, 2.7, 6.5, 5\}$. Pelo fato do primeiro cenário ser mais provável de ocorrer do que o segundo, os valores resultantes de V são mais próximos dos valores de s_1 do que dos valores de s_2 . Ao utilizar as importâncias resultantes V , o modelo pode maximizar a importância total do *release* considerando todos os possíveis valores de importância dos requisitos.

Considere que a equipe de desenvolvimento definiu as estimativas de custo de cada requisito como $C = \{4, 5, 5, 4, 6, 3\}$. A variação de custo esperada foi configurada como metade do custo do requisito e é representada por $\hat{C} = \{2, 2.5, 2.5, 2, 3, 1.5\}$. O orçamento do *release* foi configurado para $b = 22$. Considerando um nível de robustez $\Gamma = 0$, todas as estimativas são consideradas corretas e o modelo retorna para o NRP clássico. A solução ótima nesse caso é o subconjunto $\{r_1, r_3, r_4, r_5, r_6\}$, indicando que tais requisitos foram selecionados para o próximo *release* e consistindo em uma importância total de 27.5 (a soma dos valores de importância dos requisitos r_1, r_3, r_4, r_5 e r_6). Como o nível de robustez é $\Gamma = 0$, nenhuma variação de custo foi considerada e o custo total é 22 (a soma das estimativas de custo dos requisitos r_1, r_3, r_4, r_5 e r_6). Percebe-se que as soluções ótimas são normalmente encontradas no limite das restrições. Ao se considerar as incertezas de custo, contudo, tais soluções podem ser inválidas.

Quando o nível de robustez é configurado para $\Gamma = N$, por exemplo, o modelo torna-se o pior caso, onde todas as estimativas de custo são consideradas erradas. Em tal situação, todas as variações de custo serão consideradas e todos os requisitos apresentarão custo máximo ($c_i + \hat{c}_i$). Nesse novo ambiente, a solução ótima seria o subconjunto $\{r_3, r_5, r_6\}$ com um subconjunto $W = \{r_3, r_5, r_6\}$, apresentando uma importância total de 20.2 e custo total de 21. Por conta das incertezas de custo, os requisitos r_1 e r_4 tiveram que ser removidos da solução ótima anterior. Apesar de robusta, tal solução é conservadora pois é improvável que a equipe de desenvolvimento erre as estimativas de custo de todos os requisitos.

O modelo robusto proposto trata esse conservadorismo permitindo uma variação do parâmetro de controle da robustez Γ . Por exemplo, se as estimativas de custo da equipe são historicamente 40% erradas, o nível de robustez seria $\Gamma = 2$, indicando que existe uma expectativa de que dois requisitos terão custos diferentes do que foram originalmente estimados. Nessa situação, a solução ótima seria o subconjunto $\{r_1, r_3, r_4, r_6\}$ com um subconjunto $W = \{r_1, r_3\}$,

que apresenta uma importância de 21 e um custo total de 20.5. Uma vez que é impossível saber antecipadamente quais são os dois requisitos mal estimados, o modelo considera as duas maiores variações de custo, ou seja, \hat{c}_1 and \hat{c}_3 . Dessa forma, durante o desenvolvimento do *release*, se qualquer dois dos quatro requisitos selecionados para o *release* apresentarem custo máximo, a restrição de orçamento ainda será respeitada. Assim, um nível de robustez intermediário permitiu a adição de mais um requisito para o *release*, aumentando a importância total quando comparada com o pior caso ($\Gamma = N$).

O modelo apresentado, juntamente com o parâmetro de controle da robustez Γ , trata as incertezas de custo permitindo que o Engenheiro de Requisitos considere tais incertezas em diferentes níveis. O modelo clássico do NRP pode ser visto como uma abordagem otimista, onde é possível realizar o melhor planejamento do *release* possível. Entretanto, tal solução pode se mostrar inválida por conta das incertezas. A situação de pior caso também pode ser atingida pelo modelo robusto, mas implica em uma perda considerável em qualidade da solução. Um nível de robustez intermediário pode, ao mesmo tempo, garantir uma solução robusta e melhorar a qualidade da solução quando comparada com a abordagem conservadora.

4.3 Conclusões

Este capítulo apresentou a proposta de modelagem robusta para o Problema do Próximo Release. Por conta das incertezas dos valores de importância dos requisitos serem influenciados pelo acontecimento de eventos durante o desenvolvimento do *release*, essa incerteza foi quantificada de forma discreta e probabilística. É definido um conjunto de cenários onde a importância de cada requisito pode variar. A partir da probabilidade de ocorrência de cada cenário, é possível calcular uma importância ponderada dos requisitos, considerando todos os possíveis valores de importância que os requisitos podem assumir.

Pelo fato da incerteza de custo já ser normalmente quantificada utilizando intervalos de variação em outros trabalhos, essa mesma quantificação foi adotada, gerando uma modelagem contínua e determinística para essa incerteza. Além da estimativa tradicional de custo para cada requisito, a equipe de desenvolvimento ainda define uma variação máxima esperada para o custo de cada requisito. A partir da introdução de um parâmetro de controle da robustez, é possível garantir que se tenha uma solução robusta mas que ao mesmo tempo não seja muito conservadora. Vale ressaltar ainda que ambas propostas de tratamento das incertezas de importância e custo dos requisitos são generalizações do modelo tradicional do NRP, sendo possível utilizar essa modelagem clássica facilmente quando necessário.

Um exemplo da utilização do modelo foi utilizado para demonstrar o comportamento do mesmo. Foi mostrado como é possível atingir tanto o NRP clássico como a abordagem mais conservadora. A partir de uma configuração intermediária do parâmetro de controle

da robustez foi mostrado que é possível encontrar uma solução robusta que não perca muito em qualidade da solução.

Assim, a partir do modelo robusto para o NRP apresentado nessa seção, foi possível atingir o objetivo principal desse trabalho, bem como um dos objetivos específicos. Na próxima seção é mostrado o estudo empírico realizado para validação e avaliação da proposta.

5 ESTUDO EMPÍRICO DA PROPOSTA

Este capítulo apresenta o estudo empírico realizado para validar e avaliar o comportamento do modelo robusto para o NRP proposto neste trabalho. O capítulo é dividido em duas seções. A primeira expõe as configurações do estudo empírico, mostrando tanto as instâncias utilizadas como a adaptação de cada técnica de busca ao modelo proposto. A segunda seção exhibe os resultados do estudo e as respectivas análises.

Este estudo empírico foi idealizado com o objetivo de atingir alguns dos objetivos específicos deste trabalho, que são a adaptação de diferentes técnicas de busca para resolução do modelo robusto proposto e a análise do “preço da robustez” desse modelo robusto. Com relação a esse último objetivo específico, o “preço da robustez” representa o quanto é perdido em qualidade da solução quando a solução robusta para uma certa instância é comparada com a solução não-robusta para a mesma instância. A qualidade da solução é representada pelo seu valor de *fitness*, ou valor de função objetivo. Essa perda em qualidade da solução é medida através do “fator de redução” (BERTSIMAS; SIM, 2004), que indica a porcentagem de perda em qualidade devido à robustez.

Considere α_k como o valor de *fitness* de alguma instância quando o nível de robustez é $\Gamma = k \times N$, onde N é o número de requisitos e k representa a porcentagem de requisitos que têm seu custo estimado incorretamente. Naturalmente, os valores que k pode assumir estão no intervalo $0 \leq k \leq 1$. Dessa forma, $k = 0$ resulta em $\Gamma = 0$, representando o NRP tradicional (sem robustez), e $k = 1$ resulta em $\Gamma = N$, representando o pior caso, quando todas as estimativas são consideradas erradas.

O “fator de redução” é então calculado comparando α_k com o valor de *fitness* encontrado pelo modelo não-robusto α_0 . Assim, o “fator de redução” δ_k pode ser calculado do seguinte modo:

$$\delta_k = 100 \times \left(1 - \frac{\alpha_k}{\alpha_0}\right) \quad (5.1)$$

Considere o mesmo exemplo de utilização do modelo dado no capítulo anterior. O valor ótimo de *fitness* do modelo clássico do NRP ($\Gamma = 0$) é $\alpha_0 = 27.5$. Considerando o nível de robustez de 40%, $\Gamma = 0.4N$, o valor ótimo de *fitness* é $\alpha_{0.4} = 21$. A partir da Equação 5.1, é encontrado um fator de redução $\delta_{0.4} = 23.63$. Este valor indica que, para se garantir 40% de robustez na solução, haverá uma perda de 23.63% em qualidade da solução. De forma similar, para $\Gamma = N$, um fator de redução $\delta_1 = 26.64$ é encontrado, representando uma perda de 26.64% em valor de *fitness*.

A seção 5.1 apresenta as configurações do estudo empírico. Mostra a configuração do conjunto de instâncias e a adaptação das técnicas de busca utilizadas na avaliação. Os resultados da avaliação empírica são mostrados na seção 5.2, bem como as respectivas análises.

5.1 Configurações do Estudo Empírico

As configurações das instâncias e das técnicas de busca são apresentadas de forma separada.

5.1.1 Instâncias

O conjunto de instâncias é formado tanto por instâncias artificiais como reais. As instâncias artificiais foram geradas aleatoriamente, seguindo uma distribuição uniforme e projetadas para apresentar diferentes situações de planejamento do próximo *release*, incluindo diferentes números de requisitos, diferentes estratégias de variação de custo e diferentes relações de precedência entre requisitos.

As instâncias artificiais foram geradas com diferentes números de requisitos, representados por $\{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$, respectivamente. Cada instância tem 2, 3 ou 4 cenários. A quantidade de cenários na instância foi definida de forma aleatória. A probabilidade de cada cenário também foi definida aleatoriamente. Os valores v_i^j de importância dos requisitos em cada cenário podem assumir um número inteiro entre 1 e 10. A estimativa de custo c_i também varia de 1 a 10.

Diferentes estratégias para as variações de custo foram consideradas. A variação de custo \hat{c}_i pode assumir um dos seguintes valores $\{10\%, 20\%, 30\%, 40\%, 50\%\}$. Uma variação de custo de 10% indica que cada variação de custo \hat{c}_i é configurada como sendo 10% da respectiva estimativa de custo c_i . Além de variar a porcentagem de 10% a 50% da estimativa original, uma abordagem aleatória também foi aplicada, onde cada variação de custo \hat{c}_i foi independentemente gerada como um número aleatório entre 0 e 50% de c_i .

A densidade de precedência entre requisitos pode assumir um dos seguintes valores $\{0, 10\%, 20\%\}$. A densidade de precedência indica a porcentagem de requisitos que terão precedências. Após escolhidos de forma aleatória quais requisitos terão precedências, devem ser definidos quais serão os requisitos precedentes de cada requisito escolhido. Para cada requisito com precedências, são definidos de forma aleatória de 1% a 5% de outros requisitos como precedentes.

Todos os possíveis aspectos de planejamento do próximo *release* mostrados acima (número de requisitos, estratégia de variação de custo e densidade de precedência) foram combinados, resultando em um conjunto de $10 \times 6 \times 3 = 180$ instâncias artificiais. O nome das

instâncias artificiais está no formato $I_S_R_C_P$, onde R representa o número de requisitos, C indica a estratégia de variação de custo e P representa a densidade de precedência. A instância $I_S_250_20_10$, por exemplo, é artificial, tem 250 requisitos, os valores de \hat{c}_i são configurados para 20% de c_i e apresenta densidade de precedência de 10%.

As instâncias reais foram adaptadas a partir de (XUAN et al., 2012). Nesse trabalho, as instâncias para o NRP foram extraídas de repositórios de bugs de três grandes projetos de código aberto, incluindo Eclipse (ambiente integrado para desenvolvimento Java) (ECLIPSE, 2014) e Mozilla (conjunto de aplicações web) (MOZILLA, 2014).

Um repositório de bugs é um fórum onde usuários (usuários finais, desenvolvedores, testadores etc) podem reportar bugs relacionados ao projeto. Cada bug é então considerado como um requisito. Em tal fórum, um relato de bug pode ser comentado por vários usuários. A importância de cada requisito é então calculada como o número de usuários que comentaram no respectivo relato do bug, o que representa o único cenário possível. Além disso, a gravidade do bug é mapeada para a estimativa de custo do mesmo. Para seguir o mesmo formato das instâncias artificiais, tanto a importância como o custo dos requisitos são normalizadas para o intervalo entre 1 e 10.

Várias instâncias foram extraídas de cada repositório de bug. O número de requisitos das instâncias reais são $\{20, 50, 100, 150, 200\}$, respectivamente. Com relação à variação de custo, as mesmas estratégias utilizadas nas instâncias artificiais foram também aplicadas às instâncias reais. Uma vez que bugs são normalmente independentes entre si, as instâncias reais não apresentam relações de precedência entre requisitos.

Assim, a combinação de todos os possíveis aspectos das instâncias reais, como número de requisitos, estratégia de variação de custo e repositórios de bugs, resultou em um conjunto de $5 \times 6 \times 2 = 60$ instâncias reais para o NRP. O nome das instâncias reais estão no formato $I_Re_P_R_C$, onde P representa o projeto (E para Eclipse e M para Mozilla), R é o número de requisitos e C a estratégia de variação de custo. A instância $I_Re_E_150_50$, por exemplo, foi gerada a partir do repositório de bugs do Eclipse, tem 150 requisitos e a variação de custo \hat{c}_i é configurada para 50% da estimativa de custo c_i .

Considerando tanto as instâncias artificiais como reais, a avaliação empírica foi executada sob 240 instâncias.

5.1.2 Técnicas de Busca

As técnicas de busca Algoritmo Genético, Têmpera Simulada e Busca Aleatória foram consideradas na avaliação empírica, como descritas na seção 2.5.

As configurações dos parâmetros do Algoritmo Genético e da Têmpera Simulada

foram obtidas empiricamente através de um processo de experimentação inspirado por Souza et al. (2011). Primeiramente, 20 das 180 instâncias artificiais foram selecionadas aleatoriamente para participar do processo de configuração. Tanto para o AG como para a TS, diferentes configurações foram consideradas, variando os parâmetros de cada técnica de busca. Para o AG, um total de 27 configurações foram produzidas, variando os valores de probabilidade de cruzamento (60%, 80%, 95%), probabilidade de mutação ($1/N\%$, 0.1%, 1%) e taxa de elitismo (0, 10%, 20%), onde N é o número de requisitos. Para a TS, temperatura inicial (20, 50, 200) e taxa de resfriamento ($1/N\%$, 0.1%, 1%) foram avaliadas, resultando em 9 diferentes configurações. Para todas as instâncias selecionadas, o nível de robustez foi configurado para $\Gamma = 0$. Cada configuração das técnicas de busca foi executada 30 vezes para cada instância, coletando média e desvio padrão do valor de *fitness*. O orçamento do *release* foi configurado para 70% da soma de todas as estimativas de custo. A configuração selecionada para cada técnica foi aquela que obteve os melhores resultados na maioria das 20 instâncias selecionadas.

São mostrados a seguir a adaptação de cada técnica de busca para o modelo robusto do NRP em conjunto com os resultados do processo de configuração dos parâmetros.

Algoritmo Genético. População com N indivíduos. A população inicial é gerada aleatoriamente e composta somente por indivíduos válidos. Se um indivíduo aleatório inicial é inválido, devido à restrição de orçamento ou de precedência, o mesmo é reparado. O operador de reparação consiste em remover requisitos de forma aleatória até que a solução se torne válida. Probabilidade de cruzamento é configurada para 95%, utilizando cruzamento de um ponto. Mutação por bit flip é realizada para cada requisito na solução com uma probabilidade de $1/N\%$. Se necessário, o método de reparação também é aplicado após os operadores de cruzamento e mutação. A implementação do AG emprega elitismo, onde 20% dos melhores indivíduos da população são automaticamente incluídos na próxima geração. O algoritmo retorna o melhor indivíduo após 1000 gerações.

Têmpera Simulada. A solução inicial é gerada aleatoriamente utilizando o mesmo método da população inicial do AG. Temperatura inicial e taxa de resfriamento foram configuradas para 50 e $1/N\%$, respectivamente. A cada iteração, N soluções vizinhas são avaliadas. Solução vizinha é aquela que pode ser produzida a partir da solução corrente com a adição ou remoção de um único requisito. Todas as técnicas de busca foram configuradas para realizar o mesmo número de avaliações de *fitness*. Assim, a temperatura final é dinamicamente calculada para permitir somente $1000 \times N$ avaliações de *fitness*.

Busca Aleatória. O algoritmo de busca aleatória consiste em gerar $1000 \times N$ soluções aleatórias. O algoritmo retorna a melhor solução ao final. A geração da solução aleatória é a mesma como na população inicial do AG.

Cada técnica de busca foi executada para todas as 240 instâncias, com diferentes níveis de robustez para cada instância. O parâmetro de controle Γ foi configurado para

$\{0, 0.05N, 0.1N, 0.15N, \dots, N\}$. Para cada instância e cada nível de robustez, cada algoritmo foi executado 30 vezes, obtendo média e desvio padrão do valor de *fitness*. O orçamento do *release* foi novamente configurado para 70% do custo total. Considerando todas as técnicas de busca, todas as instâncias e todos os níveis de robustez, um total de 15120 execuções foram realizadas.

Com o objetivo de permitir uma total replicação de todo o estudo empírico realizado, todas as instâncias artificiais e reais, juntamente com o código fonte utilizado, estão disponíveis na página <http://goes.uece.br/mhepaixao/robustNRP/>. Também contém todos os resultados do estudo empírico, inclusive os que tiveram que ser omitidos por restrições de espaço.

5.2 Resultados e Análises

Os resultados do estudo empírico são apresentados nesta seção. Como mostrado na seção 1.2, para uma melhor análise do “preço da robustez”, a mesma será realizada em partes, tratadas separadamente a cada subseção.

5.2.1 Preço da Robustez para diferentes níveis de robustez

Essa análise consiste em avaliar o quanto é perdido na qualidade da solução com a variação do parâmetro Γ . Para se realizar uma avaliação ‘justa’, devem ser consideradas instâncias com a mesma estratégia de variação de custo e mesma densidade de precedência. As diferentes estratégias para variação do custo e as diferentes densidades de precedência são avaliadas nas seções 5.2.2 e 5.2.3, respectivamente. Dessa forma, essa análise é baseada somente nas instâncias com variação de custo de 10% e sem precedência. Uma vez que as instâncias reais são abordadas na seção 5.2.4, somente as instâncias artificiais foram utilizadas nesta análise.

A Tabela 1 apresenta os resultados computados pelo Algoritmo Genético (AG), Têmpera Simulada (TS) e Busca Aleatória (BA) para as instâncias artificiais com variação de custo de 10% e sem relações de precedência. Alguns níveis de robustez são omitidos por motivo de espaço. Os melhores resultados encontrados para cada instância e cada nível de robustez são destacados.

Como pode ser visto a partir da tabela, com o aumento do parâmetro Γ , ou seja, com o aumento da robustez, o valor de *fitness* tende a diminuir. Esse comportamento é o mesmo para todas as instâncias e todas as técnicas de busca utilizadas. Como o valor de *fitness* é dado pela soma das importâncias dos requisitos incluídos no *release*, uma diminuição em *fitness* representa menos requisitos a serem implementados. Com relação à qualidade da solução, o AG apresentou os melhores resultados para todas as instâncias e todos os níveis de robustez. O AG também apresentou o menor desvio padrão. Novamente a partir de uma visão de Engenharia

Tabela 1 – Valores de *fitness* para as instâncias artificiais com variação de custo de 10% e sem precedência entre requisitos

Instância	TB	Γ						
		0	0.1N	0.3N	0.5N	0.7N	0.9N	N
I_S_50_10_0	AG	247.18 ± 2.37	244.34 ± 2.42	239.14 ± 1.47	235.41 ± 2.31	233.38 ± 2.52	234.28 ± 2.31	233.9 ± 2.21
	TS	221.37 ± 3.77	219.21 ± 3.45	213.80 ± 3.54	211.91 ± 3.52	210.81 ± 3.31	209.94 ± 3.07	210.9 ± 2.83
	BA	216.59 ± 4.46	213.95 ± 3.36	211.53 ± 3.83	208.71 ± 3.15	207.58 ± 4.12	207.25 ± 3.46	208.3 ± 3.89
I_S_100_10_0	AG	442.02 ± 1.77	436.79 ± 1.24	427.41 ± 2.21	422.09 ± 2.07	421.10 ± 1.91	419.59 ± 2.06	420.19 ± 1.72
	TS	374.03 ± 7.14	370.97 ± 5.05	362.23 ± 5.35	359.38 ± 5.30	358.05 ± 6.05	356.76 ± 4.36	359.77 ± 5.48
	BA	365.71 ± 6.66	363.56 ± 6.56	357.52 ± 4.93	355.83 ± 4.55	355.21 ± 4.52	355.57 ± 4.37	355.82 ± 6.00
I_S_150_10_0	AG	726.53 ± 2.60	718.86 ± 2.54	709.97 ± 2.09	703.17 ± 2.46	698.73 ± 3.39	699.14 ± 2.40	699.03 ± 2.76
	TS	571.66 ± 14.79	565.89 ± 18.9	555.91 ± 12.70	545.41 ± 11.7	545.45 ± 8.08	542.24 ± 8.63	546.08 ± 7.68
	BA	548.74 ± 10.04	545.02 ± 9.33	545.89 ± 10.64	541.67 ± 6.55	541.31 ± 7.76	538.73 ± 6.71	545.00 ± 9.63
I_S_200_10_0	AG	954.89 ± 1.41	942.07 ± 2.64	924.52 ± 1.92	911.59 ± 2.71	905.24 ± 2.72	905.23 ± 1.84	904.86 ± 2.43
	TS	786.23 ± 18.64	764.01 ± 21.80	739.73 ± 18.62	725.13 ± 12.71	723.47 ± 12.81	725.20 ± 11.12	723.95 ± 13.14
	BA	720.08 ± 11.59	722.52 ± 10.68	719.31 ± 10.98	717.45 ± 9.83	715.73 ± 8.32	716.09 ± 10.54	714.66 ± 7.48
I_S_250_10_0	AG	1143.95 ± 2.83	1129.95 ± 3.21	1108.57 ± 2.99	1094.12 ± 3.34	1088.55 ± 2.97	1087.74 ± 2.51	1087.15 ± 3.12
	TS	941.78 ± 24.00	915.15 ± 26.80	881.67 ± 21.36	867.41 ± 22.0	862.25 ± 18.84	857.91 ± 17.76	869.01 ± 16.08
	BA	849.4 ± 10.87	848.74 ± 10.67	846.99 ± 9.67	848.46 ± 9.07	849.3 ± 9.47	847.99 ± 10.86	848.58 ± 11.38
I_S_300_10_0	AG	1469.1 ± 3.82	1453.43 ± 3.16	1427.47 ± 4.91	1412.69 ± 2.78	1406.98 ± 3.76	1405.29 ± 2.91	1405.84 ± 3.71
	TS	1162.39 ± 24.26	1129.45 ± 35.45	1090.82 ± 39.95	1069.84 ± 25.68	1055.64 ± 19.3	1060.28 ± 28.04	1063.2 ± 20.69
	BA	1037.7 ± 15.38	1035.03 ± 11.91	1032.42 ± 12.49	1038.18 ± 14.54	1032.91 ± 11.72	1034.29 ± 12.77	1036.44 ± 10.15
I_S_350_10_0	AG	1674.29 ± 3.26	1654.59 ± 3.00	1624.21 ± 3.28	1605.46 ± 4.25	1596.94 ± 4.54	1596.16 ± 3.71	1596.66 ± 3.25
	TS	1340.63 ± 24.98	1316.59 ± 26.96	1249.19 ± 22.7	1229.91 ± 26.63	1223.46 ± 22.15	1226.32 ± 30.14	1229.05 ± 29.26
	BA	1190.15 ± 17.95	1189.73 ± 15.16	1194.7 ± 16.25	1190.77 ± 14.23	1197.38 ± 17.01	1190.05 ± 15.23	1195.32 ± 14.39
I_S_400_10_0	AG	1889.65 ± 2.97	1866.14 ± 3.77	1831.72 ± 4.71	1812.2 ± 4.72	1804.9 ± 4.25	1804.19 ± 5.02	1803.19 ± 5.11
	TS	1487.35 ± 34.28	1459.83 ± 34.11	1384.28 ± 24.31	1370.63 ± 39.38	1364.46 ± 29.07	1359.43 ± 33.8	1356.56 ± 34.57
	BA	1313.27 ± 19.02	1311.23 ± 15.39	1304.73 ± 15.68	1309.03 ± 17.98	1302.14 ± 12.27	1308.73 ± 16.42	1303.68 ± 16.13
I_S_450_10_0	AG	2141.9 ± 5.03	2113.57 ± 4.30	2071.92 ± 5.36	2045.75 ± 5.22	2031.64 ± 5.39	2031.18 ± 5.62	2030.51 ± 5.81
	TS	1758.86 ± 25.95	1705.22 ± 32.22	1645.25 ± 32.19	1608.39 ± 32.09	1610.41 ± 27.25	1603.45 ± 29.08	1605.75 ± 22.72
	BA	1310.17 ± 16.28	1311.23 ± 15.39	1304.73 ± 15.68	1309.03 ± 17.98	1302.14 ± 12.27	1308.73 ± 16.42	1303.68 ± 16.13
I_S_500_10_0	AG	2385.36 ± 3.10	2356.31 ± 4.45	2310.83 ± 4.65	2283.37 ± 4.28	2271.31 ± 5.99	2270.22 ± 4.47	2268.79 ± 4.87
	TS	1920.01 ± 34.36	1871.46 ± 30.91	1794.33 ± 38.5	1760.09 ± 35.33	1749.81 ± 30.49	1748.15 ± 30.9	1744.9 ± 39.74
	BA	1643.72 ± 11.84	1646.83 ± 21.31	1647.24 ± 16.07	1647.95 ± 21.04	1647.01 ± 15.94	1647.96 ± 20.51	1647.55 ± 21.66

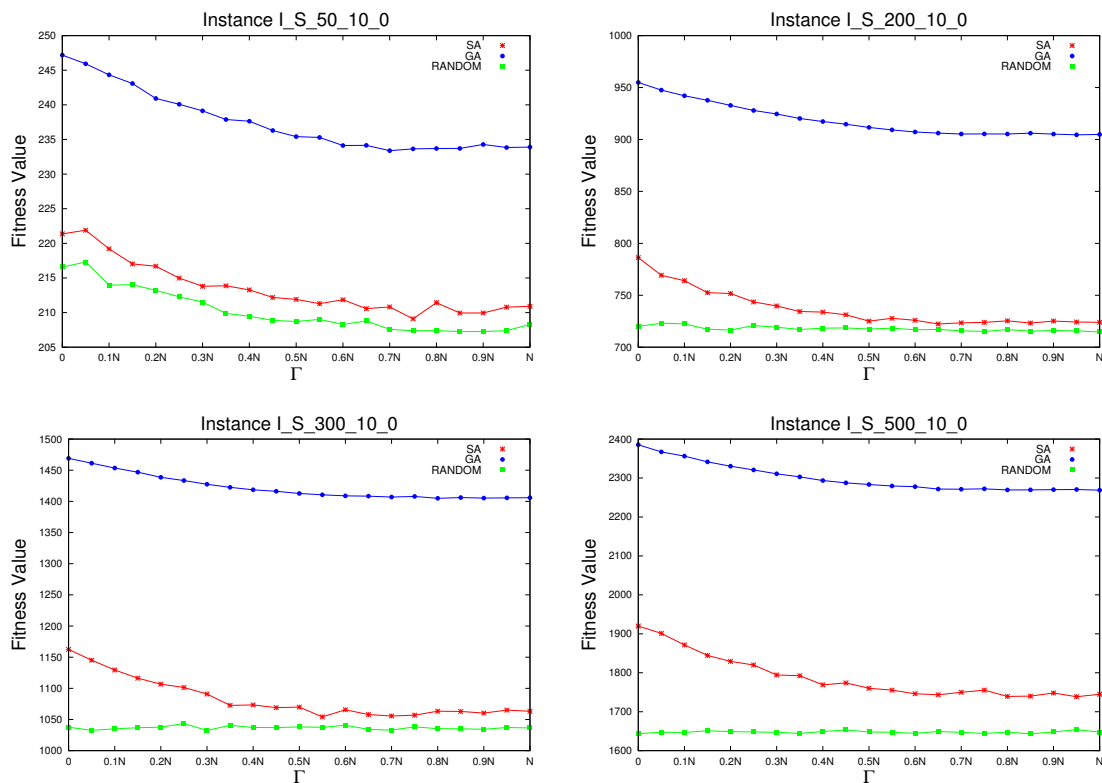


Figura 8 – Valores de *fitness* encontrados pelo AG, TS e BA para algumas instâncias artificiais com variação de custo de 10% e sem relações de precedência. Fonte: Autor.

de software, o AG, por apresentar melhores valores de *fitness* em comparação à TS e BA, foi capaz de incluir mais requisitos no *release*. Por apresentar uma menor desvio padrão, pode-se dizer que o AG se mostrou mais coerente na seleção do *release* ao longo das várias execuções.

Como já era esperado, o algoritmo de BA não conseguiu encontrar boas soluções para o modelo robusto proposto, sendo superado em qualidade da solução tanto pelo AG como pela TS para todas as instâncias e todos os níveis de robustez. Dessa forma, a proposta passa pelo teste de sanidade ao demonstrar que de fato são necessários algoritmos mais sofisticados para se encontrar boas soluções.

A Figura 8 apresenta os valores de *fitness* das três técnicas de busca para algumas instâncias artificiais. Como indicado acima, o valor de *fitness* diminui claramente com o aumento do nível de robustez. Além disso, também é visível o fato do AG ser capaz de encontrar os melhores resultados. Os resultados encontrados para as quatro instâncias mostradas são parecidos, bem como são similares os resultados para as demais instâncias não mostradas na figura.

Os fatores de redução para as instâncias artificiais com variação de custo de 10% e sem precedências entre requisitos são apresentados na Tabela 2. Como pode ser visto nessa tabela, a perda em *fitness* devido a robustez é consideravelmente pequena. Considerando todas

as instâncias, o AG pode atingir 10% de robustez perdendo 1.2% em qualidade da solução, em média. Para maiores níveis de robustez, como 50% e 100%, a perda média do AG é de 4.22% e 4.79%, respectivamente. Em outras palavras, utilizando o AG, é possível garantir 100% de robustez com uma pequena perda de 4.79% em qualidade da solução. Para a TS, o fator de redução para $\Gamma = 0.5N$ e $\Gamma = N$ é de 6.94% e 7.21%, respectivamente. A BA apresentou pequenos fatores de redução, mas devido ao fato do algoritmo não ter sido capaz de encontrar boas soluções, tais resultados não refletem o comportamento do modelo robusto proposto.

Tabela 2 – Fatores de redução para as instâncias artificiais com 10% de variação de custo e sem precedência entre requisitos

Instância	TB	Γ						
		0.05N	0.1N	0.3N	0.5N	0.7N	0.9N	N
I_S_50_10_0	AG	0.51	1.15	3.25	4.76	5.58	5.22	5.37
	TS	0.23	0.98	3.42	4.27	4.77	5.16	4.73
	BA	0.33	1.22	2.34	3.64	4.16	4.31	3.83
I_S_100_10_0	AG	0.74	1.18	3.31	4.51	4.73	5.07	4.94
	TS	0.82	0.82	3.15	3.92	4.27	4.62	3.81
	BA	0.3	0.59	2.24	2.7	2.87	2.77	2.7
I_S_150_10_0	AG	0.58	1.06	2.28	3.22	3.83	3.77	3.79
	TS	0.54	1.01	2.76	4.59	4.58	5.15	4.47
	BA	0.71	0.68	0.52	1.29	1.35	1.82	0.68
I_S_200_10_0	AG	0.77	1.34	3.18	4.53	5.2	5.2	5.24
	TS	2.16	2.83	5.91	7.77	7.98	7.76	7.92
	BA	0.42	0.34	0.11	0.37	0.6	0.55	0.75
I_S_250_10_0	AG	0.65	1.22	3.09	4.36	4.84	4.91	4.97
	TS	1.4	2.83	6.38	7.9	8.44	8.91	7.73
	BA	0.26	0.08	0.28	0.11	0.01	0.17	0.1
I_S_300_10_0	AG	0.53	1.07	2.83	3.84	4.23	4.34	4.31
	TS	1.48	2.83	6.16	7.96	9.18	8.78	8.53
	BA	0.49	0.26	0.51	0.05	0.46	0.33	0.12
I_S_350_10_0	AG	0.57	1.18	2.99	4.11	4.62	4.67	4.64
	TS	1.02	1.79	6.82	8.26	8.74	8.53	8.32
	BA	0.33	0.04	0.38	0.05	0.61	0.01	0.43
I_S_400_10_0	AG	0.64	1.24	3.07	4.1	4.48	4.52	4.58
	TS	1.28	1.85	6.93	7.85	8.26	8.6	8.79
	BA	0.24	0.08	0.42	0.09	0.61	0.11	0.5
I_S_450_10_0	AG	0.69	1.32	3.27	4.49	5.15	5.17	5.2
	TS	0.86	3.05	6.46	8.55	8.44	8.84	8.71
	BA	0.13	0.16	0.19	0.19	0.28	0.21	0.23
I_S_500_10_0	AG	0.77	1.22	3.12	4.28	4.78	4.83	4.89
	TS	0.99	2.53	6.55	8.33	8.86	8.95	9.12
	BA	0.21	0.19	0.21	0.26	0.2	0.26	0.23

Os fatores de redução encontrados pelo AG e pela TS ao longo do crescimento de Γ para algumas instâncias artificiais podem ser vistos na Figura 9. Como já mostrado e discutido anteriormente, a perda em qualidade da solução com o crescimento do nível de robustez é claro. Vale destacar os fatores de redução encontrados pelo AG, que não ultrapassam 5% em nenhuma das instâncias mostradas na figura. O AG demonstra mais uma vez sua qualidade ao encontrar soluções com maior valor de *fitness* e menor fator de redução.

Uma suposição comum é que um diferente nível de robustez leva a um diferente fator de redução. Para pequenos níveis de robustez, a diferença no fator de redução aparenta ser de fato significativa. No entanto, depois que Γ assume o valor de metade do número de requisitos ($\Gamma = 0.5N$), o fator de redução parece se estabilizar, como também pode ser visto nas Figuras 8 e 9.

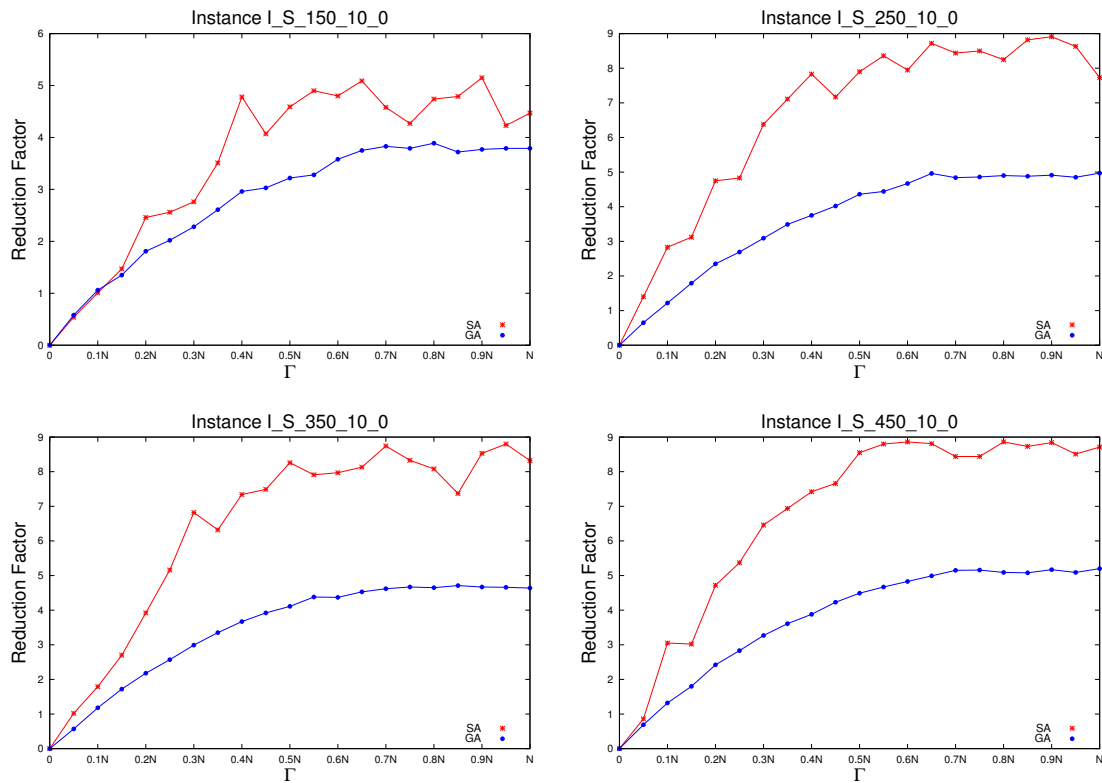


Figura 9 – Fatores de redução encontrados pelo AG e TS para algumas instâncias artificiais com variação de custo de 10% e sem relações de precedência. Fonte: Autor.

A fim de avaliar esse comportamento, várias amostras foram compostas contendo os fatores de redução de todas as instâncias para um certo nível de robustez e técnica de busca. Considerando o AG, por exemplo, a amostra contendo os fatores de redução para $\Gamma = 0.05N$ é $\{0.51, 0.74, 0.58, 0.77, 0.65, 0.53, 0.57, 0.64, 0.69, 0.77\}$, enquanto a amostra para $\Gamma = 0.1N$ é $\{1.15, 1.18, 1.06, 1.34, 1.22, 1.07, 1.18, 1.24, 1.32, 1.22\}$. A partir dessas amostras, é possível utilizar testes estatísticos para atestar a igualdade ou diferença estatística entre duas amostras, avaliando assim, a influência do nível de robustez no “preço da robustez”.

A partir de testes de normalidade, atestou-se que as amostras não seguem uma distribuição normal. Dessa forma, como mostrado na seção 2.6, o teste Wilcoxon da ferramenta de computação estatística *R* foi utilizado para comparar diferentes amostras e determinar se elas são estatisticamente diferentes. Tal teste recebe duas amostras como entrada e retorna um certo *p-value* como resultado. Esse *p-value* pode ser entendido como a probabilidade das amostras serem na verdade iguais. Nesse trabalho, duas amostras são consideradas estatisticamente diferentes quando o teste resulta em um $p\text{-value} \leq 0.05$, o que representa um nível de significância de 95%. Para essa análise em particular, considere $p_{a,b}^{TB}$ como sendo o *p-value* para a comparação entre a amostra com nível de robustez *a* e a amostra com nível de robustez *b*, para a técnica de busca *TB*.

Considerando o AG, para baixos níveis de robustez, como $\Gamma = 0.05N$ e $\Gamma = 0.1N$,

o *p-value* é $p_{0,05,0.1}^{AG} = 1.79 \times 10^{-3}$. Esse valor indica que a probabilidade das amostras serem mesmo diferentes é maior que 99%. Em outras palavras, é quase certo que um aumento no nível da robustez de $\Gamma = 0.05N$ para $\Gamma = 0.1N$ leva a uma mudança significativa na qualidade da solução. Para outros pequenos valores de Γ , esse comportamento continua: $p_{0.1,0.15}^{AG} = 1.78 \times 10^{-3}$, $p_{0.15,0.2}^{AG} = 3.24 \times 10^{-3}$, $p_{0.2,0.25}^{AG} = 4.0 \times 10^{-2}$. Entretanto, com o aumento do nível da robustez, as diferenças entre amostras tonam-se estatisticamente menos óbvias. Para $\Gamma = 0.5N$ e $\Gamma = 0.55N$, por exemplo, o teste resulta em $p_{0.5,0.55}^{AG} = 0.40$, o que indica uma probabilidade de 40% das amostras serem iguais. Para maiores valores de Γ , os *p-values* são ainda maiores: $p_{0.9,0.95}^{AG} = 0.96$ e $p_{0.95,1}^{AG} = 1$. Tais resultados indicam que, para níveis de robustez elevados, pequenas mudanças no parâmetro Γ não influenciam no “preço da robustez”.

Para a TS os resultados são similares. Para pequenos valores de Γ , os *p-values* são $p_{0.05,0.1}^{TS} = 0.028$, $p_{0.1,0.15}^{TS} = 0.053$ e $p_{0.15,0.2}^{TS} = 0.075$. Para níveis de robustez maiores, os *p-values* são $p_{0.85,0.9}^{TS} = 0.59$, $p_{0.9,0.95}^{TS} = 0.79$ e $p_{0.95,1}^{TS} = 0.63$. Como pode ser visto, o mesmo comportamento é encontrado pela TS. Para níveis de robustez menores, um pequeno aumento no Γ conduz a uma mudança estatisticamente significativa no valor de *fitness*. Para maiores níveis de robustez, um aumento em Γ não influencia o fator de redução.

Um fato interessante é que o fator de redução aparenta ser similar entre diferentes instâncias. O teste de Wilcoxon foi também utilizado para avaliar tal observação. Neste caso, a amostra consiste nos fatores de redução de uma única instância para todos os níveis de robustez, incluindo os que não são mostrados na Tabela 2. Para esta análise, considere $p_{a,b}^{TB}$ como sendo o *p-value* para a comparação entre a amostra com número de requisitos a e a amostra com número de requisitos b , para a técnica de busca *TB*.

Considerando os resultados do AG, para as instâncias com 200 e 450 requisitos, por exemplo, o *p-value* calculado é $p_{200,450}^{AG} = 0.56$, o que fortemente indica nenhuma diferença entre os resultados. Por outro lado, para as instâncias com 50 e 150 requisitos, o *p-value* é $p_{50,150}^{AG} = 0.01$, o que estatisticamente determina uma diferença entre os fatores de redução das duas instâncias. Assim, para comparar o “preço da robustez” entre instâncias com diferentes números de requisitos, o teste foi executado para todos os possíveis pares de tais instâncias.

Como resultado, 36 dos 45 testes apresentaram um *p-value* menor que 0.05. Logo, pode-se afirmar que existem evidências empíricas e estatísticas de que o “preço da robustez” do modelo proposto é independente do número de requisitos na instância.

A análise realizada nesta subseção está relacionada com o “preço da robustez” do modelo proposto e a sua variação com as mudanças no nível da robustez. A avaliação apresentada acima mostra que, com o aumento do nível da robustez, o valor de *fitness* tende a diminuir. Entretanto, essa perda em qualidade da solução só é estatisticamente significativa para pequenos valores de Γ . A partir de $\Gamma = 0.5N$, não existe evidência estatística para comprovar que um maior nível de robustez leva a uma maior perda em qualidade da solução. Em outras palavras,

garantir um nível de robustez de 50% não é muito diferente de garantir um nível de robustez de 55% ou mais. Além disso, com relação ao “preço da robustez”, o modelo proposto apresenta-se praticamente da mesma forma para instâncias com diferentes números de requisitos. Encorajado por essa estabilidade, é possível dizer que um alto nível de robustez pode ser atingido perdendo uma porção consideravelmente pequena de *fitness*. Considerando o AG, uma robustez de 100% pode ser obtida através de uma perda média de somente 4.79%.

Uma vez que o AG encontrou as melhores soluções e apresentou o menor desvio padrão para todas as instâncias e todos os níveis de robustez, será a única técnica de busca considerada para as análises posteriores.

5.2.2 Preço da Robustez para diferentes variações no custo dos requisitos

Como dito anteriormente, diferentes equipes de desenvolvimento apresentam habilidades de estimativa distintas. Esta análise está relacionada com o “preço da robustez” quando diferentes estratégias para a variação de custo são utilizadas. A Tabela 3 apresenta os fatores de redução computados pelo AG para diferentes estratégias de variação de custo e diferentes níveis de robustez. As diferentes variações de custo (VC) consideradas foram: 10%, 20%, 30%, 40%, 50% e aleatória (AL), como apresentado nas configurações do estudo empírico. Precedências entre requisitos não foram consideradas. O resultado de algumas instâncias foi omitido devido a restrições de espaço.

Como pode ser notado, e como esperado, considerando uma única instância, quanto maior a variação do custo, maior é o fator de redução. Contudo, este crescimento não é linear. Para a instância com 300 requisitos, por exemplo, com uma variação de custo de 10%, um nível de robustez de 30% resulta em um fator de redução de 2.83%. Ao mudar a variação de custo para 30% e 50%, a perda em *fitness* cresce para 8.39% e 13.77%, respectivamente. Uma variação de custo de 50% é consideravelmente alta e mesmo assim, é possível atingir altos níveis de robustez com uma pequena perda em qualidade da solução. Considerando todas as instâncias e configurando a variação esperada para metade da estimativa original do custo ($\hat{c}_i = 50\%$), um nível de robustez de 100% pode ser atingido perdendo em torno de 19% em valor de *fitness*.

A Figura 10 apresenta os fatores de redução do AG para algumas instâncias artificiais não mostradas na Tabela 3. Como pode ser visto, o fator de redução visivelmente aumenta com o aumento da variação de custo. Também é notável a similaridade nos resultados de todas as instâncias.

Considerando qualquer instância na Figura 10, os fatores de redução para as diferentes estratégias de variação de custo aparentam ser completamente diferentes um do outro. Para avaliar esse comportamento, as amostras são agora formadas pelos fatores de redução de

Tabela 3 – Fatores de redução encontrados pelo AG para instâncias artificiais com diferentes variações de custo em sem relações de precedência

Instância	TB	Γ						
		$0.05N$	$0.1N$	$0.3N$	$0.5N$	$0.7N$	$0.9N$	N
I_S_100_VC_0	10%	0.74	1.18	3.31	4.51	4.73	5.07	4.94
	20%	1.31	2.53	6.31	8.37	9.03	9.16	9.11
	30%	2.04	4.01	9.31	12.14	13.0	12.84	13.01
	40%	2.53	5.09	12.13	15.3	16.2	16.02	15.94
	50%	3.3	6.35	15.19	18.45	19.19	18.93	18.98
	AL	2.8	5.09	9.02	10.17	10.44	10.39	10.41
I_S_200_VC_0	10%	0.77	1.34	3.18	4.53	5.2	5.2	5.24
	20%	1.34	2.47	6.37	8.85	9.85	9.85	9.84
	30%	1.99	3.84	9.33	12.84	13.87	13.83	13.81
	40%	2.8	5.13	12.31	16.42	17.48	17.39	17.55
	50%	3.41	6.26	15.24	19.78	20.73	20.77	20.75
	AL	2.33	4.09	8.83	10.79	11.28	11.38	11.29
I_S_300_VC_0	10%	0.53	1.07	2.83	3.84	4.23	4.34	4.31
	20%	1.18	2.3	5.69	7.55	8.21	8.3	8.35
	30%	1.85	3.44	8.39	11.02	11.87	11.95	11.84
	40%	2.35	4.59	11.1	14.15	15.06	15.12	14.89
	50%	3.03	5.74	13.77	17.28	18.11	17.98	17.98
	AL	2.38	4.16	7.99	9.51	10.03	9.95	9.88
I_S_400_VC_0	10%	0.64	1.24	3.07	4.1	4.48	4.52	4.58
	20%	1.23	2.33	5.92	7.81	8.59	8.53	8.51
	30%	1.91	3.66	8.93	11.51	12.31	12.19	12.32
	40%	2.57	4.85	11.68	14.79	15.67	15.7	15.71
	50%	3.19	6.16	14.31	18.01	18.6	18.6	18.65
	AL	2.33	4.1	8.03	9.87	10.27	10.15	10.24
I_S_500_VC_0	10%	0.77	1.22	3.12	4.28	4.78	4.83	4.89
	20%	1.28	2.39	6.08	8.18	9.02	9.01	9.03
	30%	1.83	3.7	9.0	11.8	12.83	12.91	12.78
	40%	2.59	4.87	11.98	15.38	16.39	16.31	16.31
	50%	3.22	6.04	14.62	18.56	19.49	19.54	19.41
	AL	2.47	4.28	8.29	9.94	10.46	10.41	10.53

uma certa estratégia de variação de custo para uma certa instância. Considere $p_{a,b}^{NR}$ como sendo o p -value para a comparação entre a amostra com variação de custo de $a\%$ e a amostra com variação de custo de $b\%$, para a instância com número de requisitos NR .

Para a instância com 250 requisitos, por exemplo, os resultados são: $p_{10,20}^{250} = 3 \times 10^{-3}$, $p_{20,30}^{250} = 3.3 \times 10^{-2}$, $p_{30,40}^{250} = 0.012$, $p_{40,50}^{250} = 0.017$ e $p_{50,AL}^{250} = 4.9 \times 10^{-3}$. Como pode ser visto, os fatores de redução para diferentes variações de custo são realmente diferentes um do outro. Esses resultados comprovam estatisticamente que uma mudança na estratégia de variação de custo leva a um “preço da robustez” diferente. Para todas as outras instâncias os resultados são similares.

Foi mostrado na subseção anterior que, para uma variação de custo de 10% e sem relações de precedência, o “preço da robustez” só é estatisticamente influenciado pelo nível da robustez para pequenos valores de Γ . Quando todas as possíveis estratégias de variação de custo são consideradas esse comportamento continua o mesmo. Para essa avaliação, as amostras são formadas pelos fatores de redução de todas as instâncias e todas as variações de custo para um certo nível de robustez. Relações de precedência não são consideradas. Para o AG, os p -values para pequenos valores de Γ são $p_{0.05,0.1}^{AG} = 5.32 \times 10^{-9}$, $p_{0.1,0.15}^{AG} = 3.33 \times 10^{-3}$ e $p_{0.15,0.2}^{AG} = 8.09 \times 10^{-2}$. Para maiores níveis de robustez, os p -values são $p_{0.5,0.55}^{AG} = 0.62$,

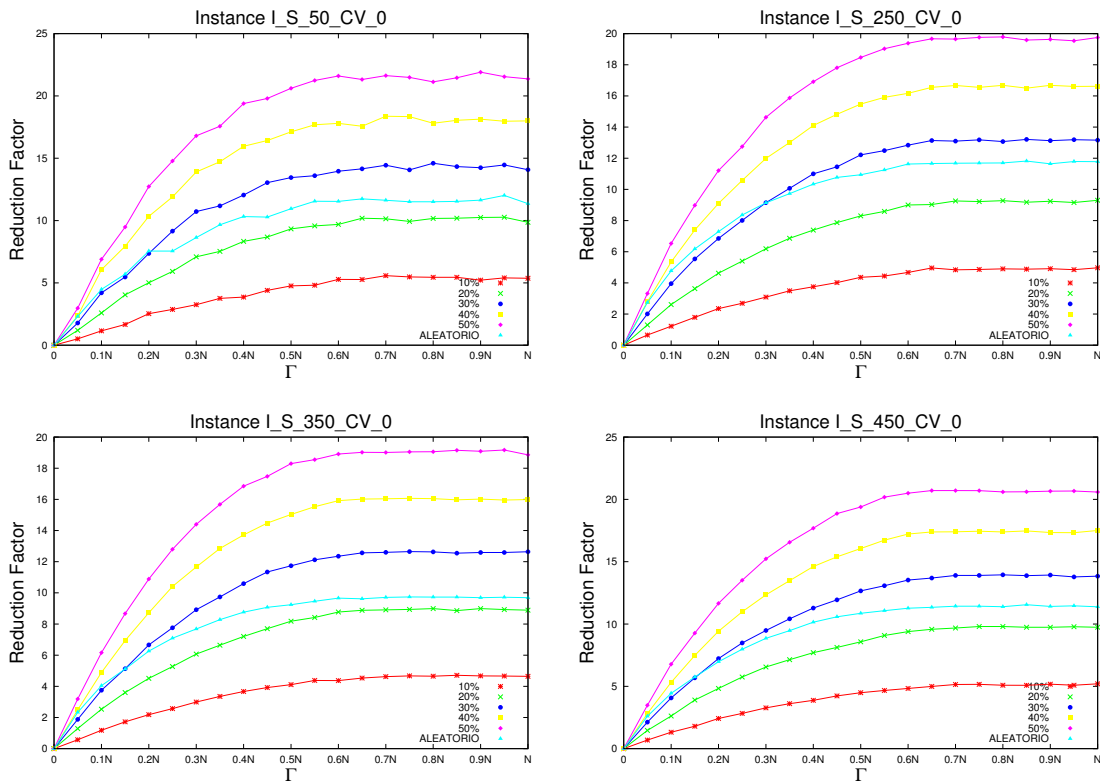


Figura 10 – Fatores de redução do AG para algumas instâncias artificiais com diferentes variações de custo e sem relações de precedência. Fonte: Autor.

$p_{0,9,0,95}^{AG} = 0.97$ e $p_{0,95,1}^{AG} = 0.96$. Desse modo, o comportamento do “preço da robustez” em ficar praticamente constante a partir de $\Gamma = 0.5N$ é comprovado estatisticamente para todas as estratégias de variação.

A estratégia de variação de custo exerce um importante papel no “preço da robustez”. É estatisticamente comprovado que diferentes variações de custo levam a diferentes fatores de redução, de forma que quanto maior a variação, maior é a redução. Contudo, esse aumento no “preço da robustez” não é linear. Mesmo com altas variações de custo, como 50%, é possível garantir uma robustez de 100% perdendo, em média, somente 19% em qualidade da solução. Também é comprovado estatisticamente, para todas as variações de custo analisadas, que uma mudança no nível da robustez só é significativa para pequenos valores de Γ . Para maiores níveis de robustez, o “preço da robustez” pode ser considerado o mesmo.

5.2.3 Preço da Robustez para instâncias com interdependências entre requisitos

Em uma situação de planejamento do próximo *release*, os requisitos podem ter relações de precedência entre si. Particularmente para o NRP, se um requisito r_i é um precedente do requisito r_j , deve-se incluir r_i no *release* para também se incluir r_j . No entanto, r_i pode ser incluído no *release* independentemente de r_j . A densidade de precedência (DP) representa a

porcentagem de requisitos que têm precedentes na instância, como apresentado nas configurações do estudo empírico.

Neste estudo empírico, as diferentes densidades de precedência consideradas foram: 0, 10% e 20%. Uma vez que as diferentes estratégias para variação do custo foram abordadas na seção anterior, essa análise considera somente instâncias com variação de custo de 10%. A Tabela 4 apresenta os fatores de redução do AG para as instâncias com variação de custo de 10% e diferentes densidades de precedência.

Tabela 4 – Fatores de redução encontrados pelo AG para as instâncias artificiais com variação de custo de 10% e diferentes densidades de precedência

Instância	TB	Γ						
		0.05N	0.1N	0.3N	0.5N	0.7N	0.9N	N
I_S_50_10_DP	0	0.51	1.15	3.25	4.76	5.58	5.22	5.37
	10%	0.53	0.85	3.62	4.6	5.18	5.27	5.27
	20%	0.55	1.39	3.57	5.29	5.57	5.87	5.76
I_S_100_10_DP	0	0.74	1.18	3.31	4.51	4.73	5.07	4.94
	10%	0.54	1.15	3.1	4.43	4.79	4.78	4.92
	20%	0.78	1.33	3.36	4.67	4.82	4.93	5.13
I_S_150_10_DP	0	0.58	1.06	2.28	3.22	3.83	3.77	3.79
	10%	0.47	1.08	2.3	3.26	3.62	3.94	3.9
	20%	0.47	0.89	2.29	3.15	3.63	3.66	3.65
I_S_200_10_DP	0	0.77	1.34	3.18	4.53	5.2	5.2	5.24
	10%	0.67	1.23	3.22	4.49	5.2	5.13	5.14
	20%	0.7	1.27	3.22	4.49	5.11	5.19	5.19
I_S_250_10_DP	0	0.65	1.22	3.09	4.36	4.84	4.91	4.97
	10%	0.76	1.36	3.13	4.41	4.93	5.0	4.97
	20%	0.66	1.23	3.1	4.32	4.84	4.9	4.92
I_S_300_10_DP	0	0.53	1.07	2.83	3.84	4.23	4.34	4.31
	10%	0.29	1.02	2.83	3.82	4.07	4.24	4.0
	20%	0.54	1.07	2.73	3.82	4.28	4.33	4.36
I_S_350_10_DP	0	0.57	1.18	2.99	4.11	4.62	4.67	4.64
	10%	0.69	1.21	3.05	4.14	4.73	4.69	4.69
	20%	0.52	1.16	3.19	4.3	4.82	4.57	4.72
I_S_400_10_DP	0	0.64	1.24	3.07	4.1	4.48	4.52	4.58
	10%	0.32	1.08	2.8	3.6	4.11	4.08	4.39
	20%	0.59	1.22	2.87	3.95	4.46	4.44	4.48
I_S_450_10_DP	0	0.69	1.32	3.27	4.49	5.15	5.17	5.2
	10%	0.75	1.38	3.35	4.51	5.11	5.28	5.21
	20%	0.57	1.61	3.65	4.4	4.88	4.97	4.96
I_S_500_10_DP	0	0.77	1.22	3.12	4.28	4.78	4.83	4.89
	10%	0.49	0.91	2.98	4.16	4.53	4.8	4.36
	20%	0.14	0.05	1.38	2.56	2.58	2.73	2.66

Para uma densidade de precedência de 10%, por exemplo, atinge-se uma robustez de 30% perdendo 3.03% em valor de *fitness*, em média. Com relação à densidade de precedência de 20%, 70% de robustez resulta em uma perda média de 4.5% em qualidade da solução. Quando as relações de precedência entre requisitos são consideradas, o “preço da robustez” ainda é consideravelmente pequeno.

Alguns dos resultados visto na Tabela 4 são mostrados graficamente na Figura 11. Para todas as densidades de precedência mostradas o fator de redução se encontra em torno de 5%. De forma similar às demais análises, os resultados são visivelmente parecidos para todas as instâncias mostradas.

Como pode ser visto, considerando qualquer uma das instâncias apresentadas na

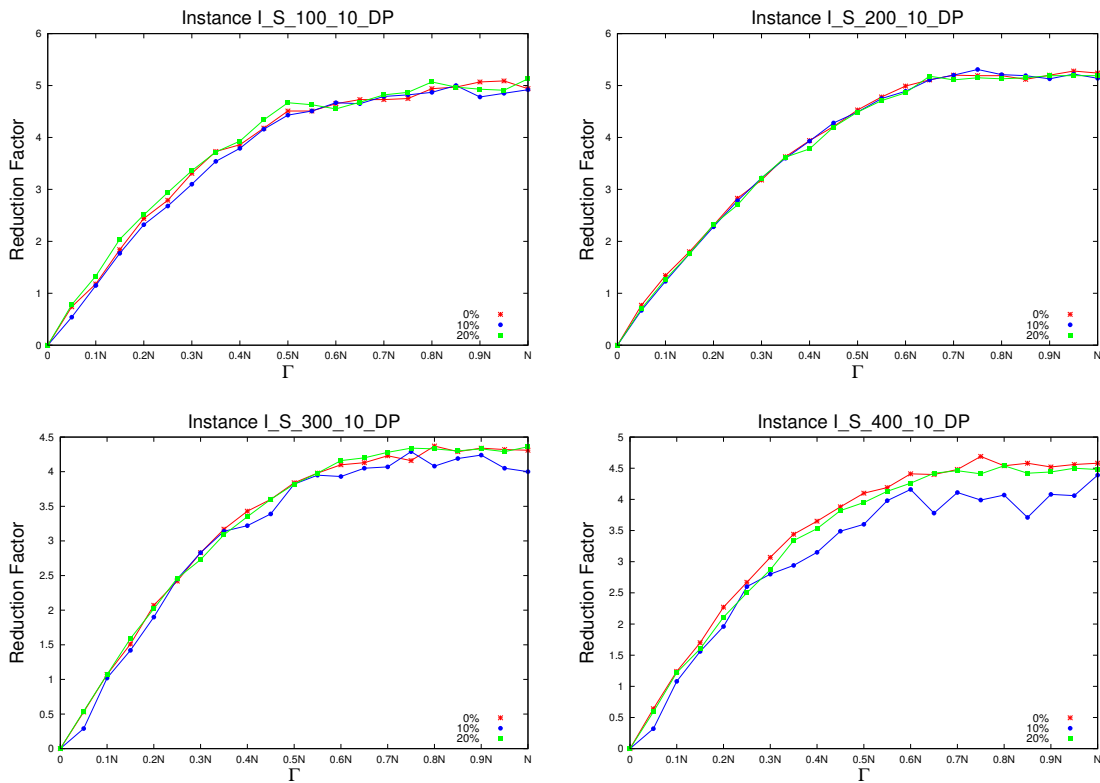


Figura 11 – Fatores de redução do AG para algumas instâncias artificiais com variações de custo de 10% e diferentes relações de precedência. Fonte: Autor.

Tabela 4 e na Figura 11, os fatores de redução aparentam ser similares, independentemente da densidade de precedência considerada. Para avaliar essa observação, as amostras são formadas pelos fatores de redução de uma certa instância para uma certa densidade de precedência. Considere $p_{a,b}^{NR}$ como sendo o p -value da comparação entre a amostra com densidade de precedência de $a\%$ e a amostra com densidade de precedência de $b\%$, para a instância com número de requisitos NR .

Para a instância com 150 requisitos, por exemplo, os p -values encontrados foram: $p_{0,10}^{150} = 0.78$, $p_{0,20}^{150} = 0.60$ e $p_{10,20}^{150} = 0.67$. Exceto para a instância com 500 requisitos, os p -values para todas as outras instâncias são parecidos. Tais resultados comprovam estatisticamente que o “preço da robustez” é o mesmo para diferentes densidades de precedência entre requisitos. Para as mesmas instâncias, com diferentes estratégias de variação de custo, os resultados também são similares.

Mesmo quando as relações de precedência são consideradas, o impacto do nível de robustez é praticamente o mesmo como mostrado nas seções anteriores. Com as amostras sendo agora formadas pelos fatores de redução de um certo nível de robustez para as instâncias com diferentes densidades de precedência e variação de custo de 10%, os p -values para pequenos valores de Γ são: $p_{0.05,0.1}^{AG} = 5.52 \times 10^{-10}$, $p_{0.1,0.15}^{AG} = 2.01 \times 10^{-9}$ e $p_{0.15,0.2}^{AG} = 6.18 \times 10^{-9}$. Para níveis de robustez maiores, os p -values são: $p_{0.6,0.65}^{AG} = 0.39$, $p_{0.9,0.95}^{AG} = 0.95$ e $p_{0.95,1}^{AG} = 0.83$.

A variação no nível da robustez só tem uma influência significativa no “preço da robustez” para pequenos valores de Γ . A partir de $\Gamma = 0.6N$, a perda em qualidade da solução torna-se quase constante.

A partir dos resultados apresentados, é possível afirmar que, em geral, a densidade de precedência não tem influência no “preço da robustez” do modelo proposto. Como nas análises anteriores, a perda em qualidade devido à robustez é pequena e o fator de redução para diferentes densidades de precedência é estatisticamente o mesmo, independente do número de requisitos e da estratégia de variação de custo. Além do mais, o comportamento do modelo robusto para diferentes níveis de robustez não é afetado pelas diferentes densidades de precedência. Variações no parâmetro Γ só são estatisticamente significativas para baixos níveis de robustez.

5.2.4 Preço da Robustez para instâncias reais do problema

A fim de realizar essa análise, as mesmas avaliações realizadas para as instâncias artificiais também foram realizadas para as instâncias reais. A Tabela 5 apresenta os resultados do AG para algumas instâncias reais com diferentes estratégias de variação de custo. São apresentados tanto resultados do Eclipse como do Mozilla. Resultados das instâncias reais restantes foram omitidos por questões de espaço.

Para uma variação de custo de 10%, um nível de robustez de 100% é atingido perdendo 6.17% em valor de *fitness*, em média. Para as instâncias reais do NRP, assim como para as instâncias artificiais, o modelo robusto proposto permite um alto nível de robustez com uma perda consideravelmente pequena em qualidade da solução. Com o aumento da variação de custo, o fator de redução também aumenta, comportamento esse também presente nos resultados das instâncias artificiais. Para uma alta variação de custo, como 50%, pode-se garantir 100% de robustez perdendo por volta de 22% em valor de *fitness*. Tal “preço da robustez” pode ainda ser considerado pequeno.

A Figura 12 mostra os resultados de algumas instâncias reais com relação ao fator de redução. Percebe-se que de fato os resultados das instâncias reais são similares aos resultados das instâncias artificiais. A estratégia de variação de custo novamente aparenta ser um fator importante para o “preço da robustez”. Da mesma forma, o comportamento do “preço da robustez” se tornar quase constante após valores intermediários de Γ também é visível. Vale ainda ressaltar que os resultados são similares para todas as instâncias mostradas.

Ao realizar uma comparação estatística dos fatores de redução de diferentes variações de custo para uma mesma instância, a diferença é comprovada. Considere $p_{a,b}^{P,NR}$ como sendo o *p-value* da comparação entre a amostra com variação de custo de $a\%$ e a amostra com variação de custo de $b\%$ para a instância do projeto P (E para Eclipse e M para Mozilla) e

Tabela 5 – Fatores de redução do AG para instâncias reais com diferentes variações de custo

Instância	TB	Γ						
		$0.05N$	$0.1N$	$0.3N$	$0.5N$	$0.7N$	$0.9N$	N
I_Re_E_20_VC	10%	3.02	3.22	4.45	6.24	6.03	7.33	8.22
	20%	1.45	3.94	8.37	12.24	11.68	11.34	11.89
	30%	2.96	4.82	10.74	15.91	16.53	16.53	15.91
	40%	4.33	7.49	14.85	18.56	21.92	21.24	21.03
	50%	3.96	7.5	17.98	22.29	24.38	24.38	22.98
	AL	4.3	5.93	11.39	13.51	13.51	13.78	13.71
I_Re_M_50_VC	10%	0.22	0.86	3.03	4.29	5.02	4.87	4.9
	20%	0.84	2.28	6.1	8.67	9.89	10.06	10.26
	30%	1.58	3.87	9.57	13.09	14.03	14.24	14.16
	40%	2.04	4.73	12.77	16.62	17.24	16.95	16.85
	50%	2.31	5.58	15.29	19.42	20.09	20.21	20.09
	AL	2.0	4.32	8.29	10.53	11.32	11.09	10.68
I_Re_E_100_VC	10%	0.72	1.31	3.54	5.59	6.49	6.44	6.28
	20%	1.91	2.72	7.49	11.7	12.25	12.16	12.12
	30%	2.56	4.21	10.79	16.04	16.86	16.73	16.96
	40%	2.76	5.66	14.3	20.12	20.65	21.04	20.84
	50%	4.06	7.42	18.0	23.82	24.89	24.59	24.88
	AL	3.45	5.68	11.46	13.55	13.95	14.31	14.17
I_Re_M_150_VC	10%	0.47	0.98	2.52	4.12	4.76	4.66	4.78
	20%	1.04	2.19	5.56	8.57	9.3	9.15	9.29
	30%	1.69	3.23	8.26	12.12	13.02	13.1	13.17
	40%	2.38	4.53	11.19	16.11	16.74	16.57	16.51
	50%	2.91	5.45	13.81	19.08	19.48	19.64	19.56
	AL	2.04	4.11	8.94	10.94	11.14	10.99	11.22
I_Re_E_200_VC	10%	0.85	1.55	3.75	5.91	6.84	6.65	6.7
	20%	1.59	2.76	7.42	11.64	12.21	12.35	12.42
	30%	2.66	4.84	11.36	16.42	17.55	17.54	17.34
	40%	2.96	5.62	14.94	20.5	21.58	21.49	21.28
	50%	3.82	7.28	18.59	24.65	25.11	25.14	25.26
	AL	3.14	5.5	11.78	13.45	13.47	13.97	14.09

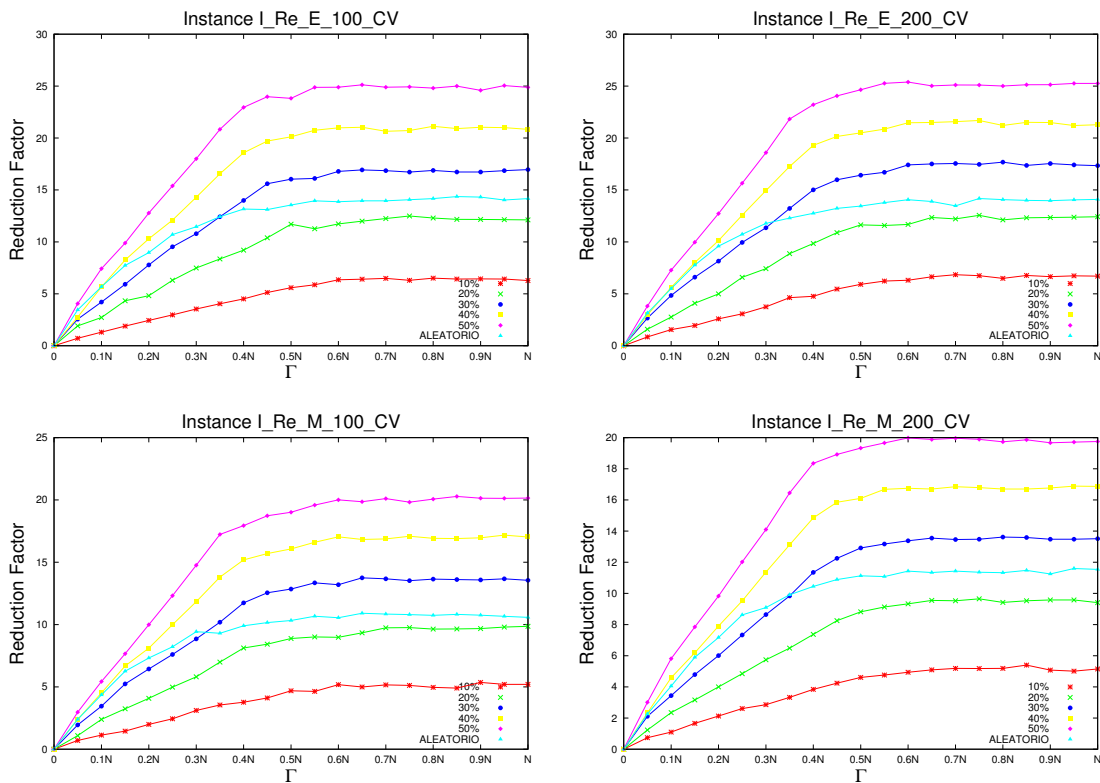


Figura 12 – Fatores de redução do AG para algumas instâncias reais com diferentes variações de custo. Fonte: Autor.

número de requisitos NR . Para a instância do Eclipse com 100 requisitos, por exemplo, os p -values são: $p_{10,20}^{E,100} = 7.1 \times 10^{-3}$, $p_{20,30}^{E,100} = 6.3 \times 10^{-2}$, $p_{30,40}^{E,100} = 0.015$, $p_{40,50}^{E,100} = 0.014$ e $p_{50,RD}^{E,100} = 7.8 \times 10^{-3}$. Para todas as outras instâncias reais os p -values são similares. Assim, a estratégia de variação de custo tem uma influência significativa no “preço da robustez” das instâncias reais.

Com relação ao “preço da robustez” para as instâncias reais quando o nível de robustez varia, os resultados também são parecidos com os das instâncias artificiais. Amostras são compostas pelos fatores de redução de um certo nível de robustez para todas as instâncias reais, inclusive as que não são apresentadas na tabela. Para pequenos valores de Γ , uma mudança no nível da robustez conduz a uma mudança significativa no fator de redução. Os p -values são: $p_{0.05,0.1}^{GA} = 1.21 \times 10^{-8}$, $p_{0.1,0.15}^{GA} = 3.65 \times 10^{-3}$ e $p_{0.15,0.2}^{GA} = 5.1 \times 10^{-2}$. Para maiores valores de Γ , a mudança no fator de redução é praticamente insignificante: $p_{0.5,0.55}^{GA} = 0.7$, $p_{0.9,0.95}^{GA} = 0.98$ e $p_{0.95,1}^{GA} = 0.96$.

Em conclusão, todos os resultados mostrados nas avaliações acima são consistentes em mostrar que o comportamento apresentado pelo modelo quando aplicado às instâncias reais do NRP, com relação ao “preço da robustez”, é similar ao encontrado nas instâncias artificiais. Assim, o modelo apresenta uma perda de qualidade consideravelmente pequena devido à robustez quando aplicado a instâncias reais. Além do mais, a estratégia de variação de custo tem um papel importante no “preço da robustez” e o nível de robustez apresenta influência estatística no “preço da robustez” somente para pequenos valores de Γ .

5.3 Conclusões

Este capítulo apresentou o estudo empírico realizado para validar e avaliar o modelo robusto para o Problema do Próximo Release proposto neste trabalho. Primeiramente foram apresentadas as configurações do estudo, incluindo a descrição das instâncias utilizadas e as adaptações das técnicas de busca para o modelo proposto. A seguir foram mostrados os resultados encontrados, bem como as respectivas análises.

O conjunto de instâncias utilizados foi composto tanto por instâncias artificiais como por instâncias reais do NRP. As instâncias artificiais foram geradas aleatoriamente, apresentando várias situações diferentes de planejamento do próximo *release*. As instâncias reais foram adaptadas de repositórios de bugs de dois grandes projetos de código aberto. No total, foram utilizadas 240 instâncias no estudo empírico. Para resolução do modelo foram aplicadas as metaheurísticas Algoritmos Genéticos e Têmpera Simulada. Um algoritmo de Busca Aleatória também foi utilizado como teste de sanidade das outras técnicas de busca.

A análise dos resultados se concentrou em avaliar o “preço da robustez” mediante a aplicação do modelo proposto a instâncias que apresentavam diferentes situações de planejamento

do próximo *release*. Primeiramente, o modelo foi aplicado a instâncias artificiais, utilizando uma variação de custo de 10% da estimativa original e sem relações de precedência. Como resultado, demonstrou-se que o ganho em robustez foi obtido com uma perda pequena em valor de *fitness* para todas as instâncias. Testes estatísticos foram utilizados para avaliar o impacto do nível da robustez no “preço da robustez”. Como resultado, o modelo apresentou uma perda em qualidade quase constante depois que o parâmetro de controle da robustez foi configurado para metade do número de requisitos. Além disso, testes estatísticos também comprovaram que o “preço da robustez” é praticamente independente do número de requisitos na instância.

Após isso, o “preço da robustez” foi computado para as instâncias artificiais com diferentes estratégias de variação de custo. Como esperado e estatisticamente comprovado, quanto maior a variação, maior é a perda em *fitness*. No entanto, mesmo com variações altas, foi possível atingir altos níveis de robustez com uma pequena perda em qualidade da solução. Apesar de influenciar diretamente no “preço da robustez” em si, a variação de custo não influencia no comportamento do modelo de apresentar uma perda em *fitness* praticamente constante a partir de níveis de robustez intermediários.

Em sequência, o modelo foi aplicado a instâncias com diferentes relações de precedência entre requisitos. Foi comprovado que, para as interdependências consideradas, diferentes relações de precedência não influenciam no “preço da robustez”. O modelo conseguiu ainda gerar soluções com pequena perda em qualidade devido a robustez e conseguiu manter a mesma perda em *fitness* a partir de níveis de robustez intermediários.

Ao final, o modelo robusto foi aplicado às instâncias reais. Os resultados foram similares aos das instâncias artificiais. Assim, todos os resultados indicam que a formulação proposta pode ser utilizada para produzir soluções robustas com pequenas perdas em qualidade, inclusive em projetos reais.

Neste capítulo foi mostrado tanto a adaptação e implementação de diferentes técnicas de busca para o modelo proposto como a avaliação do “preço da robustez” do modelo. Dessa forma, a partir do estudo empírico realizado, o restante dos objetivos específicos foram totalmente atingidos.

6 CONSIDERAÇÕES FINAIS

Neste capítulo serão feitas as considerações finais deste trabalho de dissertação. Serão apresentadas as principais contribuições da pesquisa bem como algumas limitações da mesma. Posteriormente são apontadas direções de pesquisa para trabalhos futuros. Finalmente, é apresentada a conclusão do trabalho.

6.1 Contribuições e Limitações da Pesquisa

O modelo de desenvolvimento de software iterativo e incremental vem sendo cada vez mais utilizado em âmbito comercial. Apesar das várias vantagens desse processo, algumas dificuldades são incorporadas no gerenciamento do projeto, como a escolha de quais requisitos serão implementados a cada *release* do sistema. A seleção dos requisitos deve ser feita de forma a maximizar a satisfação do cliente, respeitar o orçamento disponível para o *release* e respeitar as interdependências entre os requisitos, caracterizando o problema conhecido como NRP. Dessa forma, a partir de uma modelagem matemática do problema, algoritmos de busca podem ser utilizados para se realizar um planejamento ótimo do próximo *release*.

A motivação desta pesquisa se deve ao fato das atuais abordagens baseadas em busca para o NRP não considerarem as incertezas relacionadas aos requisitos, principalmente as incertezas de importância e custo de desenvolvimento de cada requisito. Ao assumir que tais valores são corretos, soluções encontradas pelos algoritmos de busca podem se mostrar inválidas ao colocadas em prática, num ambiente incerto, diferente do que foi modelado.

Assim, a principal contribuição desta pesquisa consiste na proposta de um novo modelo matemático para o NRP que considera as incertezas de importância e custo dos requisitos. Esse modelo foi desenvolvido a partir da Otimização Robusta e é capaz de gerar soluções robustas pra o NRP, ou seja, *releases* que, mesmo na presença de estimativas incertas, continuam respeitando todas as restrições de custo e interdependências. Após encontrar as melhores estratégias para quantificação das incertezas e desenvolver o modelo robusto do problema, duas metaheurísticas (Algoritmos Genéticos e Têmpera Simulada) foram utilizadas para resolução do modelo. Foram utilizadas instâncias artificiais e reais do problema, apresentando diferentes situações de planejamento do próximo *release*. Os resultados foram utilizados para se realizar uma análise detalhada do “preço da robustez” do modelo proposto, ou seja, o quanto é perdido em qualidade da solução devido à robustez.

Acredita-se que todos os objetivos listados por esta pesquisa foram atingidos ao longo do seu desenvolvimento. Apesar disso, o trabalho ainda apresenta algumas limitações: a utilização de instâncias reais adaptadas de repositórios de bugs. Tais instâncias são relativamente simples e não conseguem explorar todas as características do modelo proposto; o fato

de se considerar somente um tipo de interdependência entre requisitos; a falta da avaliação do modelo proposto por profissionais que trabalham diariamente com planejamento de *releases*. Tal avaliação teria proporcionado valiosos *feedbacks* para uma melhora do modelo; a utilização de somente duas técnicas de busca mais sofisticadas para resolução do modelo. A utilização de mais algoritmos de busca proporcionaria análises ainda mais confiáveis do comportamento do modelo;

6.2 Trabalhos Futuros

O aperfeiçoamento e a continuidade deste trabalho, além das próprias limitações apresentadas anteriormente, constituem as oportunidades para trabalhos futuros, que incluem:

- Aplicar o modelo robusto proposto a um projeto real de desenvolvimento de softwares. De preferência, essa aplicação deve ser realizada dentro da empresa, sendo monitorada por alguém com conhecimento do modelo e sendo coletados dados e resultados para futuras análises.
- Considerar outras interdependências entre requisitos, incluindo as demais interdependências funcionais e as interdependências de valor.
- Investigar diferentes formas de quantificar as incertezas de importância e custo dos requisitos, verificando se diferentes formas de quantificação levam a diferentes comportamentos do “preço da robustez”.
- Analisar diferentes formas de se obter o parâmetro de controle da robustez.
- Aplicar mais técnicas de busca na resolução do modelo, incluindo algoritmos exatos e outras metaheurísticas como otimização por colônia de formigas e otimização por enxame de partículas.
- Desenvolver uma aplicação que possibilite uma interface gráfica para utilização do modelo. Essa interface deve ser capaz de: cadastrar os requisitos e inserir as respectivas informações de cenários, custos de desenvolvimento e variação de custo esperada. Após esse cadastro, escolher uma certa técnica de busca para resolução e visualizar o *release* gerado.
- Utilizar as mesmas quantificações das incertezas de importância e custo dos requisitos para outros problemas da Engenharia de Requisitos também tratados pela SBSE, incluindo a versão multiobjetivo do NRP, o Problema do Planejamento de Releases e o Problema da Priorização de Requisitos.

- Aplicar o *framework* da Otimização Robusta a outros problemas da Engenharia de Software, como gerenciamento de projetos.

6.3 Conclusão

O Problema do Próximo Release consiste em selecionar quais requisitos serão implementados no próximo *release* do software. A escolha deve maximizar a satisfação do cliente, respeitando o orçamento disponível e os relacionamentos entre os requisitos. Abordagens baseadas em busca vêm se mostrando interessantes para abordar esse problema, mas as modelagens matemáticas utilizadas atualmente não consideram as incertezas inerentes aos requisitos.

Este trabalho apresentou uma modelagem matemática baseada em Otimização Robusta para o NRP. Esse novo modelo é capaz de gerar *releases* robustas, que mesmo na presença de incertezas nas estimativas de importância e custo de desenvolvimento dos requisitos, ainda apresentam uma boa qualidade e respeitam todas as restrições.

Diferentes instâncias artificiais e reais foram utilizadas para avaliar e validar o modelo. As instâncias apresentaram diferentes situações de planejamento do próximo *release*, incluindo diferentes números de requisitos, diferentes habilidades de estimativa e diferentes relações entre requisitos. As metaheurísticas Algoritmos Genéticos e Têmpera Simulada foram aplicadas e os resultados coletados foram utilizados para avaliar o “preço da robustez” do modelo proposto.

De forma geral, a perda em qualidade da solução devido a robustez não é grande, inclusive para grandes variações nas incertezas de custo. O “preço da robustez” se mostrou praticamente constante no momento em que o nível de robustez foi configurado para metade da quantidade de requisitos e ainda se apresentou independente da quantidade de requisitos na instância. O “preço da robustez” é altamente influenciado pela estratégia de variação de custo utilizada mas, ao mesmo tempo, não é influenciado pelas relações entre requisitos. Finalmente, os resultados encontrados para as instâncias reais se mostram muito similares aos encontrados para as instâncias artificiais.

REFERÊNCIAS

- ABDULLAH, M. M.; RICHARDSON, A.; HANIF, J. Placement of sensors/actuators on civil structures using genetic algorithms. *Earthquake engineering & structural dynamics*, Wiley Online Library, v. 30, n. 8, p. 1167–1184, 2001.
- AISSI, H.; BAZGAN, C.; VANDERPOOTEN, D. Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, Elsevier, v. 197, n. 2, p. 427–438, 2009.
- AKKER, J. M. Van den; BRINKKEMPER, S.; DIEPEN, G.; VERSEDAAL, J. Determination of the next release of a software product: an approach using integer linear programming. In: *Proceeding of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2005)*. [S.l.: s.n.], 2005. p. 119–124.
- ANDRADE, E.L. Introdução à pesquisa operacional. *Rio de Janeiro: LTC*, 1998.
- ANTONIOL, G.; PENTA, M. Di; HARMAN, M. A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty. In: *IEEE. Software Metrics, 2004. Proceedings. 10th International Symposium on*. [S.l.], 2004. p. 172–183.
- ARCURI, A.; BRIAND, L. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: *IEEE. Software Engineering (ICSE), 2011 33rd International Conference on*. [S.l.], 2011. p. 1–10.
- ASSUNÇÃO, W. K. G.; COLANZI, T. E.; POZO, A. T. R.; VERGILIO, S. R. Establishing integration test orders of classes with several coupling measures. In: *ACM. Proceedings of the 13th annual conference on Genetic and evolutionary computation*. [S.l.], 2011. p. 1867–1874.
- BAGNALL, A. J.; RAYWARD-SMITH, V. J.; WHITTLE, I. M. The next release problem. *Information and Software Technology*, Elsevier, v. 43, n. 14, p. 883–890, 2001.
- BAI, D.; CARPENTER, T.; MULVEY, J. Making a case for robust optimization models. *Management science*, INFORMS, v. 43, n. 7, p. 895–907, 1997.
- BAKER, P.; HARMAN, M.; STEINHOFEL, K.; SKALIOTIS, A. Search based approaches to component selection and prioritization for the next release problem. In: *IEEE. Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on*. [S.l.], 2006. p. 176–185.
- BARROS, M.; DIAS-NETO, A. C. A survey of empirical investigations on ssbse papers. In: *Search Based Software Engineering*. [S.l.]: Springer, 2011. p. 268–268.
- BERTSIMAS, D.; SIM, M. The price of robustness. *Operations research*, INFORMS, v. 52, n. 1, p. 35–53, 2004.
- BEYER, H. G.; SENDHOFF, B. Robust optimization—a comprehensive survey. *Computer methods in applied mechanics and engineering*, Elsevier, v. 196, n. 33, p. 3190–3218, 2007.
- BIANCHI, N.; BOLOGNANI, S. Design optimisation of electric motors by genetic algorithms. In: *IET. Electric Power Applications, IEE Proceedings-*. [S.l.], 1998. v. 145, n. 5, p. 475–483.

- BLICKLE, T.; THIELE, L. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, MIT Press, v. 4, n. 4, p. 361–394, 1996.
- BOEHM, B.; ABTS, C.; CHULANI, S. Software development cost estimation approaches, a survey. *Annals of Software Engineering*, Springer, v. 10, n. 1, p. 177–205, 2000.
- BRASIL, M. M. A.; NEPOMUCENO, T. G.; FERREIRA, T. do N.; FREITAS, F. G.; SOUZA, J. T. Uma abordagem baseada em busca para o problema da seleção de requisitos de software na presença de similaridade. In: *Anais do XLIV Simpósio Brasileiro de Pesquisa Operacional*. [S.l.: s.n.], 2012.
- BRASIL, M. M. A.; SILVA, T. G. N.; FREITAS, F.G. de; FERREIRA, T. N.; CORTÉS, M. I.; SOUZA, J. T. Aplicando técnicas de busca multiobjetivas na priorização de requisitos de software. In: *Anais do XLIII Simpósio Brasileiro de Pesquisa Operacional*. [S.l.: s.n.], 2011.
- BRIAND, L. C.; FENG, J.; LABICHE, Y. Using genetic algorithms and coupling measures to devise optimal integration test orders. In: ACM. *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. [S.l.], 2002. p. 43–50.
- BURKE, E. K.; KENDALL, G. *Search methodologies*. [S.l.]: Springer, 2005.
- CAO, L.; RAMESH, B. Agile requirements engineering practices: An empirical study. *Software, IEEE, IEEE*, v. 25, n. 1, p. 60–67, 2008.
- CARLSHAMRE, P.; SANDAHL, K.; LINDVALL, M.; REGNELL, B.; DAG, J. Natt och. An industrial survey of requirements interdependencies in software product release planning. In: IEEE. *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*. [S.l.], 2001. p. 84–91.
- COLARES, F.; SOUZA, J. T.; CARMO, R.; PÁDUA, C.; MATEUS, G. R. A new approach to the software release planning. In: IEEE. *Software Engineering, 2009. SBES'09. XXIII Brazilian Symposium on*. [S.l.], 2009. p. 207–215.
- DOLADO, J. J. A validation of the component-based method for software size estimation. *Software Engineering, IEEE Transactions on*, IEEE, v. 26, n. 10, p. 1006–1021, 2000.
- DU, X.; WANG, Y.; CHEN, W. Methods for robust multidisciplinary design. *AIAA*, v. 1785, p. 1–10, 2000.
- DULIKRAVICH, G. S.; EGOROV-YEGOROV, I. N. Robust optimization of concentrations of alloying elements in steel for maximum temperature, strength, time-to-rupture and minimum cost and weight. *ECCOMAS–Computational Methods for Coupled Problems in Science and Engineering*, Fira: Santorini Island, Greece, p. 25–28, 2005.
- DURILLO, J. J.; ZHANG, Y.; ALBA, E.; HARMAN, M.; NEBRO, A. J. A study of the bi-objective next release problem. *Empirical Software Engineering*, Springer, v. 16, n. 1, p. 29–60, 2011.
- DURILLO, J. J.; ZHANG, Y. Y.; ALBA, E.; NEBRO, A. J. A study of the multi-objective next release problem. In: IEEE. *Search Based Software Engineering, 2009 1st International Symposium on*. [S.l.], 2009. p. 49–58.
- ECLIPSE. 2014. <http://www.eclipse.org/>. Acessado em: Junho de 2014.

- EIBEN, A. E.; SMITH, J. E. *Introduction to evolutionary computing*. [S.l.]: springer, 2003.
- ELBAUM, S.; MALISHEVSKY, A. G.; ROTHERMEL, G. Test case prioritization: A family of empirical studies. *Software Engineering, IEEE Transactions on*, IEEE, v. 28, n. 2, p. 159–182, 2002.
- ERASER, A. Simulation of genetic systems by automatic digital computers. i. introduction. *Australian Journal of Biological Sciences*, v. 10, p. 484–491, 1957.
- FERREIRA, T. N.; SOUZA, J. T. Uma abordagem aco para o problema do próximo release com interdependência de requisitos. In: *Anais do III Workshop de Engenharia de Software Baseada em Busca*. [S.l.: s.n.], 2012.
- FREITAS, F. G.; COUTINHO, D. P.; SOUZA, J. T. Software next release planning approach through exact optimization. *International Journal of Computer Applications*, v. 22, n. 8, p. 1–8, 2011.
- FREITAS, F. G.; SILVA, T.; CARMO, R.; SOUZA, J. T. On the applicability of exact optimization in search based software engineering. In: SPRINGER-VERLAG. *Proceedings of the Third international conference on Search based software engineering*. [S.l.], 2011. p. 276–276.
- FREITAS, F. G.; SOUZA, J. T. Applying mathematical programming to efficient software release management. *Computational Intelligence and Information Technology*, Springer, p. 564–568, 2011.
- GREER, D.; RUHE, G. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, Elsevier, v. 46, n. 4, p. 243–253, 2004.
- GUEORGUIEV, S.; HARMAN, M.; ANTONIOL, G. Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering. In: ACM. *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. [S.l.], 2009. p. 1673–1680.
- HARKER, S. D. P.; EASON, K. D.; DOBSON, J. E. The change and evolution of requirements as a challenge to the practice of software engineering. In: IEEE. *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*. [S.l.], 1993. p. 266–272.
- HARMAN, M. The current state and future of search based software engineering. In: IEEE COMPUTER SOCIETY. *2007 Future of Software Engineering*. [S.l.], 2007. p. 342–357.
- HARMAN, M.; CLARK, J. Metrics are fitness functions too. In: IEEE. *Software Metrics, 2004. Proceedings. 10th International Symposium on*. [S.l.], 2004. p. 58–69.
- HARMAN, M.; JONES, B. F. Search-based software engineering. *Information and Software Technology*, Elsevier, v. 43, n. 14, p. 833–839, 2001.
- HARMAN, M.; KRINKE, J.; REN, J.; YOO, S. Search based data sensitivity analysis applied to requirement engineering. In: ACM. *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. [S.l.], 2009. p. 1681–1688.
- HOLLAND, J. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. USA: University of Michigan, 1975.

- JIANG, H.; XUAN, J.; REN, Z. Approximate backbone based multilevel algorithm for next release problem. In: ACM. *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. [S.l.], 2010. p. 1333–1340.
- JIANG, H.; ZHANG, J.; XUAN, J.; REN, Z.; HU, Y. A hybrid aco algorithm for the next release problem. In: IEEE. *Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on*. [S.l.], 2010. p. 166–171.
- JINTAO, M.; LAI, L. L.; YIHAN, Y. Application of genetic algorithms in reactive power optimization. *Chinese Society For Electrical Engineering*, v. 5, 1995.
- JORGENSEN, M. Evidence-based guidelines for assessment of software development cost uncertainty. *Software Engineering, IEEE Transactions on*, IEEE, v. 31, n. 11, p. 942–954, 2005.
- JUDSON, R. Genetic algorithms and their use in chemistry. *Reviews in Computational Chemistry, Volume 10*, Wiley Online Library, p. 1–73, 2007.
- KARLSSON, J.; WOHLIN, C.; REGNELL, B. An evaluation of methods for prioritizing software requirements. *Information and Software Technology*, Elsevier, v. 39, n. 14, p. 939–947, 1998.
- KIRKPATRICK, S.; VECCHI, M. P.; GELATT, C. D. Optimization by simulated annealing. *science*, Washington, v. 220, n. 4598, p. 671–680, 1983.
- KIRSOPP, C.; SHEPPERD, M. J.; HART, J. Search heuristics, case-based reasoning and software project effort prediction. Morgan Kaufmann Publishers Inc., 2002.
- LARMAN, C.; BASILI, V. R. Iterative and incremental developments. a brief history. *Computer*, IEEE, v. 36, n. 6, p. 47–56, 2003.
- LEUNG, S. C. H.; TSANG, S. O. S.; NG, W.; WU, Y. A robust optimization model for multi-site production planning problem in an uncertain environment. *European Journal of Operational Research*, Elsevier, v. 181, n. 1, p. 224–238, 2007.
- LI, M.; AZARM, S. Multiobjective collaborative robust optimization with interval uncertainty and interdisciplinary uncertainty propagation. *Journal of mechanical design*, American Society of Mechanical Engineers, v. 130, n. 8, 2008.
- LIMA, D.; FREITAS, F. G.; CAMPOS, G.; SOUZA, J. T. A fuzzy approach to requirements prioritization. *Search Based Software Engineering*, Springer, p. 64–69, 2011.
- LINHARES, G. R. M.; FREITAS, F. G.; CARMO, R. A. F.; MAIA, C. L.; SOUZA, J. T. Aplicação do algoritmo grasp reativo para o problema do próximo release. *XLII Simpósio Brasileiro de Pesquisa Operacional (SBPO'2010)*, 2010.
- MALCOLM, S. A.; ZENIOS, S. A. Robust optimization for power systems capacity expansion under uncertainty. *Journal of the operational research society*, JSTOR, p. 1040–1049, 1994.
- MKAOUER, MohamedWiem; KESSENTINI, Marouane; BECHIKH, Slim; CINNÉIDE, Mel Ó. A robust multi-objective approach for software refactoring under uncertainty. In: *Search-Based Software Engineering*. [S.l.]: Springer International Publishing, 2014, (Lecture Notes in Computer Science, v. 8636). p. 168–183.

- MOZILLA. 2014. <http://www.mozilla.org/>. Acessado em: Junho de 2014.
- MULVEY, J. M.; VANDERBEI, R. J.; ZENIOS, S. A. Robust optimization of large-scale systems. *Operations research*, INFORMS, v. 43, n. 2, p. 264–281, 1995.
- PAIXAO, M.; BRASIL, M. M. A.; NEPOMUCENO, T.; SOUZA, J. T. Aplicando o algoritmo ant-q na priorização de requisitos de software com precedência. In: *III Workshop Brasileiro de Engenharia de Software Baseada em Busca - WESB'12*. [S.l.: s.n.], 2012.
- R-PROJECT. 2014. <http://www.r-project.org/>. Acessado em: Junho de 2014.
- RAMIREZ, A. J.; FREDERICKS, E. M.; JENSEN, A. C.; CHENG, B. H. C. Automatically relaxing a goal model to cope with uncertainty. In: *Search Based Software Engineering*. [S.l.]: Springer, 2012. p. 198–212.
- ROTHERMEL, G.; UNTCH, R. H.; CHU, C.; HARROLD, M. J. Prioritizing test cases for regression testing. *Software Engineering, IEEE Transactions on*, IEEE, v. 27, n. 10, p. 929–948, 2001.
- ROY, B. Robustness in operations research and decision aiding. In: *Flexibility and Robustness in Scheduling*. [S.l.]: Wiley, 2008. p. 35–52.
- ROY, B. To better respond to the robustness concern in decision aiding: four proposals based on a twofold observation. In: *Handbook of Multicriteria Analysis*. [S.l.]: Springer, 2010. p. 3–24.
- RUHE, G. Software release planning. *Handbook of software engineering and knowledge engineering*, v. 3, p. 365–394, 2005.
- RUHE, G.; SALIU, M. O. The art and science of software release planning. *Software, IEEE*, IEEE, v. 22, n. 6, p. 47–53, 2005.
- SAGRADO, J. Del; AGUILA, I. M.; ORELLANA, F. J. Requirements interaction in the next release problem. In: ACM. *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*. [S.l.], 2011. p. 241–242.
- SAGRADO, J. Del; AGUILA, I. M. del; ORELLANA, F. J. Ant colony optimization for the next release problem: A comparative study. In: IEEE. *Search Based Software Engineering (SSBSE), 2010 Second International Symposium on*. [S.l.], 2010. p. 67–76.
- SOMMERVILLE, I. *Software Engineering*. [S.l.]: Addison Wesley, 2011.
- SOUZA, J. T.; MAIA, C.; FERREIRA, T.; CARMO, R.; BRASIL, M. An ant colony optimization approach to the software release planning with dependent requirements. *Search Based Software Engineering*, Springer, p. 142–157, 2011.
- SOUZA, J. T.; MAIA, C. L.; FREITAS, F. G.; COUTINHO, D. P. The human competitiveness of search based software engineering. In: IEEE. *Search Based Software Engineering (SSBSE), 2010 Second International Symposium on*. [S.l.], 2010. p. 143–152.
- TAGUCHI, G. *Introduction to quality engineering: designing quality into products and processes*. [S.l.: s.n.], 1986.

WANG, J.; RONG, G. Robust optimization model for crude oil scheduling under uncertainty. *Industrial & Engineering Chemistry Research*, ACS Publications, v. 49, n. 4, p. 1737–1748, 2009.

WEGENER, J.; MUELLER, F. A comparison of static analysis and evolutionary testing for the verification of timing constraints. *Real-Time Systems*, Springer, v. 21, n. 3, p. 241–268, 2001.

XUAN, J.; JIANG, H.; REN, Z.; LUO, Z. Solving the large scale next release problem with a backbone-based multilevel algorithm. *Software Engineering, IEEE Transactions on*, v. 38, n. 5, p. 1195–1212, sept.-oct. 2012. ISSN 0098-5589.

YANG, B.; HU, H.; JIA, L. A study of uncertainty in software cost and its impact on optimal software release time. *Software Engineering, IEEE Transactions on, IEEE*, v. 34, n. 6, p. 813–825, 2008.

YU, G. On the max-min 0-1 knapsack problem with robust optimization applications. *Operations Research*, INFORMS, v. 44, n. 2, p. 407–415, 1996.

ZHANG, Y. *Repository of Publications on Search Based Software Engineering*. 2014. http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/. Acessado em: Junho de 2014.

ZHANG, Y.; ALBA, E.; DURILLO, J. J.; ELDH, S.; HARMAN, M. Today/future importance analysis. In: *ACM Genetic and Evolutionary Computation Conference (GECCO 2010)*. [S.l.: s.n.], 2010. p. 1357–1364.

ZHANG, Y.; FINKELSTEIN, A.; HARMAN, M. Search based requirements optimisation: Existing work and challenges. *Requirements Engineering: Foundation for Software Quality*, Springer, p. 88–94, 2008.

ZHANG, Y.; HARMAN, M.; MANSOURI, S. A. The multi-objective next release problem. In: *ACM. Proceedings of the 9th annual conference on Genetic and evolutionary computation*. [S.l.], 2007. p. 1129–1137.