

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS GRADUAÇÃO EM INFORMÁTICA

**Uma Abordagem Baseada em Riscos de Software para
Seleção de Requisitos**

Aruan Galves Nascimento Amaral

João Pessoa - PB
Agosto de 2017

Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Informática

**Uma Abordagem Baseada em Riscos de Software para
Seleção de Requisitos**

Aruan Galves Nascimento Amaral

João Pessoa - PB
Agosto, 2017

Aruan Galves Nascimento Amaral

**Uma Abordagem Baseada em Riscos de Software para
Seleção de Requisitos**

Dissertação apresentada ao Centro de Informática
da Universidade Federal da Paraíba, como
requisito parcial para obtenção do título de mestre
em informática (sistemas de computação).

Orientador: Prof. Dr. Gledson Elias

João Pessoa - PB

Agosto, 2017

A485a Amaral, Aruan Galves Nascimento.
 Uma abordagem baseada em riscos de software para
 seleção de requisitos / Aruan Galves Nascimento Amaral. - João
 Pessoa, 2017.
 159 f. : il.

 Orientador: Gledson Elias.
 Dissertação (Mestrado) - UFPB/CI

 1. Informática. 2. Gerenciamento de riscos. 3. Requisitos de
 software. I. Título.

UFPB/BC

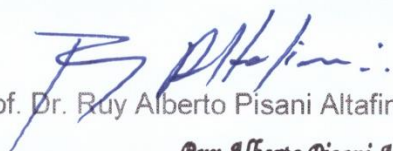


UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA



Ata da Sessão Pública de Defesa de Dissertação de Mestrado de Aruan Galves Nascimento Amaral, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 22 de agosto de 2017.

1 Aos vinte e dois dias do mês de agosto, do ano de dois mil e dezessete, às quatorze horas,
2 no Centro de Informática da Universidade Federal da Paraíba, em Mangabeira, reuniram-se
3 os membros da Banca Examinadora constituída para julgar o Trabalho Final do Sr. Aruan
4 Galves Nascimento Amaral, vinculado a esta Universidade sob a matrícula nº 2015103376,
5 candidato ao grau de Mestre em Informática, na área de "Sistemas de Computação", na linha
6 de pesquisa "Computação distribuída", do Programa de Pós-Graduação em Informática, da
7 Universidade Federal da Paraíba. A comissão examinadora foi composta pelos professores:
8 Gledson Elias da Silveira (PPGI-UFPB), Orientador e Presidente da Banca, Clauriton de
9 Albuquerque Siebra (PPGI-UFPB), Examinador Interno, e Rita Suzana Pitangueira Maciel
10 (UFBA), Examinadora Externa à Instituição. Dando início aos trabalhos, o Presidente da
11 Banca, cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e
12 passou a palavra ao candidato para que o mesmo fizesse a exposição oral do trabalho de
13 dissertação intitulado "Uma abordagem baseada em riscos de software para seleção de
14 requisitos". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que
15 emitiu o seguinte parecer: "**aprovado**". Do ocorrido, eu, Ruy Alberto Pisani Altafim, Vice-
16 Coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai
17 assinada por mim e pelos membros da banca examinadora. João Pessoa, 22 de agosto de
18 2017.


Prof. Dr. Ruy Alberto Pisani Altafim

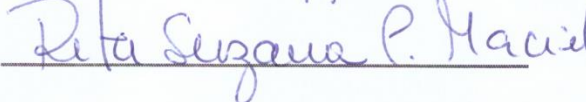
Ruy Alberto Pisani Altafim
Vice-Coordenador do Programa de
Pós-Graduação em Informática
SIAPE 1971934

Prof. Dr. Gledson Elias da Silveira
Orientador (PPGI-UFPB)

Prof. Dr. Clauriton de Albuquerque Siebra
Examinador interno (PPGI-UFPB)

Prof. Dr. Rita Suzana Pitangueira Maciel
Examinadora externa à Instituição (UFBA)





Agradecimentos

Primeiramente, gostaria de agradecer ao meu orientador por todo o processo envolvido na realização desta dissertação, desde as primeiras aulas de Engenharia de Software no mestrado que por conseguinte adentrei o Estágio Docência fazendo parte desta disciplina, que foi uma das minhas experiências mais positivas que passei no mestrado, além de todo o auxílio nas orientações da qualificação, dos artigos submetidos, além desta própria dissertação. Além disso, agradeço pela compreensão dele pelos problemas pessoais que enfrentei durante minha caminhada no mestrado.

Em seqüência, agradeço aos meus pais por todo o apoio nesse mestrado e que, mesmo diante das dificuldades encontradas no dia-a-dia, puderam me ajudar conforme suas possibilidades em resolver todos os problemas que enfrentei ao longo destes 30 meses. Extendendo esse agradecimento à toda a minha família pelo apoio afetivo que encontrei neles, não só neste mestrado, mas durante toda minha vida.

Agradeço também aos meus amigos e todas as pessoas que passei a conhecer e interagi ao longo do meu mestrado que me auxiliaram na realização deste trabalho, no apoio em todos os momentos de necessidade e descontração.

Finalmente, agradeço à Deus, sendo obrigado a testemunhar Sua ação em vários pontos do meu mestrado e de minha vida, que abriu todos os caminhos que permitiu a finalização deste trabalho e do meu fortalecimento pessoal.

Resumo

Em abordagens de desenvolvimento incremental, existe um grande interesse em entregar *releases* do sistema dentro do cronograma, sem estourar o orçamento, e, ao mesmo tempo, que agregue o máximo de satisfação possível aos interessados no projeto. Assim, o processo de seleção de requisitos é um fator-chave para identificar um subconjunto bom ou até mesmo ótimo de requisitos candidatos que atendam estas condições. Técnicas tradicionais que envolvem um processo manual de seleção e priorização de requisitos encontram limitações ao lidar com um grande conjunto de requisitos. Nesta direção, o Problema do Próximo *Release*, do inglês Next Release Problem (NRP), apresenta uma modelagem computacional para este processo decisório, evoluindo desde uma abordagem simples com um único objetivo a ser alcançado dado um orçamento-limite, partindo para abordagens com múltiplos objetivos que flexibilizam a tomada de decisão, sem restringir limites pré-fixados para as metas a serem alcançadas no *release*. Apesar desta evolução, a maior parte das contribuições para o Problema do Próximo *Release* não tratam riscos de software, que é um fator-chave que pode impactar profundamente o custo do projeto e a satisfação dos stakeholders. Deste modo, este trabalho propõe uma abordagem baseada em riscos para um problema do próximo release multi-objetivo, em que os riscos são incorporados nas avaliações de custo e satisfação do sistema a ser entregue. De modo a validar a proposta, além de uma revisão sistemática comparando a literatura existente relacionada com o tema da abordagem proposta, o trabalho apresenta resultados de três metaheurísticas distintas capazes de lidar com múltiplos objetivos a serem otimizados. Estes algoritmos são aplicados a duas bases de dados semi-reais que revelam a eficiência e aplicabilidade potencial da abordagem proposta.

Palavras-chave: Problema do próximo release, otimização multi-objetiva, gerenciamento de riscos, requisitos de software.

Abstract

In incremental development approaches, there is a great interest to deliver system releases in a timely manner, while avoiding budget overruns and maximizing the perceived satisfaction for the stakeholders involved in the project. Thus, the requirements selection process is a key-factor in identifying a good or optimal subset of candidate requirements that meet these conditions. Traditional techniques that involve manual processes for selecting and prioritizing requirements have limitations when addressing a large number of requirements. In such a direction, the Next Release Problem (NRP) presents a computational model for this decision process, evolving from a simple single-objective approach, with a maximum allowed budget, to multi-objective approaches that make the decision process more flexible, without restricting the model to pre-fixed limits related to the goals to be achieved on the release. Despite this evolution, most of the contributions for this problem does not address software risks, which is a key-factor that may deeply impact on project cost and stakeholders' satisfaction. Therefore, this dissertation proposes a risk-based approach for a multi-objective next-release problem, in which risks are incorporated into the cost and satisfaction evaluation for the system to be delivered. In order to validate such a proposal, besides a systematic review comparing the current literature related to the proposal's theme, this dissertation presents results of three distinct metaheuristics capable of dealing with multiple objectives to be optimized. These algorithms are applied to two semi-real datasets that reveal the efficiency and potential applicability of the proposed approach.

Keywords: Next release problem, multi-objective optimization, risk management, software requirements.

Lista de Abreviaturas e Siglas

AHP	<i>Analytic Hierarchy Process</i> ; Processo Analítico de Hierarquia
cGA	<i>Cellular Genetic Algorithms</i> ; Algoritmos Genéticos Celulares
GA	<i>Genetic Algorithms</i> ; Algoritmos Genéticos
HUX	<i>Half Uniform Crossover</i> ; Cruzamento de Metade Uniforme
jMetal	<i>Metaheuristic Algorithms in Java</i>
k-NN	<i>k-th Nearest Neighbor</i> ; algoritmo do k-ésimo vizinho mais próximo
MOCeII	<i>MultiObjective Cellular genetic algorithm</i>
MONRP	<i>Multi-Objective Next Release Problem</i> ; Problema do Próximo Release Multi-Objetivo
MOP	<i>Multi-objective Optimization Problem</i> ; Problema de Otimização Multi-Objetivo
NRP	<i>Next Release Problem</i> ; Problema do Próximo Release
NSGA-II	<i>Non-dominated Sorting Genetic Algorithm-II</i>
SBSE	<i>Search Based Software Engineering</i> ; Engenharia de Software Baseada em Buscas
SPEA2	<i>Strength Pareto Evolutionary Algorithm</i>
SRP	<i>Software Release Planning</i> ; Planejamento de Releases de Software
UX	<i>Uniform Crossover</i>

Sumário

Introdução.....	1
1.1. Motivação e Justificativa	3
1.2. Objetivos.....	5
1.3. Organização do Trabalho.....	7
2. Fundamentação Teórica.....	9
2.1. Problema do Próximo Release.....	13
2.2. Problema do Próximo Release Multiobjetivo.....	15
2.2.1. Indicadores de Qualidade	17
2.2.1.1. Espalhamento	18
2.2.1.2. Hipervolume.....	20
2.3. Gerenciamento de Riscos	22
2.4. Considerações Finais	31
3. Metaheurísticas.....	33
3.1. Algoritmos Genéticos	34
3.1.1. Background da Biologia	35
3.1.2. Definição de um algoritmo genético clássico.....	37
3.1.3. Comparativo com diferentes metaheurísticas.....	40
3.1.4. Operadores.....	43
3.1.4.1. Operador de seleção	43
3.1.4.2. Operador de cruzamento	46
3.1.4.3. Operador de mutação	50
3.1.4.4. Operador de substituição	51
3.1.5. Critérios de parada.....	52
3.2. Algoritmos Genéticos Multiobjetivos	53
3.2.1. NSGA-II	55
3.2.1.1. Atribuição de dominância e separação em fronts.....	56

3.2.1.2.	Classificação de espalhamento.....	58
3.2.1.3.	Loop principal do NSGA-II	60
3.2.2.	SPEA2	60
3.2.2.1.	Loop principal do SPEA2	62
3.2.2.2.	Atribuição de fitness.....	62
3.2.2.3.	Seleção ambiental.....	64
3.2.3.	MOCeII	64
3.3.	Considerações Finais	68
4.	Abordagem Proposta	70
4.1.	Caracterização do Problema	71
4.1.1.	Recomendação de Impacto de Requisitos	72
4.1.2.	Recomendação de Sobrecarga de Requisitos	77
4.1.3.	Recomendação de Requisitos	80
4.1.3.1.	Avaliação de Satisfação	83
4.1.3.2.	Avaliação de Custo.....	84
4.1.3.3.	Avaliação Integrada de Custo-Satisfação.....	86
4.2.	Considerações Finais	87
5.	Resultados e Discussão.....	88
5.1.	Experimento com uma busca exaustiva utilizando o dataset Motorola.....	88
5.2.	Experimento multi-objetivo utilizando os datasets Microsoft Word e ReleasePlanner	96
5.3.	Considerações Finais	106
6.	Trabalhos Relacionados.....	108
6.1.	Estratégia de Busca por Trabalhos Relacionados.....	108
6.1.1.	Estratégia de busca por estudos primários relacionados à proposta.....	109
6.1.2.	Critério de inclusão dos estudos primários.....	109
6.2.	Listagem de Trabalhos Relacionados	110

6.3.	Detalhamento de Trabalhos Relacionados	112
6.3.1.	Quantitative Studies in Software Release Planning under Risk and Resource Constraints.....	112
6.3.2.	A New Approach to the Software Release Planning.....	113
6.3.3.	A Multiobjective Optimization Approach to the Software Release Planning with Undefined Number of Releases and Interdependent Requirements.....	113
6.3.4.	A Scenario-based Robust Model for the Next Release Problem.....	114
6.3.5.	A Recoverable Robust Approach to the Next Release Problem	115
6.3.6.	Robust Next Release Problem: Handling Uncertainty During Optimization...	116
6.3.7.	Interactive Software Release Planning with Preferences Base	117
6.3.8.	Risk-Aware Multi-stakeholder Next Release Planning Using Multi-objective Optimization	118
6.3.9.	Uma Abordagem Multiobjetivo baseada em Otimização Interativa para o Planejamento de Releases.....	119
6.3.10.	The Value of Exact Analysis in Requirements Selection	119
6.3.11.	A Multi Objective Teacher-Learning-Artificial Bee Colony (MOTLABC) Optimization for Software Requirements Selection.....	119
6.4.	Comparação com trabalhos relacionados	120
6.5.	Considerações finais	127
7.	Conclusão	129
7.1.	Trabalhos Futuros	132
	Referências Bibliográficas.....	134
A.	jMetal.....	140
A.1.	Arquitetura do jMetal	142
A.2.	Configurando um experimento com o jMetal.....	144
A.2.1.	Modelagem do problema	145
A.2.2.	Definição do experimento.....	146

INTRODUÇÃO

No campo da Engenharia de Software, não é uma idéia recente a adoção de processos de **desenvolvimento incremental** baseado em entregas iterativas do sistema, em que cada novo *release* é melhorado com o acréscimo de alguma nova funcionalidade, mapeada de um subconjunto de requisitos candidatos, que são de interesse para os stakeholders envolvidos no processo de desenvolvimento. Mesmo nas décadas de 1970 e 1980, um período amplamente dominado pela abordagem de desenvolvimento em cascata, grandes entusiastas de abordagens incrementais já existiam. De fato, apesar de muitos terem a percepção de que o desenvolvimento iterativo e incremental é uma prática “moderna”, suas origens e publicações podem ser rastreadas em várias décadas no passado (LARMAN e BASILI, 2003). Entretanto, apenas mais recentemente, com a popularização da Internet, que permitiu a criação de ferramentas que agrega facilidades a grupos de trabalho e coordenação remota dentro de comunidades de software de código-aberto, processos iterativos e incrementais se tornaram mais abrangentes dentro da indústria de software, provendo uma visão de desenvolvimento evolucionário de software que é incremental, iterativo, contínuo, interativo e sensível às circunstâncias sociais e organizacionais (SCACCHI, 2002).

Nos processos de desenvolvimento iterativo e incremental, uma das fases mais importantes é o processo de eliciação de requisitos, onde as necessidades das partes interessadas no sistema de software são mapeadas para um conjunto de requisitos candidatos a serem desenvolvidos na próxima iteração do sistema, ou então selecionados apenas posteriormente em alguma versão futura deste mesmo sistema. Entretanto, é largamente conhecido que, dentro de uma única iteração do processo de software, existem limites de tempo e orçamento, que, idealmente, não devem ser ultrapassados (SCACCHI, 2002). Deste modo, considerando um cenário em que o conjunto total de requisitos candidatos ultrapassam o orçamento disponível para o próximo release, a equipe de desenvolvimento deve lidar com o problema de decidir quais requisitos devem ser priorizados, selecionados e entregues na próxima iteração. Além do esforço necessário em negociar tais requisitos com os stakeholders, a equipe de desenvolvimento também encara dificuldades ao selecionar um subconjunto apropriado de requisitos candidatos que possam, da melhor maneira possível, balancear as exigências e restrições do projeto, tais como orçamento, custos de requisitos, preferências de clientes ou usuários finais, assim como a importância destes clientes para o projeto, mantendo

os custos do projeto sob controle e elevando o máximo possível o grau de satisfação percebido pelos stakeholders envolvidos no produto.

De modo a aliviar o esforço envolvido no processo de decisão, as informações relacionadas aos requisitos candidatos podem ser coletadas e quantificadas para torná-las possíveis de serem tratadas computacionalmente. Uma abordagem notável nesta direção é o **modelo custo-importância** (KARLSSON e RYAN, 1997). Este modelo tem como meta maximizar a qualidade de software, percebida através da quantização da percepção individual dos stakeholders em relação a cada requisito. Além disso, este modelo também possibilita minimizar os custos e o tempo de entrega. Nesta abordagem, um processo de revisão sistemática dos requisitos é conduzido conjuntamente com os stakeholders, realizando uma coleta de dados e gerando como saída um diagrama bidimensional que lida com os custos e importância de requisitos do ponto de vista destes stakeholders.

Os cinco passos necessários para utilizar a abordagem custo-importância são descritos a seguir. Inicialmente, os engenheiros de requisitos cuidadosamente realizam uma revisão dos requisitos candidatos (*i*), de forma que eles sejam descritos em sua completude e de maneira não-ambígua. Para obter a avaliação de importância dos requisitos (*ii*), o método AHP é sugerido pelos autores como a técnica ideal para captar uma estimativa da prioridade de cada requisito candidato. O método AHP é novamente aplicado (*iii*) sobre os requisitos candidatos mas, desta vez, ele é utilizado com a participação dos engenheiros de software responsáveis por captar uma estimativa de custo de cada requisito candidato. O próximo procedimento é mapear estes resultados de custo e importância em um diagrama bidimensional (*iv*), conforme observamos na Figura 1. O último procedimento (*v*) envolve a participação dos stakeholders, em que eles utilizam o diagrama como um mapa conceitual para analisar e discutir os requisitos candidatos. Finalmente, o resultado desta discussão será de fundamental importância para que o gerente de projeto possa decidir quais requisitos deverão ser priorizados, assim como também planejar a entrega do requisito. Note que o diagrama apresentado na Figura 1 está subdividido em três regiões classificadas como *alto*, *médio* e *baixo*. Os requisitos na primeira região contém uma razão importância-custo maior que 2. Os requisitos de proporção média de importância em relação ao custo necessário compreendem valores entre 0,5 e 2. E, finalmente, os requisitos de baixa razão média de importância-custo possuem uma avaliação menor que 0,5. Isto serve para facilitar a tomada de decisão dos gerentes de projeto ao priorizar requisitos com uma alta proporção importância-custo e descartar os requisitos que possuem um baixo retorno de importância em relação ao quanto deve ser investido para sua implementação.

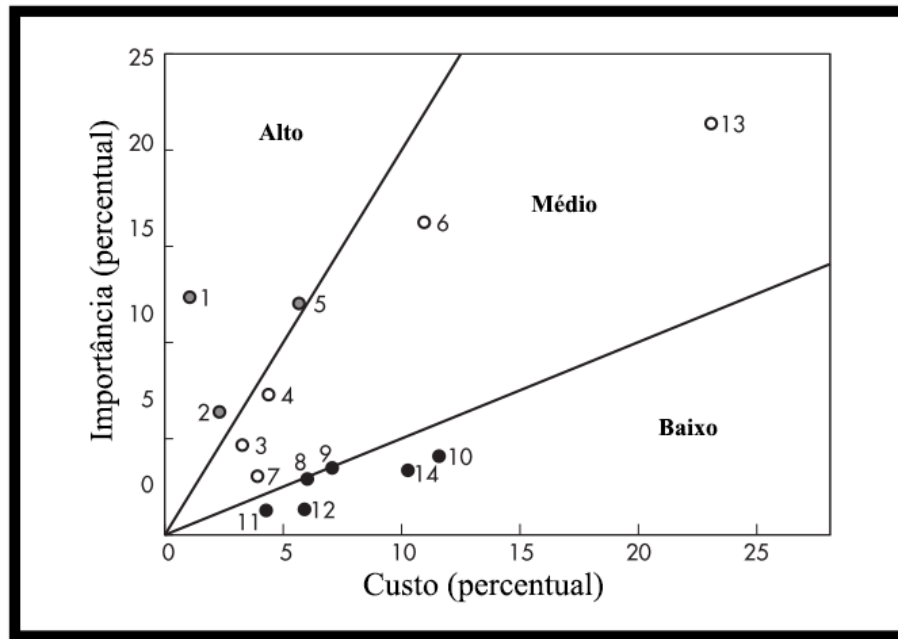


Figura 1 - Diagrama importância-custo contendo 14 requisitos candidatos, obtidos com base em um estudo de caso (KARLSSON e RYAN, 1997).

Posteriormente, no início da década de 2000, o processo de seleção de requisitos foi definido e modelado como o **Problema do Próximo Release** (também conhecido como NRP ou *Next Release Problem*) (BAGNALL, RAYWARD-SMITH e I. M. WHITTLEY, 2001). Na literatura, diversas propostas que caracterizam e solucionam este problema de diferentes e complementares pontos de vista podem ser encontrados (DURILLO *et al.*, 2011) (RUHE e GREER, 2003) (LI *et al.*, 2014) (COLARES *et al.*, 2009) (BAKER *et al.*, 2006). Inicialmente, os avanços principais são representados por propostas que possuem um maior foco em encontrar a melhor solução que mantém o orçamento do projeto sob controle ao mesmo tempo que a satisfação dos stakeholders é aumentada. Posteriormente, o conceito de um limite de orçamento pré-fixado foi descartado em favor de oferecer não apenas uma única solução, mas um conjunto de boas soluções candidatas com diferentes graus de custo e satisfação, tornando o processo de tomada de decisão mais flexível e personalizável (DURILLO *et al.*, 2011). Esta nova abordagem é denominada Problema do Próximo Release Multi-Objetivo (MONRP ou *Multi-Objective Next Release Problem*).

1.1. Motivação e Justificativa

Apesar das relevantes contribuições e do grande avanço representado pelas abordagens MONRP, podemos notar na literatura que estas propostas mantêm um foco no modelo tradicional de avaliar o orçamento do projeto, custos de requisitos, preferências de clientes e sua relativa importância para o projeto, ignorando outros importantes aspectos, tais como riscos

de software, incluindo aqueles associados ao projeto de software, produto de software e organização, que podem impactar profundamente, de maneira positiva ou negativa, nas avaliações finais de custos de requisitos e satisfação de clientes para o próximo release.

A incorporação de uma análise de risco na modelagem de um problema do próximo release aparenta ser uma importante contribuição e evolução. Tal linha de raciocínio pode ser reforçada pelo fato que, por sua vez, a adoção de um processo de gerenciamento de risco é um fator importante associado ao sucesso de um projeto de software (ALAM, 2014) e, de maneira contrária, a ausência deste tipo de processo é um fator significativo relacionado ao fracasso de projetos de software (VERNER, SAMPSON e CERPA, Junho 2008), visto que esta ausência obstrui a visão de gerentes de projeto e suas respectivas equipes de desenvolvimento em avaliar pontos de falha específicos em seus projetos de software. Por exemplo, um estudo realizado sobre 50.000 projetos de software concluídos, revela que 53% destes projetos incorreram em algum problema de entrega, incluindo um estouro de orçamento médio em 56% (MASTICOLA, 2007). Adicionalmente, 18% destes projetos foram cancelados e, portanto, não apresentaram resultados. Portanto, a avaliação de custos de requisitos deve ser mantida na modelagem e planejamento do próximo release. Como consequência, os custos de riscos e os valores associados à gestão de riscos devem ser consideradas simultaneamente. O autor sugere que o fator-chave em desenvolver uma atividade eficaz de gerenciamento de riscos é expressá-lo em termos econômicos. Além disso, o próprio ato de medição e controle do processo é em si uma atividade de risco, que também é um fator desejável enquanto ele puder ser realizado de maneira harmônica com as demais atividades do processo. Considerando o custo esperado de um projeto de software, ao agrupar as duas categorias de falha de projeto, em que 53% dos projetos incorreram em algum problema de entrega e 18% foram cancelados antes de apresentar qualquer resultado, totalizam um percentual de 71% de projetos com falha. Além disso, dados de 2004 também informam que, dentro destes projetos que fracassaram, o orçamento é excedido com uma média de 54% em relação à sua projeção de custo inicial. Com posse destes dados, o autor propõe um modelo probabilístico em que é possível estimar a eficácia de técnicas de gerenciamento de risco que possam aumentar a chance de sucesso no projeto. Este modelo, apesar de sinalizar que a utilização de boas práticas de gerenciamento de risco incorrem em um custo adicional que representam uma fração do orçamento, caso elas melhorem as chances de sucesso do projeto por um percentual significativo, elas acabam por representar um custo menor do que o caso em que o orçamento fosse estourado, o que compensaria o gasto empregado no uso destas técnicas de gerenciamento de riscos.

Outro estudo elenca 57 fatores de falha possíveis dentro de 70 projetos de software que não obtiveram sucesso (VERNER, SAMPSON e CERPA, Junho 2008), dentre os três fatores mais críticos que foram encontrados, em primeiro lugar destaca-se o atraso na entrega, que correspondeu a 93% dos projetos. O segundo fator foi a subestimação do orçamento, que ocorreu em 81% destes projetos. Finalmente, o terceiro fator mais importante foi a falha no gerenciamento de riscos, controle ou reavaliação de riscos, que representou 76% dos projetos que fracassaram. Portanto, novamente se reafirma a necessidade de um modelo de gerenciamento de riscos adequado para a entrega bem sucedida de um produto de software.

Um risco em um projeto de software pode ser identificado como uma ameaça em atender os objetivos esperados de um projeto (BANNERMAN, 2008). Na maioria dos casos, o risco é visto como uma consequência negativa para o projeto, apesar que, em sua formulação clássica, o risco também recebe uma definição positiva. Ou seja, ao tomar uma determinada decisão, o gerente de projeto pode acarretar uma consequência positiva ou negativa de acordo com o que era esperado. Além disso, caso este evento de risco venha a ocorrer, diferentes estratégias para responder a este evento podem ser empregadas. Estas estratégias podem se tratar de evasão de possíveis eventos de risco, o que nem sempre é possível de ser feito. O tratamento deste risco pode ser transferido para uma organização externa. Aceitação deste risco e especialmente e mitigação deste risco envolvem atividades proativas e internas à equipe de desenvolvimento em que eles devem tomar uma série de ações para evitar a ocorrência deste risco ou, caso ele ocorra, diminuir suas consequências negativas o máximo possível.

1.2. Objetivos

Nesta direção, considerando a relevância do gerenciamento de riscos, este trabalho tem como objetivo principal propor uma abordagem baseada em riscos para a seleção de requisitos no contexto do problema do próximo release multi-objetivo. Sendo o primeiro trabalho em que a análise de riscos é incorporada em ambas as medições de custo e satisfação para um problema de próximo release.

Na abordagem proposta, os requisitos candidatos são associados aos riscos de software identificados, que, por sua vez, são relacionados às técnicas de mitigação de riscos. Portanto, a proposta também possui como novidade a inclusão de dados referentes à técnicas de tratamento de riscos em seu modelo.

Foi observado também nos trabalhos relacionados que, apesar das avaliações de probabilidade e consequência de um evento de risco serem indicadores clássicos para

monitoramento de riscos, nenhuma proposta relacionada utiliza estes dados explicitamente. Assim, a abordagem aqui proposta também se trata da primeira contribuição que inclui estes dados a respeito de riscos. Estes valores, juntamente com os custos de aplicação de cada técnica de mitigação, são responsáveis por estimar o impacto dos riscos nos custos dos requisitos e na satisfação dos clientes.

O problema do próximo release clássico foi descartado em favor de uma abordagem do problema do próximo release multi-objetivo pois, ao observarmos o histórico das propostas para este tipo de problema (PITANGUEIRA, MACIEL e BARROS, 2015), observamos nitidamente que a modelagem multi-objetiva representa o estado da arte na Engenharia de Software Baseada em Buscas (do inglês, *Search Based Software Engineering* - SBSE).

Além disso, como forma de conciliar e balancear conflitos de interesses entre clientes, de forma similar a abordagens existentes, a abordagem proposta também considera as preferências dos clientes e a importância dos mesmos do ponto de vista da organização de desenvolvimento. Portanto, o modelo proposto não descarta informações a respeito da importância de clientes e custos de requisitos, pois observamos que o orçamento está relacionado a um dos fatores de risco do projeto e a satisfação de clientes é um bom índice capaz de avaliar o grau de sucesso no subconjunto de requisitos candidatos que foram selecionados para o próximo release. Além disto, conforme descrito anteriormente, o modelo custo-satisfação já se trata de um formato consolidado na literatura.

Posteriormente, considerando os trabalhos relacionados, que de alguma forma incorporam informações de riscos ou alguma outra medição similar, o presente trabalho também tem por objetivo destacar as boas práticas adotadas por tais trabalhos relacionados e compará-las com a abordagem proposta neste trabalho.

Por fim, sabemos também que uma simples comparação teórica não é suficiente para validar o modelo. Portanto, este trabalho também tem por objetivo realizar experimentos comparando as principais metaheurísticas de busca multi-objetivas disponíveis na literatura, evidenciando a aplicabilidade prática da abordagem proposta e a eficiência das metaheurísticas em relação a diferentes indicadores de qualidade. Apesar do modelo clássico de próximo release definir o relacionamento de pré-requisitos, em que um dado requisito candidato pode exigir que um ou mais outros requisitos candidatos também sejam implementados, os experimentos propostos não utilizam a definição de pré-requisitos. Isto não foi empregado por não se tratar do foco da contribuição do modelo, além que a definição de uma função de restrição já se trata de um problema bem resolvido na literatura (BAGNALL, RAYWARD-SMITH e I. M.

WHITTLEY, 2001), em que as soluções candidatas de seleção de requisitos que não respeitam esse relacionamento são removidas do conjunto, restringindo o espaço de busca.

1.3. Organização do Trabalho

O trabalho está organizado da seguinte forma:

- No **Capítulo 2**, são apresentados os fundamentos para a formulação clássica para um problema do próximo release para em seguida apresentarmos a reformulação deste problema em uma abordagem multi-objetivo. Em sequência, apresentamos os conceitos envolvidos em um problema de otimização multi-objetivo, destacando dois indicadores de qualidade: hipervolume e espalhamento. Finalmente, o capítulo se encerra com a descrição de um processo de gerência de riscos, permitindo pavimentar o caminho para a modelagem de um problema de próximo release multi-objetivo incorporando riscos.
- No **Capítulo 3**, três metaheurísticas capazes de solucionar problemas de otimização multi-objetivo são apresentadas em detalhes, mas antes, observamos que estes algoritmos são uma variação dos algoritmos genéticos clássicos. Deste modo, apresentamos também os fundamentos envolvidos na utilização de algoritmos genéticos, destacando a adaptação para um contexto de múltiplos objetivos.
- No **Capítulo 4**, a abordagem baseada em riscos para o problema do próximo release multi-objetivo é descrita em detalhes. Observamos os principais conceitos envolvidos que irão compor os parâmetros de entrada do algoritmo, assim como também descrevemos as equações envolvidas na resolução do problema, organizadas em dois objetivos a serem otimizados, custo e satisfação dos clientes.
- No **Capítulo 5**, os resultados de dois experimentos distintos são apresentados. O primeiro experimento observa o comportamento dos resultados em comparação com o que foi proposto no modelo. O segundo experimento utiliza duas bases de dados semi-reais que são avaliadas em relação aos três algoritmos de otimização multi-objetivo apresentados no Capítulo 3, permitindo a avaliação dos resultados em relação a estas duas bases de dados para o problema de próximo release multi-objetivo.
- No **Capítulo 6**, levantamos o estado da arte com os principais trabalhos relacionados ao tema proposto em formato de revisão sistemática. Estes trabalhos então são compilados em uma tabela afim de facilitar a análise comparativa.

- O **Capítulo 7** apresenta as conclusões do trabalho, observando os objetivos estabelecidos na introdução em relação aos resultados obtidos nos experimentos executados. Por fim, o capítulo propõe possíveis aperfeiçoamentos do modelo proposto, assim como possíveis futuros desdobramentos para novas pesquisas relativas ao tema do trabalho.
- No **Apêndice A**, apresentamos uma implementação e cenário de utilização destes três algoritmos em um framework em Java denominado jMetal. Observamos também as vantagens em utilizar um framework especializado em prover soluções para estes problemas de otimização em substituição a reimplementar estes algoritmos separadamente

2. FUNDAMENTAÇÃO TEÓRICA

A entrega de um sistema de software útil e que atenda às necessidades dos stakeholders envolve a análise de uma grande quantidade de informações referentes aos requisitos candidatos para integrar este sistema. Esta quantidade de informações se torna ainda maior de modo diretamente proporcional ao tamanho e complexidade do projeto de software a ser desenvolvido. Esta complexidade envolve uma grande quantidade de requisitos a serem elicitados e avaliados em um processo de Engenharia de Requisitos, no qual aspectos técnicos e não-técnicos também são levados em consideração (FOGELSTRÖM *et al.*, 2009). Finalmente, todos estes dados devem ser mapeados de alguma maneira em uma análise de custo-benefício capaz de informar os tomadores de decisão dos riscos e oportunidades envolvidos neste empreendimento. É importante notar também que, por serem baseados em estimativas, esta análise possui uma incerteza intrínseca limitada pela quantidade de informação disponível naquele momento.

Ao admitirmos a utilização de uma abordagem incremental, nem todos os requisitos candidatos a serem entregues na próxima versão do sistema poderão ser entregues em uma única iteração. Portanto, será necessário definir um subconjunto de requisitos que maximize a percepção de satisfação dos clientes ao mesmo tempo que minimiza os custos e antecipa o tratamento de riscos críticos envolvidos nesta seleção.

Em um primeiro momento, podemos admitir um processo de seleção informal destes requisitos, porém, foi demonstrado que a falta de um processo apropriado de gerenciamento e rastreamento de requisitos gera uma solução insatisfatória para clientes, desenvolvedores e usuários (KARLSSON e RYAN, 1997). Deste modo, a informação coletada a respeito dos requisitos deve ser organizada de uma maneira clara, não-ambígua de modo a se tornar capaz de fornecer um feedback mais eficaz na alocação de recursos do projeto, além de estabelecer relações de custo-benefício em objetivos conflitantes, tais como qualidade, custo ou *time-to-market*. Este último objetivo é de grande importância em uma abordagem incremental que admite entregas rápidas e frequentes, deste modo os requisitos devem ser classificados de acordo com alguma medida de prioridade em relação aos demais, sendo assim alocados nas primeiras versões do sistema.

Uma série de técnicas clássicas de priorização de requisitos estão consolidados na literatura (BERANDER e ANDREWS, 2005) (FERNANDES e MACHADO, 2016), sendo brevemente descritas a seguir. O **processo hierárquico analítico** (do inglês, Analytical

Hierarchical Process - AHP) é um processo sistemático de tomada de decisão para realizar a tarefa de priorizar os requisitos candidatos. Este método é conduzido realizando uma comparação de todos os pares de requisitos classificados hierarquicamente. Deste modo, é possível estabelecer relações de quais requisitos são mais importantes em relação aos demais e em que grau de importância eles se diferenciam entre si. Apesar da grande precisão deste método, ele possui uma desvantagem natural em exigir $n \times (n - 1)/2$ comparações, onde n é o número de requisitos candidatos. Portanto, para uma grande quantidade de requisitos, este procedimento manual, que conta com a participação de todos os stakeholders interessados na continuidade do processo, se torna bastante oneroso.

Uma segunda técnica, de **votação cumulativa**, também denominada de **teste dos 100 dólares**, consiste em fornecer 100 pontos para cada *stakeholder*. Por sua vez, cada *stakeholder* deve atribuir pontos aos requisitos de maior interesse. Apesar de ser uma técnica mais simples em comparação com a anterior, ela enfrenta dificuldades em lidar com uma grande quantidade de requisitos, pois os pontos se tornam mais escassos, e, ao mesmo tempo, um *stakeholder* pode cometer um erro de cálculo e não distribuir todos os 100 pontos disponíveis para todos os requisitos, dificultando a interpretação da pontuação atribuída por diferentes stakeholders. Portanto, é aconselhável que esta técnica seja utilizada com auxílio de alguma ferramenta automatizada. Outra desvantagem é que esta técnica pode ser aplicada apenas uma vez, visto que uma repetição do teste pode fazer com que o stakeholder seja enviesado de modo a atribuir mais pontos para os requisitos de seu maior interesse que não foram bem avaliados por seus pares.

Outro método possível e que é o mais comumente utilizado se trata da técnica de **agrupamento**. Os requisitos são agrupados em categorias, do mais importante ao menos importante. Por uma questão de simplificação, normalmente essas categorias recebem nomes tais como “crítico”, “padrão”, “opcional”. Outros termos podem ser utilizados, caso as definições destes termos se tornem bastante claras para os stakeholders. Uma dificuldade nesta abordagem é decorrente do fato que os stakeholders possuem uma tendência em agrupar como “crítico” uma grande quantidade de requisitos que lhes pareçam ser interessante. Deste modo, existem casos em que um limite de requisitos por categoria é definido para resolver este problema. Uma limitação desta abordagem é que requisitos encaixados numa mesma categoria não possuem distinção de prioridade entre si.

A técnica de **ranqueamento** é um método simples em que os n requisitos candidatos são ordenados de 1 (o mais importante) até n (o menos importante). Esta técnica simples é mais

adequada para aplicar a apenas um stakeholder pois, no caso de existir mais de um stakeholder, ainda permanece o desafio de distinguir as diferentes atribuições de prioridade dentre eles. É possível realizar uma média do valor de ranqueamento entre todos os stakeholders, mas isso pode ocasionar situações em que requisitos estejam empatados, algo que o ranqueamento busca evitar.

A técnica dos **top 10 requisitos** é um bom método em ocasiões em que todos os stakeholders são igualmente importantes para o processo. Cada membro envolvido deve consultar a lista de requisitos candidatos e elencar seus 10 requisitos mais importantes para si, sem necessidade de estabelecer uma ordem de prioridade nesta lista. Deste modo, os requisitos mais recorrentes em todas estas listas podem ser priorizados para entrega. Isto pode ocasionar problemas, já que stakeholders podem se tornar insatisfeitos caso menos requisitos de sua lista sejam entregues em comparação com os demais. Assim, é desejável realizar um balanceamento entre as listas, sem se ater à rigidez da metodologia proposta.

Em tais cenários, conforme apresentados anteriormente, sem o suporte de abordagens sistemáticas de tomadas de decisão, o processo de seleção de requisitos se torna uma tarefa complexa, desafiadora e propensa à erros. Portanto, a adoção de abordagens manuais, ad-hoc são inadequadas para selecionar os requisitos que maximizam a satisfação dos stakeholders ao mesmo tempo que minimizam os custos do projeto. Tal inexequibilidade é percebida especialmente nos casos em que existam uma grande quantidade de informações relativa ao conjunto de requisitos candidatos. Adicionalmente, a existência de conflitos de interesse entre stakeholders, impõe à equipe de desenvolvimento a responsabilidade e o esforço brutal de conciliar e balancear estes *trade-offs* durante o processo de tomada de decisão. Estas dificuldades aumentam de maneira exponencial a medida que a quantidade de requisitos a serem contemplados se tornam cada vez maiores. Observemos que, para qualquer conjunto contendo n requisitos candidatos, as possibilidades de seleção de subconjuntos de requisitos a serem de fato entregues é equivalente à 2^n . Em outras palavras, o simples processo de escolher requisitos, sem antes levarmos em consideração como a equipe deverá lidar com quaisquer restrições de projeto ou conflitos de interesses já é uma tarefa inerentemente complexa dado um conjunto suficientemente grande de requisitos.

A medida custo-satisfação representa um indicador comumente utilizada para classificar os requisitos de software (KARLSSON e RYAN, 1997), conforme descrito no capítulo anterior. Dentro desta medida, é esperado que o subconjunto de requisitos selecionados para entrega devem apresentar a máxima qualidade, o menor custo possível, ao mesmo tempo

que a nova versão do sistema deve ser entregue o mais rapidamente possível. Deste modo, os requisitos são avaliados por engenheiros de requisitos, conjuntamente com os clientes e usuários do sistema para levantar informações a respeito da importância e estimativa de custo para estes requisitos. Tradicionalmente, os requisitos são então dispostos em um diagrama bidimensional de custo e satisfação de forma a dar suporte à discussão a respeito de quais destes requisitos devem ser selecionados para a próxima versão do sistema.

Um grande problema desta seleção manual de requisitos é que, conforme a quantidade de requisitos for aumentando, se torna cada vez mais difícil analisar cada possível seleção de requisitos para o sistema dentro de um tempo hábil. Podemos notar que o problema cresce em uma proporção de 2^n combinações possíveis, considerando ‘ n ’ a quantidade de requisitos disponíveis para seleção. Deste modo, é desejável obter uma técnica automatizada capaz de analisar estas possibilidades de seleção e retornar alguma informação a respeito da análise de custo-satisfação destes subconjuntos de requisitos.

Apesar do ganho em agilidade no uso de ferramentas computacionais para observar as possibilidades de subconjuntos de requisitos, este problema pode se tornar intratável até mesmo em uma busca exaustiva, caso o conjunto de requisitos se torne muito grande. Este problema de seleção de requisitos está relacionado a um problema clássico de otimização combinatória denominada *knapsack problem*, que possui complexidade computacional *NP-hard* (BAGNALL, RAYWARD-SMITH e I. M. WHITTLEY, 2001). Deste modo, se justifica o uso de algum método de busca heurística de forma a reduzir o tempo de execução do algoritmo e que, apesar de não existir a garantia de que a solução ótima para o problema seja encontrada, é possível encontrar uma solução aceitável para os tomadores de decisão.

Neste contexto, este problema de busca foi formalizado como sendo o Problema do Próximo Release (*Next Release Problem*) (BAGNALL, RAYWARD-SMITH e I. M. WHITTLEY, 2001), no qual os requisitos são definidos em razão de sua estimativa de custo e a satisfação percebida pelos clientes na escolha de cada um. Portanto, o Problema do Próximo Release consiste em escolher o melhor subconjunto de requisitos que maximize a satisfação do cliente sem estourar o orçamento alocado para aquela versão do sistema.

Desde sua formalização em 2001, o problema do próximo release evoluiu no aspecto de objetivos avaliados (PITANGUEIRA, MACIEL e BARROS, 2015). Inicialmente, a maioria dos trabalhos adotavam uma abordagem mono-objetiva ou, em outras palavras, o orçamento do projeto é pré-fixado para cada release. Porém, a partir de 2007 (ZHANG,

MANSOURI e HARMAN, 2007) uma nova abordagem multiobjetiva passou a definir uma crescente tendência nas abordagens propostas para o problema. Neste cenário, o orçamento não é mais fixado, mas um conjunto de boas soluções é fornecida para diferentes valores de orçamento para flexibilizar a possibilidade de tomada de decisões. Porém, como podemos observar (PITANGUEIRA, MACIEL e BARROS, 2015), pouco se fala a respeito das incertezas envolvidas nos indicadores de qualidade utilizados para avaliar as soluções.

2.1. Problema do Próximo Release

O Problema do Próximo Release é definido na forma tal que os requisitos são mapeados em um conjunto R , representando todas as possíveis melhorias para o sistema. Dado o conjunto C de clientes, cada cliente j possui interesse em um determinado subconjunto de requisitos candidatos, este subconjunto é dado por $R_j \subseteq R$. Além disso, cada cliente j possui um peso $w_j \in \mathbb{Z}^*$, denotando a importância relativa daquele cliente para a empresa. Além disso, para cada requisito $r \in R$ existe um custo associado, na forma $\text{custo}(r) \in \mathbb{Z}^+$.

O problema pode apresentar restrições caso existam dependências entre requisitos. Isto se dá na forma de um grafo acíclico $G = (R, E)$ tal que $(r, r') \in E$ se e somente se r é um pré-requisito de r' . Esta relação também é transitiva, visto que para cada pré-requisito r de r' , cada um destes pré-requisitos podem conter um conjunto E de pré-requisitos. Finalmente, este grafo de dependências é ilustrado na Figura 2. Apesar das Equações 1 e 2 não tratarem dependências entre requisitos, é possível definir uma função de restrição adicional para resolver este problema, verificando se o subconjunto de requisitos candidatos selecionados para uma dada solução constituem uma solução válida, observando as dependências entre requisitos.

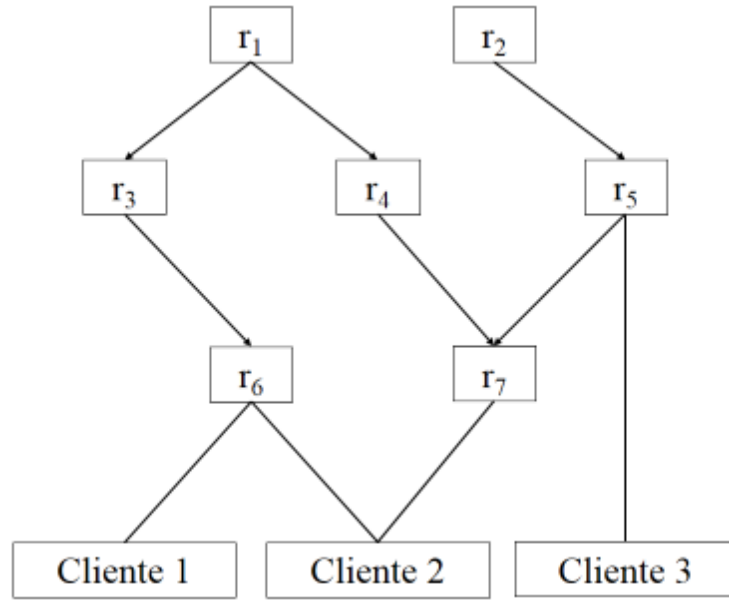


Figura 2 - Representação de dependências entre requisitos (BAGNALL, RAYWARD-SMITH e I. M. WHITTLEY, 2001).

Os clientes podem possuir interesses distintos nos requisitos. Portanto, ao considerarmos cada subconjunto R_j de requisitos de interesse para o cliente j , se considerarmos que cada requisito individual o cliente j atribui diferentes graus de importância para cada requisito, definido por $v_{i,j} \in \mathbb{Z}^+$ que atribui um valor numérico, responsável por representar o quão importante é um dado requisito i para o cliente j . Deste modo, é possível criar uma matriz V de valores de importância de requisitos. para Finalmente, o problema deve sinalizar se cada requisito ' i ' será ou não selecionado para aquela versão do sistema, uma variável binária x_i é responsável por representar este mapeamento. Com todos estes valores em mãos e com a definição de um orçamento B alocado para esta versão, podemos definir as funções de avaliação para o problema do próximo release nas Equações 1 e 2.

$$\sum_{i \in R} \text{custo}(r_i) \cdot x_i \leq B \quad (1)$$

$$\sum_{j \in C} w_j \cdot \sum_{i \in R} v_{i,j} \cdot x_i \quad (2)$$

Conforme podemos observar nestas equações, a meta é maximizar o valor final da Equação 2 ao mesmo tempo que a soma dos custos de cada requisito i selecionado na Equação 1 não deve ultrapassar o valor de orçamento B fixado para o problema.

2.2. Problema do Próximo Release Multiobjetivo

Para cálculo de um problema do próximo release biobjetivo, as Equações 1 e 2 devem ser reformuladas, conforme descrito nas Equações 3 e 4. Note que o valor de orçamento B não está mais presente na função custo.

$$\text{minimizar } f_1 = \sum_{i \in R} \text{custo}(r_i) \cdot x_i \quad (3)$$

$$\text{maximizar } f_2 = \sum_{j \in C} w_j \cdot \sum_{i \in R} v_{i,j} \cdot x_i \quad (4)$$

Do ponto de vista da execução do problema em um algoritmo multiobjetivo, a função de maximização f_2 pode ser convertida para um função de minimização $-f_2$.

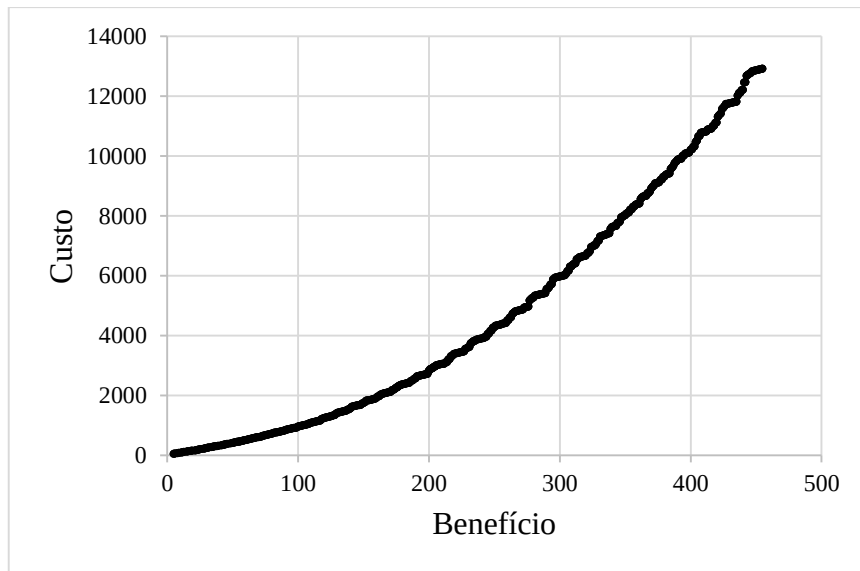


Figura 3 - Resultado de uma avaliação biobjetiva.

Como podemos observar na Figura 3, a saída do algoritmo não entrega mais um único ponto dentro de parâmetros pré-fixados, mas um vetor de soluções para as funções de avaliação sem necessidade de reiniciar a execução do algoritmo modificando os parâmetros de entrada para o problema. Isto permite uma maior flexibilização de parâmetros, tal como o valor de orçamento no caso específico do problema do próximo release. Deste modo, o tomador de decisão terá um maior leque de possibilidades de escolha, sem estar tão restrito ao valor final de saída do algoritmo, tornando-o mais amigável caso o algoritmo seja utilizado como um programa de suporte à decisão.

Para melhor compreendermos a resolução de um problema multi-objetivo, precisamos introduzir uma série de novos conceitos, que até então não eram necessários para a formulação clássica de um problema do próximo release. Com o amadurecimento de pesquisas que fazem uso de uma abordagem multiobjetiva (PITANGUEIRA, MACIEL e BARROS, 2015), os conceitos envolvidos na formulação de um problema do próximo release com múltiplos objetivos se encontram bem consolidados na literatura.

Para um problema de otimização multiobjetivo (DURILLO *et al.*, 2011), considere um vetor $\vec{x}^* = [x_1^*, x_2^*, \dots, x_n^*]$ de variáveis de decisão que satisfaçam as restrições definidas para o problema, ao mesmo tempo que **minimize** o vetor de funções objetivo $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \dots, f_k(\vec{x})]^T$, onde $\vec{x} = [x_1, x_2, \dots, x_n]^T$. O conjunto de todas as possíveis combinações de valores para $\vec{x}$ que satisfaça as restrições definidas estão contidas na região Ω do espaço de busca.

Neste cenário, o objetivo não se trata apenas de encontrar uma única solução que atenda aos parâmetros pré-determinados para o problema, mas encontrar um conjunto com as melhores soluções que minimizem o vetor de funções objetivo. Para isto, precisamos comparar as soluções encontradas e definir o quão boa esta solução é em relação às demais, isto é dado a partir do conceito de dominância de soluções. Uma solução $\vec{x}' = [x'_1, x'_2, \dots, x'_n]$ domina a solução $\vec{x}'' = [x''_1, x''_2, \dots, x''_n]$ se e somente se $f_i(\vec{x}') \leq f_i(\vec{x}'')$, $\forall i = 1, 2, \dots, k$. Além disso, deve existir pelo menos um objetivo j ($1 \leq j \leq k$) tal que $f_j(\vec{x}') < f_j(\vec{x}'')$. Em outras palavras, $\vec{x}'$ deve apresentar valores de função menores ou iguais para todos os objetivos em comparação com $\vec{x}''$, mas deve garantir que pelo menos uma das funções objetivos apresente uma avaliação melhor que $\vec{x}''$. Caso esta condição não seja satisfeita, podemos definir que as soluções são não-dominantes entre si. Uma observação a ser feita é que, caso seja necessário realizar uma **maximização** de um dado vetor de funções, basta inverter os sinais de desigualdades nas duas regras de dominância de forma a comparar as soluções obtidas, respectivamente para “maior ou igual que” e “maior”. Por definição, caso uma dada solução $\vec{x}'$ domine outra solução $\vec{x}''$, o operador de dominância é dado pela expressão $\vec{x}' \preceq \vec{x}''$ afim de representar esta relação de dominância entre pares de soluções, formando o conceito de **dominância de Pareto**.

O conjunto de soluções que não são dominadas por nenhuma outra solução da região Ω formam o **conjunto ótimo de Pareto**. A busca por este conjunto ótimo de Pareto é o objetivo de um problema de otimização multiobjetivo (*MOP - Multi-objective Optimization Problem*). O conjunto ótimo de Pareto é denotado por $P^* = \{\vec{x} \in \Omega | \nexists \vec{x}' \in \Omega, \vec{x}' \preceq \vec{x}\}$, onde Ω é o espaço de busca que contém todas as possíveis soluções para um dado problema. Este

conjunto é representado no espaço de busca como sendo uma **fronteira de Pareto** definida por $PF^* = \{\vec{f}(\vec{x}) | \vec{x} \in P^*\}$. Caso estejamos lidando com um problema de otimização que contém duas funções de objetivo a serem avaliadas, esta fronteira de Pareto representa uma curva em um espaço bidimensional. Considerando que o conjunto de soluções apresenta um conjunto finito de pontos neste espaço, estes pontos devem se aproximar o máximo quanto for possível do conjunto ótimo de Pareto.

2.2.1. Indicadores de Qualidade

Realisticamente falando, algoritmos de busca heurística não garantem que o conjunto ótimo de Pareto seja encontrado, de maneira análoga à busca pelo ponto ótimo global em um problema tradicional mono-objetivo. Outra questão importante diz respeito ao espalhamento destes pontos, de forma a reproduzir a curva de Pareto da maneira mais fiel possível. Um bom algoritmo de busca multi-objetivo deve se aproximar o máximo possível do conjunto ótimo de Pareto, ao mesmo tempo que realiza um bom espalhamento dos pontos ao longo da curva encontrada para o problema. A Figura 4.a ilustra uma solução que se aproxima do conjunto de otimalidade de Pareto, mas possui um espalhamento ruim. Na Figura 4.b, apesar do conjunto de soluções não estar próximo à otimalidade, apresenta um bom espalhamento uniforme ao longo da curva. Finalmente, o terceiro gráfico na Figura 4.c descreve um cenário ideal com um conjunto de soluções que se aproxima do conjunto ótimo de Pareto e que também apresenta um bom espalhamento uniforme ao longo da curva.

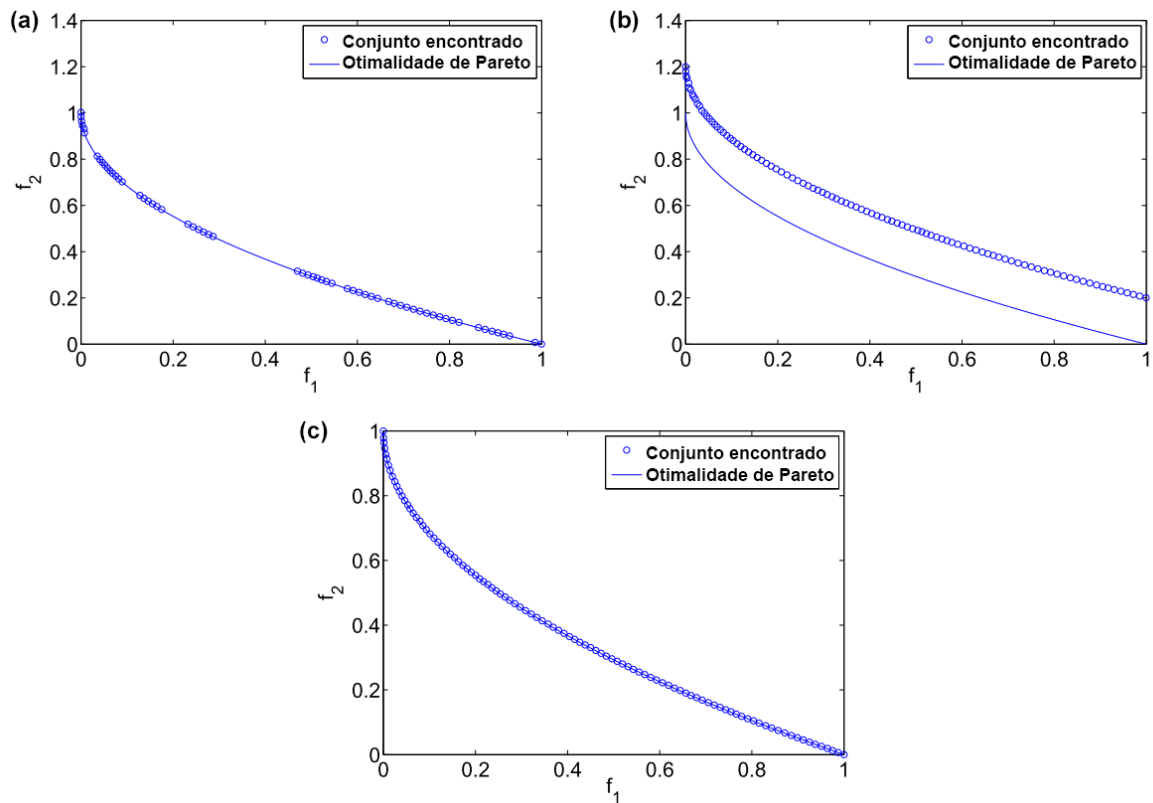


Figura 4 - Representação dos problemas de espalhamento (a), otimalidade (b) e o cenário ideal de soluções (c) (DURILLO *et al.*, 2011).

É intuitivo observarmos que uma inspeção visual não é a solução ideal para determinarmos a qualidade de um conjunto de soluções obtidos para um problema qualquer para múltiplos objetivos. Deste modo, é necessário obtermos algum indicador de qualidade responsável por quantificar estes conceitos e nos servir de um demonstrativo a respeito de quão bom é o conjunto de soluções encontrada, permitindo estabelecer uma comparação. Esta comparação é de especial utilidade nas situações em que se deseja avaliar diferentes metaheurísticas para um dado problema, de forma a estabelecer qual algoritmo de otimização multiobjetivo é capaz de fornecer o melhor resultado para o problema.

2.2.1.1. Espalhamento

O indicador de **espalhamento** (Δ) é uma medida de diversidade do conjunto de soluções, responsável por demonstrar a extensão deste espalhamento do conjunto dado por uma fronteira de Pareto (DEB, 2001), conforme observamos na Figura 5.

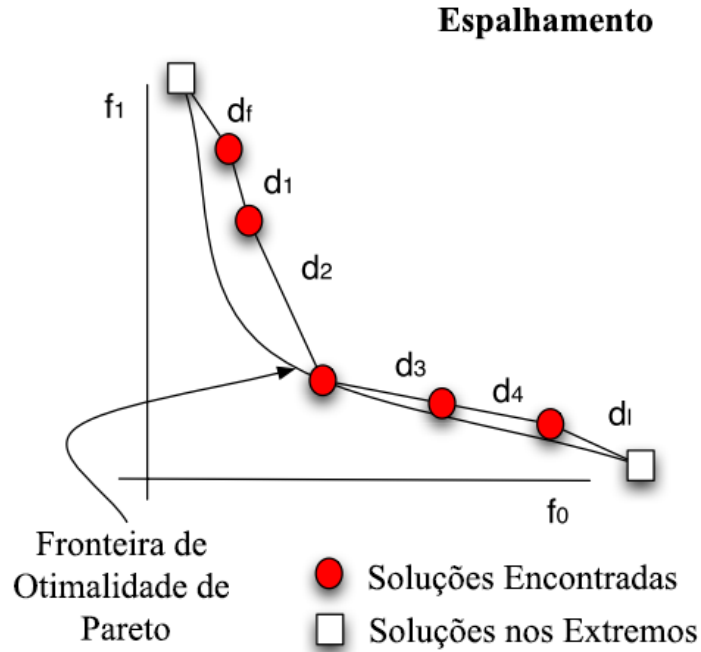


Figura 5 - Cálculo do espalhamento para dois objetivos f_0 e f_1 (DURILLO *et al.*, 2011).

O cálculo do espalhamento é dado na Equação 5. Temos d_i a distância Euclidiana entre pontos consecutivos da fronteira, enquanto que $\bar{d}$ assume o valor médio entre estas distâncias. Finalmente, temos d_f e d_l como sendo as distâncias Euclidianas para os pontos extremos do conjunto ótimo de Pareto (DURILLO *et al.*, 2009). Esta medida de desempenho recebe uma avaliação igual a zero para uma distribuição ideal, denotando um perfeito espalhamento para as soluções da fronteira. Esta métrica pode ser aplicada após a normalização das funções objetivos dentro do intervalo $[0, 1]$. Portanto, as fronteiras de Pareto encontradas que obtiverem um valor de espalhamento menor em relação as demais são consideradas melhores.

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - \bar{d}|}{d_f + d_l + (N - 1) \cdot \bar{d}} \quad (5)$$

Para obtermos um valor ideal de espalhamento igual a zero, a curva encontrada deve satisfazer duas condições (DEB, 2001): a primeira condição define que os valores para ' d_f ' e ' d_l ' assumem um valor igual a zero, ou seja, a posição do primeiro e último pontos da curva de Pareto são exatamente iguais aos valores extremos da Fronteira de Pareto. A segunda condição define que a curva possui uma distribuição ideal, ou seja, o espaçamento entre dois pontos consecutivos da curva possuem sempre a mesma distância. Neste caso, a distância entre dois

pontos consecutivos, definido por ' d_i ', sempre será igual ao valor médio das distâncias, definido por ' $\bar{d}$ '.

2.2.1.2. Hipervolume

A medida de **Hipervolume** (HV) leva em consideração tanto a convergência do conjunto de soluções na fronteira atual como também a diversidade das soluções definidas nesta fronteira (ZITZLER e THIELE, 1999). Este indicador calcula o volume, dentro do espaço de objetivos, dado pela cobertura dos elementos presentes na fronteira de soluções não-dominadas Q . Conforme podemos observar na forma de uma linha descontínua ilustrada na Figura 6 contendo os pontos $Q = \{A, B, C, D, E\}$.

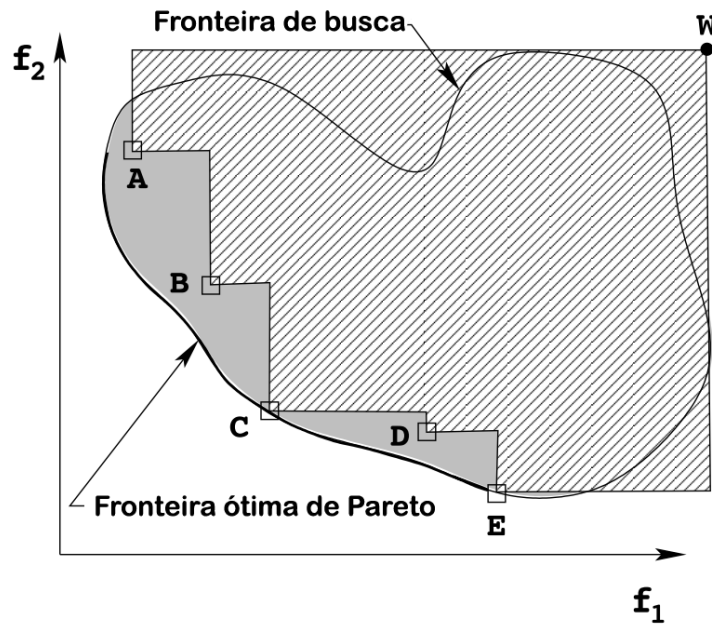


Figura 6 - Hipervolume delimitado por um conjunto de soluções não-dominadas (DEB, 2001).

Matematicamente falando, para cada solução $i \in Q$, um hipercubo v_i é construído com um ponto de referência W e a solução i como as arestas diagonais do hipercubo. O ponto de referência pode ser encontrado ao construir um vetor contendo pontos com os piores valores de avaliação para os objetivos do dado problema. Após este cálculo, a Equação 6 realiza uma união de todos os hipercubos, obtendo o valor final do hipervolume.

$$HV = volume \left(\bigcup_{i=1}^{|Q|} v_i \right) \quad (6)$$

Uma fronteira de Pareto com um valor de hipervolume maior que outro pode significar duas coisas: algumas soluções da fronteira com valor HV maior dominam as soluções da fronteira com HV menor, ou então as soluções de HV maior possuem uma melhor distribuição de pontos ao longo da fronteira. Portanto, é desejável obter valores altos de hipervolume para qualquer problema de otimização multiobjetivo (DURILLO *et al.*, 2009).

Esta métrica também pode ser normalizada a respeito dos valores de objetivo dentro de um intervalo [0, 1]. Neste caso, a formulação final do Hipervolume normalizado é dado na forma encontrada na Equação 7. Este cálculo se trata da divisão do valor de Hipervolume obtido para a solução ‘Q’ dividido pelo valor de Hipervolume da fronteira de referência ‘P*’, podendo assumir o conjunto ótimo de Pareto, caso seja conhecido (DEB, 2001).

$$HVR = \frac{HV(Q)}{HV(P^*)} \quad (7)$$

Para compreendermos melhor o significado desta medida de desempenho, consideremos uma situação em que a curva de Pareto está sendo minimizada em relação a dois objetivos diferentes quaisquer, ilustrado na Figura 7.

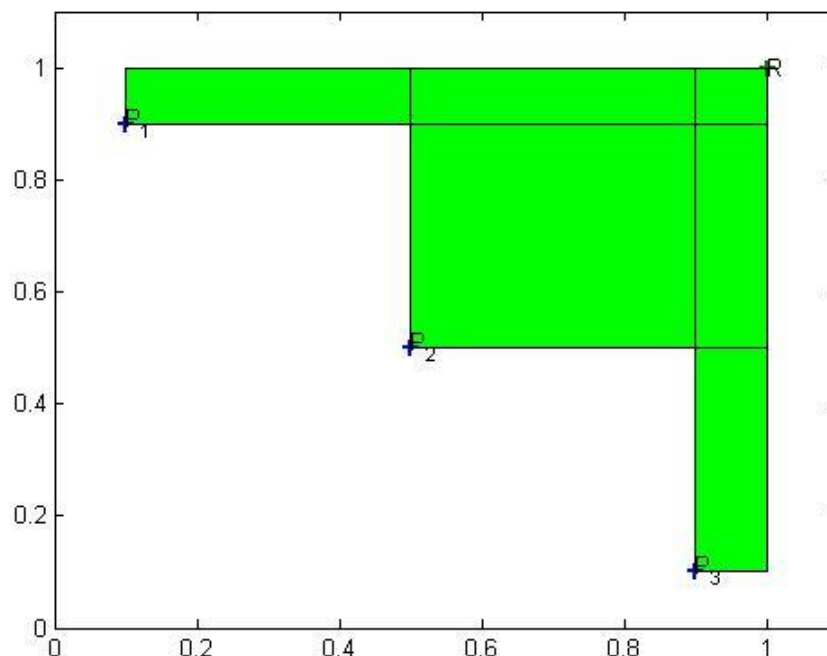


Figura 7 - Cálculo do Hipervolume com valores de função normalizados dentro do intervalo [0, 1] (CAO, 2008).

Obtemos um valor de referência em um ponto ‘R’ do espaço de busca, indicando o pior valor possível em uma situação em que estamos procurando minimizar os valores de função.

Com base nos valores destes dois objetivos para o ponto ‘R’, que formarão coordenadas cartesianas num espaço bidimensional, o espaço de busca será normalizado dentro do intervalo [0, 1]. Em um cenário ideal de minimização de valores de função, os pontos da curva de Pareto encontrada serão exatamente iguais aos pontos do vetor de referência definido pelas melhores soluções conhecidas para o dado problema. O cálculo da área, ao multiplicar largura pela altura, formará uma área total igual a um, correspondente ao valor máximo do Hipervolume. Podemos inferir que, conforme o cálculo do Hipervolume se aproxima do valor máximo ‘1’, isto significa que os pontos da curva estão se aproximando da curva ótima de Pareto. Sabemos também que, pela cálculo da área considerar tanto altura quanto a largura contemplada por estes pontos, esta medida de desempenho também é maximizada quando estes pontos atingem uma melhor distribuição ao longo do espaço de busca.

Note que estes indicadores de qualidade normalmente necessitam do conhecimento a respeito da fronteira de Otimalidade de Pareto. Dada a crescente complexidade computacional dos problemas a serem resolvidos, não é uma tarefa trivial obter a localização desta fronteira com valores ótimos. Deste modo, normalmente é utilizado um valor denominado “fronteira de referência”, composto por todas as soluções não-dominadas obtidas por uma execução anterior de algum algoritmo de otimização multiobjetivo, formando um conjunto de “melhores soluções conhecidas até o momento” (DURILLO *et al.*, 2011).

2.3. Gerenciamento de Riscos

Concluimos a definição de um problema do próximo release multi-objetivo em que estão inseridos uma lista de requisitos candidatos, seus custos de implementação, o grau de interesse dos stakeholders no desenvolvimento destes stakeholders, assim como um possível relacionamento de pré-requisitos dentre estes requisitos candidatos. Agora, para que seja possível a inclusão de um modelo de riscos para o problema do próximo release, necessitamos inicialmente definir um processo de gerenciamento de riscos e como este processo se relaciona com o desenvolvimento de um incremento de software. Para isto, a pesquisa de Boehm (BOEHM, 1991) é uma referência clássica que serve como fundamentação para as práticas e princípios do gerenciamento de risco de software. Apesar deste trabalho possuir quase três décadas desde sua publicação, notamos em pesquisas recentes (KAJKO-MATTSSON, LUNDHOLM e NORRBY, 2009) (HAN, 2014) (NORDIN *et al.*, 2014) (FELDERER, AUER e BERGSMANN, 2016) que os fundamentos e observações realizadas no trabalho de Boehm continuam sendo válidas. Isto se evidencia especialmente no fato que o processo de gerenciamento de riscos, apesar de ser uma etapa reconhecidamente importante para o sucesso

de um projeto de software, ainda é realizada de maneira informal e centralizada no papel do gerente de projeto. O gerente com experiência em tratar com potenciais riscos de projeto acaba por se tornar um fator decisivo no bom andamento na gestão de riscos. O trabalho de Boehm, apesar de apresentar um modelo formal de gerência de riscos conforme veremos a seguir, é baseado em observação nas boas práticas realizadas na indústria por estes gerentes e pelos demais atores envolvidos no processo.

Uma boa prática na gestão de riscos é o mapeamento dos mesmos desde o início do processo e subsequente monitoramento ao longo de todas as etapas deste processo, de modo que seja possível identificar quaisquer mudanças. O risco é normalmente visto como uma incerteza a ser avaliada de maneira qualitativa. No entanto, não é fácil realizar uma atribuição numérica na possibilidade de ocorrência de um evento de risco ou na sua consequência. Deste modo, os riscos são classificados usando termos lingüísticos como *alto*, *médio* e *baixo*, ou também, de maneira típica, em cinco categorias distintas desde os riscos de grau mais crítico até os riscos menos ameaçadores ao andamento esperado do projeto (BANNERMAN, 2008).

Normalmente, a exposição ao risco é dado pelo produto da probabilidade de ocorrência deste risco multiplicado pela perda ou evento negativo esperado pelo evento de risco, também denominado de severidade ou impacto (BOEHM, 1991). Entretanto, conforme podemos observar em (ZHANG, 2007), a probabilidade de ocorrência de risco e sua possível consequência não são dados suficientes para monitoramento de riscos. Portanto, também se deve levar em conta as potenciais vulnerabilidades do projeto, assim como a capacidade da organização em mitigar e reagir a ocorrência deste risco. No modelo proposto pelo autor, a probabilidade e a severidade do risco são tratadas separadamente em vez de fornecer a avaliação final do risco como um produto destes valores.

O uso de técnicas de mitigação de risco podem alterar as consequências esperadas pela ocorrência de um risco, sendo reduzidas ou completamente eliminadas. Esta intermediação entre a probabilidade de ocorrência de risco, a ação de técnicas de mitigação e seu efeito final no impacto do risco, conforme ilustrado na Figura 8 demonstra como a total ausência de técnicas apropriadas para o tratamento de risco impactam na consequência real do evento de risco, definindo então a vulnerabilidade do projeto. Esta vulnerabilidade, conforme ilustrado, pode ser tratada de acordo com as capacidades de equipe, incluindo o uso de técnicas de mitigação do risco associado à esta fonte de vulnerabilidade, diminuindo então a severidade real do risco, que define a consequência de um evento de risco.

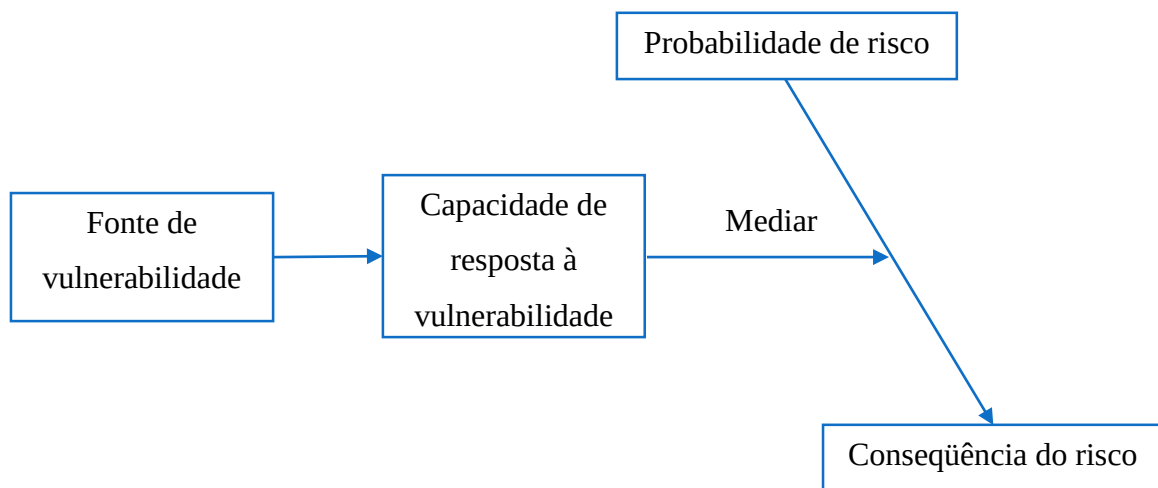


Figura 8 - Modelo de processo de riscos proposto por (ZHANG, 2007).

De acordo com a visão dos interessados no projeto de software, o critério definido como um “evento indesejado” que define um evento de risco pode variar de acordo com o papel do ator neste processo (BOEHM, 1991). Por exemplo, para clientes e desenvolvedores, o atraso e estouro de orçamento são considerados insatisfatórios. Para usuários finais, produtos que entregam uma funcionalidade errada, uma interface ruim, um programa que apresente uma performance ruim ou não confiável são os principais riscos observados. Finalmente, os mantenedores do sistema observam os principais riscos relativos à baixa qualidade do software. Estas distintas visões a respeito do risco podem compor uma *checklist* que pode auxiliar na identificação e avaliação dos riscos apresentados pelo projeto.

A Figura 9 define o esquema de Boehm para gerência de riscos. Conforme podemos observar, este processo se divide em duas grandes categorias, em que cada uma delas contém três sub-divisões.

Na primeira etapa primária, a **avaliação de risco** envolve as etapas de identificação, análise e priorização de riscos. A **identificação de risco** produz uma lista de itens específicos ao escopo do projeto que podem comprometer o sucesso deste projeto. As tarefas típicas desta etapa incluem *checklists* de riscos mais comuns em relação ao domínio do problema e o uso de experiência em projetos similares realizados no passados. A **análise de risco** avalia a probabilidade e a possível perda identificada com cada item de risco e avalia a composição de riscos interdependentes por cada item de risco. Técnicas típicas dessa etapa envolvem o uso de modelos de performance, modelos de custo, modelos de interconexões, análises estatísticas e qualitativas (tais como confiabilidade, disponibilidade e segurança). A última tarefa desta etapa primária se trata da **priorização de riscos**, que possui como saída uma lista ordenada em

ranking dos itens de risco identificados e analisados nas etapas anteriores. As técnicas típicas de priorização de risco envolvem a análise de exposição ao risco, o ganho produzido pela redução destes riscos, usualmente envolvendo algum modelo de custo-benefício ou alguma outra técnica com base no consenso dos envolvidos no projeto.

A segunda etapa primária, denominada de **controle de risco**, envolvem três sub-atividades conforme descritas abaixo. A sub-atividade de **planejamento de gerenciamento de riscos** prepara o gerente em como lidar com cada item de risco através de alguma estratégia de resposta para cada risco. Isto inclui a coordenação de cada risco individual com os demais e em relação ao plano de projeto. As técnicas típicas desta atividade incluem o uso de *checklists* com técnicas de resolução de risco, uma análise custo-benefício e o esboço de um planejamento de gestão dos riscos identificados. A segunda sub-atividade de **resolução de riscos** produz uma situação em que os riscos são eliminados ou resolvidos de alguma forma, conforme ilustrado na Figura 9, esta atividade inclui o uso de protótipos, simulações, *benchmarks*, análise de missão, acordos realizados com pessoas-chave do processo de software, abordagens de design orientado a custo e o uso de desenvolvimento incremental. A etapa final de **monitoramento de riscos** envolve o rastreamento do progresso do projeto em resolver os itens de risco identificados e a tomada de ações corretivas onde forem apropriadas. Nesta etapa, se pode incluir uma lista semanal, mensal ou baseada em *milestones* com os top 10 riscos mais importantes a serem acompanhados para reavaliação ou aplicação de alguma ação corretiva.

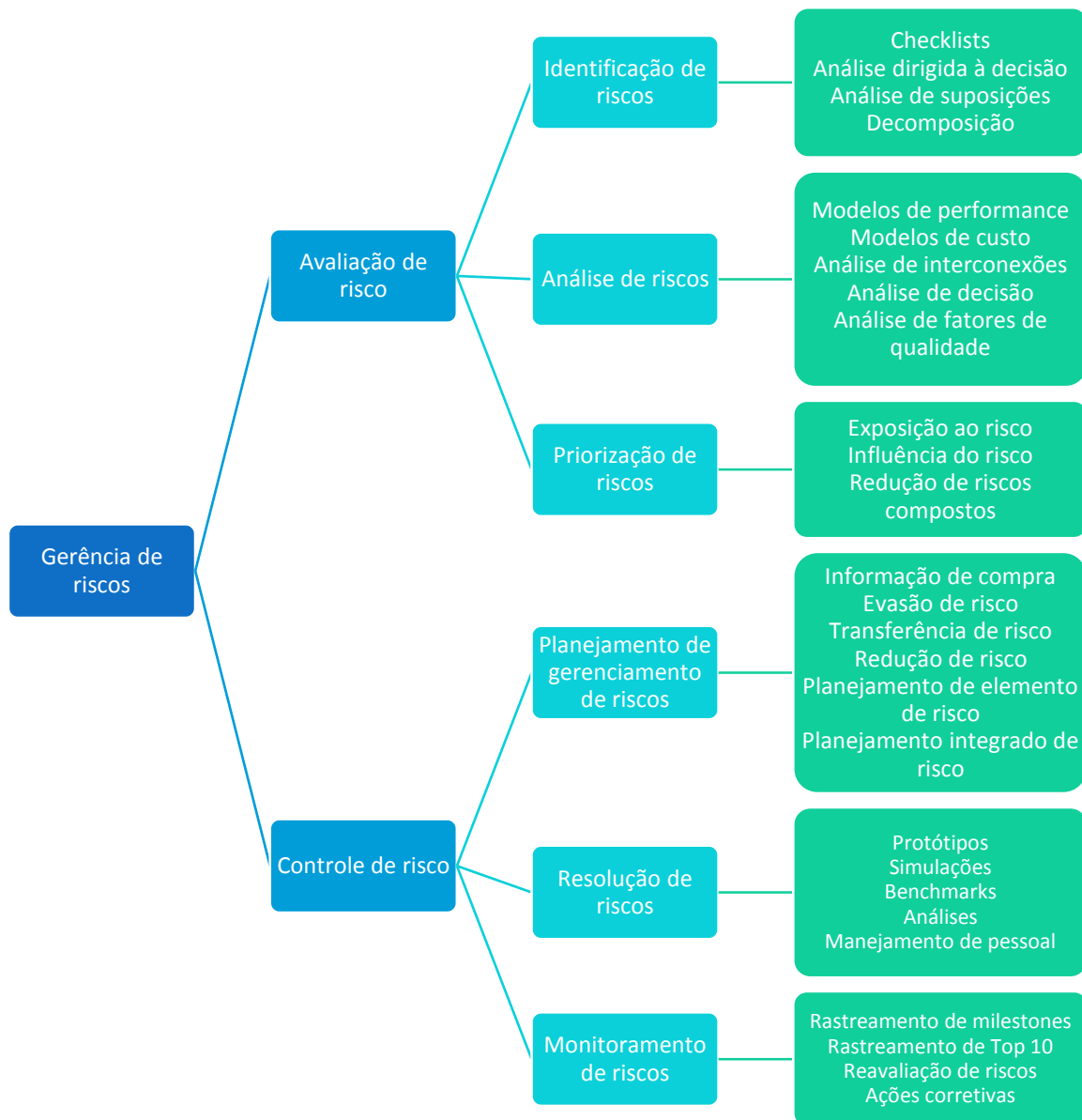


Figura 9 - Etapas do processo de gerenciamento de riscos (BOEHM, 1991).

Em resumo, uma boa definição para gerenciamento de riscos é um conjunto de princípios e práticas com objetivo de identificar, analisar e lidar com fatores de risco de modo a melhorar a possibilidade de que o projeto seja bem-sucedido e/ou evitar o fracasso do projeto (BANNERMAN, 2008). Mais do que um processo ou metodologia, é uma habilidade desenvolvida dentro de uma organização através de sucessivos aprendizados, práticas e outros mecanismos ao longo de um extenso período de tempo. O gerenciamento de riscos não se trata apenas de uma atividade de identificar e avaliar riscos, mas também saber como responder efetivamente e rapidamente a estes riscos, seja através de técnicas de mitigação de riscos ou através de estratégias de contingência.

Dentre os métodos capazes de lidar com risco, é possível destacar quatro categorias (BANNERMAN, 2008), sendo elas:

- A **evasão** tem como meta prevenir um efeito negativo impactando um projeto de software. Isto pode envolver, por exemplo, a mudança de um design do projeto de modo que a circunstância em que o evento de risco nunca possa ocorrer;
- A **transferência** de risco consiste em fazer com que o evento de risco não esteja mais sob responsabilidade da gerência de risco, mas seja transferida para alguém ou para outra equipe que seja mais capaz em lidar com este risco. Note que esta atividade não elimina o evento de risco, muito menos evita que este risco ocorra. Exemplos deste tipo de atividade são o *outsourcing* e contratos de seguro, em que terceiros são responsáveis por resolver o risco ou diminuir seus efeitos;
- A atividade de **mitigação de riscos** representa um conjunto de ações capazes de reduzir a possibilidade de ocorrência de um evento de risco ou seu potencial impacto antes que o risco ocorra. O objetivo principal desta atividade é evitar que o risco ocorra e, caso ocorra, sua consequência deve ser reduzida ao máximo grau possível. Por exemplo, o emprego de testadores e de scripts de teste independentes responsáveis por verificar e validar o software constantemente durante o desenvolvimento, de modo que se reduza a possibilidade de que defeitos sejam encontrados somente após a entrega, minimizando atrasos devidos a problemas com a qualidade do software;
- A **aceitação** do risco envolve uma série de estratégia passivas ou ativas ao risco. A ação mais simples é não tomar nenhuma atitude em relação a ocorrência de risco e apenas monitorá-lo. Isto é uma atitude comum em riscos de baixa consequência e que são considerados toleráveis. A outra ação possível é, mesmo reconhecendo que o risco representa uma ameaça real ao projeto, pouco pode se fazer para que ele seja evitado, portanto, uma ação de aceitação é a criação de um plano de contingência ou um plano de ação para o caso do risco acontecer. A validação e manutenção apropriada desta ação alternativa é necessária para que o plano ocorra conforme previsto e o evento de risco possa ser tratado adequadamente. Portanto, diferentemente da atividade de mitigação de riscos, a execução do plano de contingência só é possível de ser acionada após a ocorrência do evento de risco.

Dentre as técnicas mais importantes para gestão de riscos, podemos citar as *checklists* de identificação de risco, a priorização de riscos, o planejamento de gerenciamento de riscos e o monitoramento de riscos (BOEHM, 1991).

O uso de *checklists* para identificação de riscos podem ser utilizados para identificar as dez principais fontes de risco para o projeto em seu atual momento, podendo ser composta a partir da experiência de gerentes experientes em projetos passados que se assemelham com o projeto a ser entregue. O gerente ou os engenheiros do sistema podem utilizar a lista para auxiliar na identificação e na resolução dos itens de risco que representam a maior ameaça ao sucesso do projeto. Esta lista também provê um conjunto de técnicas de gerenciamento de risco apropriadas que foram mais bem sucedidas em evitar ou mitigar a fonte deste risco. Ao utilizar essa *checklist*, o gerente é capaz de avaliar o status do projeto de acordo com os atributos individuais relacionados com seus requisitos, pessoal, software reusável e ambiente de suporte. Estas avaliações permitem estimar a probabilidade de que um determinado risco afete o sucesso de projeto, atribuindo classificações qualitativas ou numéricas à probabilidade de cada item de risco. A maior parte dos riscos críticos se relacionam com a incapacidade de compreender o escopo do trabalho a ser feito.

Para a etapa de **análise e priorização de riscos**, após a utilização de diferentes *checklists*, ou após o uso de alguma outra técnica de identificação de risco, é provável que a lista total de riscos identificados que estão associados ao projeto pode possuir uma quantidade tão elevada que a equipe tomaria muitos anos apenas os investigando. Deste modo, algum método em que mantenha a equipe focada apenas nos riscos essenciais deve ser empregado para utilizar o tempo disponível de maneira racional nos aspectos mais críticos do projeto. A técnica mais eficaz de priorização de risco consiste na medida de probabilidade e severidade de riscos, combinadas em um indicador de exposição ao risco. Estes valores de probabilidade e impacto do risco permitem facilitar no ranqueamento dos itens de risco, listando então os riscos mais importantes que devem ser atacados. Uma limitação deste método, porém, é dificuldade em atribuir valores numéricos precisos aos graus de probabilidade e severidade do risco. O uso de *checklists* adicionais com valores de referência para a probabilidade e severidade de riscos mais comuns pode ser útil em aproximar estes valores do desejado.

Outras técnicas como a prototipação, *benchmarking* e simulação podem prover medições mais precisas de probabilidade e impacto do risco. Porém, o tempo e custo empregado nestas técnicas podem exceder o ganho obtido no tratamento deste risco, ou então o risco não possui importância suficiente para justificar o custo e tempo empregado nestas técnicas. O uso de

discussões em grupo também podem auxiliar nesta tarefa, sendo uma técnica mais simples em que cada membro da discussão atribui uma nota de 0 a 10 a respeito dos riscos que devem ser priorizados na atual etapa do processo de software. Porém, observamos que esta etapa como um todo possui um alto grau de subjetividade sujeito às avaliações dos especialistas de domínio, assim com a incerteza intrínseca em prever eventos de risco futuros que devem ser evitados, portanto, esta etapa dificilmente produz valores exatos de probabilidade ou severidade de risco.

No **planejamento de gerenciamento de riscos**, após a definição dos principais itens de risco, o gerente ou os envolvidos devem estabelecer um conjunto de funções de controle de risco. Conforme já apontado anteriormente, a definição das técnicas de mitigação de riscos devem ser delineados nesta fase, normalmente com auxílio da experiência ou lista de boas práticas que eliminaram o risco com sucesso em projetos passados. Adicionalmente, o uso de prototipação também pode auxiliar esta atividade. Por exemplo, o uso de um protótipo de certos recursos de segurança no produto pode ser um método com bom custo-benefício em prever como tais aspectos de segurança impactariam a performance do produto final, além de ser uma ferramenta útil em prever como certas decisões a respeito da implementação destes itens de segurança podem reduzir incertezas do projeto. Naturalmente, a criação deste protótipo deve ser encaixado dentro do cronograma geral do projeto, tornando esta estimativa de tempo realista. Ademais, é importante citar que cada elemento de risco crítico ao projeto deve possuir um planejamento próprio a respeito de como lidar com estes riscos.

Partindo para as etapas de **resolução e monitoramento de riscos**, consistem basicamente em implementar quaisquer técnicas levantadas nas duas etapas anteriores de acordo com o planejamento de gestão de riscos definido, especialmente no tocante à aplicação de técnicas de resolução e mitigação de risco. O monitoramento de risco garante que este processo se encadeia como um ciclo fechado, em que os riscos são continuamente avaliados em relação ao progresso do projeto e descobrir se tais riscos estão sendo minimizados ao aplicar tais ações corretivas para manter o projeto em bom andamento. Este monitoramento deve ser aplicado de maneira regular pelo chefe do gerente de projeto. O autor define como uma boa prática, em projetos de larga-escala com mais de 20 participantes, que tais revisões devem ser aplicadas mensalmente. Dentro do projeto em si, o gerente deve revisar os riscos mais críticos de maneira mais freqüente.

A revisão do projeto, de acordo com estes riscos, deve ser definido com o status atual de cada risco monitorado, como ele está posicionado no *ranking* de riscos mais críticos ao sucesso do projeto, seu *ranking* na revisão passada, o quão freqüente este item de risco permaneceu

entre os dez riscos mais críticos e um sumário do progresso na resolução deste risco ao longo do projeto. Com base na discussão desta reunião, analisando o progresso e o grau de sucesso na aplicação de ações de resolução deste risco, ao final da revisão este risco pode mudar seu posicionamento no *ranking*. Esta posição pode subir para as primeiras posições caso o risco não esteja progredindo adequadamente de acordo com o planejamento. Na direção oposta, caso o risco esteja sendo resolvido de acordo com o plano ou de maneira mais ágil que inicialmente previsto, tal risco pode reduzir sua posição no *ranking* e ser reavaliado de acordo com sua probabilidade de ocorrência ou a consequência negativa que tal evento de risco influenciaria no sucesso do projeto.

O autor denota que, para a implementação bem sucedida de um processo de gerenciamento de riscos, o processo de software e a cultura da empresa devem ser modificados gradualmente para se adequarem às necessidades das atividades deste processo de gestão de riscos (BOEHM, 1991). Entretanto, nestas últimas décadas, observamos que pouco se alterou e que o processo de gestão de riscos continua sendo um processo informal e centralizado no papel do gerente de projeto (KAJKO-MATTSSON, LUNDHOLM e NORRBY, 2009) (HAN, 2014) (NORDIN *et al.*, 2014) (FELDERER, AUER e BERGSMANN, 2016). Como amostra, a partir de uma quantidade considerável de projetos de software, em que 139 projetos de software foram avaliados é que o processo de gerenciamento de riscos não é formalmente empregado dentro do processo de desenvolvimento de software ou é realizado de maneira informal (HAN, 2014). Segundo (FELDERER, AUER e BERGSMANN, 2016), uma pesquisa realizada em 57 empresas de software na Alemanha, Áustria e Suíça, 33 delas não empregam uma metodologia de gestão de riscos. Dentre os principais motivos, se encontram a indisponibilidade recursos para custear tal processo, ou que tal processo não é enxergado como necessário para o desenvolvimento do software. Finalmente, observamos em (KAJKO-MATTSSON, LUNDHOLM e NORRBY, 2009) que, apesar da tarefa de gestão de riscos ser um processo reconhecidamente importante pelas empresas entrevistadas, a incapacidade de medir precisamente o custo-benefício do uso de um processo formal de gerenciamento de riscos inviabiliza a implementação deste processo, ao notarmos que ele é difícil de ser quantificado, dado a natureza subjetiva do processo de avaliação e tratamento de riscos. Em (NORDIN *et al.*, 2014), a pesquisa realizada com especialistas de domínio e membros da indústria local na Malásia, aponta uma possível solução para que mais empresas adotem um processo formal de gestão de risco: a criação de ferramentas de gerência de risco capazes de demonstrar como o tratamento de risco influencia no sucesso do projeto, além de apontar o gasto relativo ao

emprego das atividades de gestão de riscos. Isto forneceria algum indicador de custo-benefício no uso deste processo, podendo convencer a organização a respeito do retorno obtido pela implementação desta atividade aos seus projetos.

Finalmente, Boehm (BOEHM, 1991) conclui que a gerência de riscos não é uma abordagem baseada em uma receita pronta. Para lidar com todos os aspectos complexos relacionados à tecnologia da informação e a gestão de pessoas, o julgamento humano com base na experiência é o aspecto de sucesso principal nesta atividade, assim como tal experiência e habilidade de pessoal é o que faz projetos serem bem encaminhados.

2.4. Considerações Finais

Neste capítulo observamos a formulação clássica para um problema do próximo release, inicialmente definido com um conjunto de requisitos candidatos a serem entregues na próxima versão do produto de software. Estes requisitos candidatos contam com um custo associado e o projeto contém um conjunto de clientes interessados na continuidade do desenvolvimento do sistema representado por interesses distintos nos requisitos candidatos. Deste modo, o problema consiste em selecionar o melhor subconjunto de requisitos que sejam de maior interesse para os clientes e que não estoure o orçamento pré-definido para a versão do sistema. Posteriormente, o conceito de um orçamento pré-fixado foi abolido e o problema do próximo release multi-objetivo foi formalizado em razão desta mudança. Nesta nova formulação, várias boas soluções são entregues como solução do problema, em que o tomador de decisão deve selecionar a solução que melhor se adeque às características do projeto. Para diferenciar a qualidade de cada solução individual, o conceito de dominância de soluções é definido. Deste modo, todas as soluções não-dominadas dentre todo o conjunto de soluções possíveis formam o conjunto ótimo de Pareto. Também observamos que, para um problema com dois objetivos a serem otimizados, neste caso particular uma maximização da satisfação e uma minimização de custos, o gráfico bidimensional de conjunto de Pareto forma uma curva denominada Fronteira de Pareto. Finalmente, a segunda metade deste capítulo forneceu a fundamentação de um processo de gerência de riscos obtida a partir da observação de boas práticas na indústria. As três principais atividades do processo foram definidas como sendo a avaliação e controle de riscos. Sendo tais atividades sub-divididas em identificação, análise e priorização de risco para a primeira atividade, enquanto que a atividade posterior se sub-divide em planejamento de gestão de riscos, resolução e monitoramento de riscos. Observamos a subjetividade e a experiência influenciam na identificação, medição e na definição de ações de resposta aos itens de risco detectados no projeto. Também observamos a importância de que tais riscos sejam

continuamente revisados durante o processo para avaliar a eficácia das técnicas de mitigação de risco, assim como reavaliar a probabilidade e impacto deste risco no projeto. Concluímos esta etapa observando que, apesar da importância de um processo de gestão de riscos, muitas empresas, por diferentes razões, ainda não empregaram tal processo em seus projetos, consistindo em uma atividade informal e centralizada no papel do gerente de projeto.

3. METAHEURÍSTICAS

Este capítulo introduz os conceitos envolvidos no uso de metaheurísticas de forma a possibilitar a busca por uma solução que, apesar de não garantir que a solução ótima seja encontrada, admite uma solução de boa qualidade, caso a metaheurística possua uma configuração adequada e o problema seja bem modelado. É importante ressaltar que os conceitos aqui abordados são baseados no livro “*Introduction to Genetic Algorithms*” (SIVANANDAM e DEEPA, 2008), sendo as informações complementares baseadas em fontes diversas.

Ao enfrentarmos um problema de otimização de valores relativamente simples, podemos admitir o uso de um algoritmo simples capaz de testar todas as combinações possíveis de valores de modo a encontrarmos o melhor resultado, após exaurir todas as possibilidades de soluções. Ao realizarmos este método de **busca exaustiva**, possuímos a garantia de encontrarmos a melhor solução, ou, em outras palavras, a **solução ótima** para um problema qualquer. Considere que este problema simples realize a seleção ou exclusão de 10 requisitos, codificados de forma correspondente em 10 bits, assumindo valores ‘1’ e ‘0’ para sinalizar esta seleção ou exclusão do requisito avaliado. Portanto, o problema possui um total de 1024 (2^{10}) possibilidades. Caso este problema acomode um conjunto de requisitos maior, por exemplo, um problema que admita cinquenta requisitos distintos, o problema cresce em complexidade para acomodar 1.125.899.906.842.624 (2^{50}) possibilidades distintas. Deste modo, o problema se torna intratável de ser resolvido por um algoritmo simples que realize o cálculo de todas as possibilidades possíveis dentro deste **espaço de busca**, visto que o computador responsável por executar esta busca não forneceria a solução ótima em tempo hábil, independentemente do aumento de recursos computacionais. Este crescimento de **complexidade computacional**, no caso deste problema específico, um problema de complexidade NP-Completo, ocasiona que o problema se torna intratável, pois a quantidade de soluções existentes no espaço de busca cresce de maneira exponencial em relação ao aumento da entrada do modelo avaliado.

Deste modo, se justifica o uso de alguma técnica de busca direcionada que visa melhorar a qualidade da solução, mesmo que todos os elementos deste espaço de busca não sejam explorados pelo algoritmo. Algumas técnicas tradicionais de busca heurística incluem a busca gulosa, busca A*, porém, elas não são adequadas nos casos em que não é possível modelar uma função de custo comparando o estado atual da busca a um estado final desejado (RUSSELL e NORVIG, 2009). Em casos em que o estado atual da solução é a informação mais relevante e

este dado é derivado por uma função objetivo, como a satisfação obtida pela seleção de um conjunto de requisitos, por exemplo, outras técnicas mais adequadas se aplicam, com destaque para os Algoritmos Genéticos.

3.1. Algoritmos Genéticos

Os Algoritmos Genéticos, conforme veremos na seção seguinte, é uma metaheurística baseada nas observações extraídas das evoluções das espécies. Esta metaheurística trabalha com um conjunto de soluções que forma uma população, sendo que esta população, ao final da execução do algoritmo, deve retornar um resultado que melhor se aproxime da solução ótima global. Esta execução consiste em gerar uma população inicial aleatória, recombinar pares de indivíduos da população através de alguma técnica de cruzamento, o que irá gerar indivíduos novos. Estes indivíduos novos, por sua vez, podem ser afetados por algum operador de mutação, que modifica uma pequena parcela de características deste indivíduo, podendo emergir uma nova característica desejável para o problema. Finalmente, uma iteração de um algoritmo genético é encerrado em determinar quais indivíduos, sejam eles novos ou antigos, devem permanecer na população para serem avaliados na próxima iteração. Após isso, algum critério de parada é utilizado para definir se uma nova iteração deve ser iniciada, ou se o algoritmo deve encerrar sua execução e retornar a população encontrada na última iteração. Este ciclo é ilustrado na Figura 10.

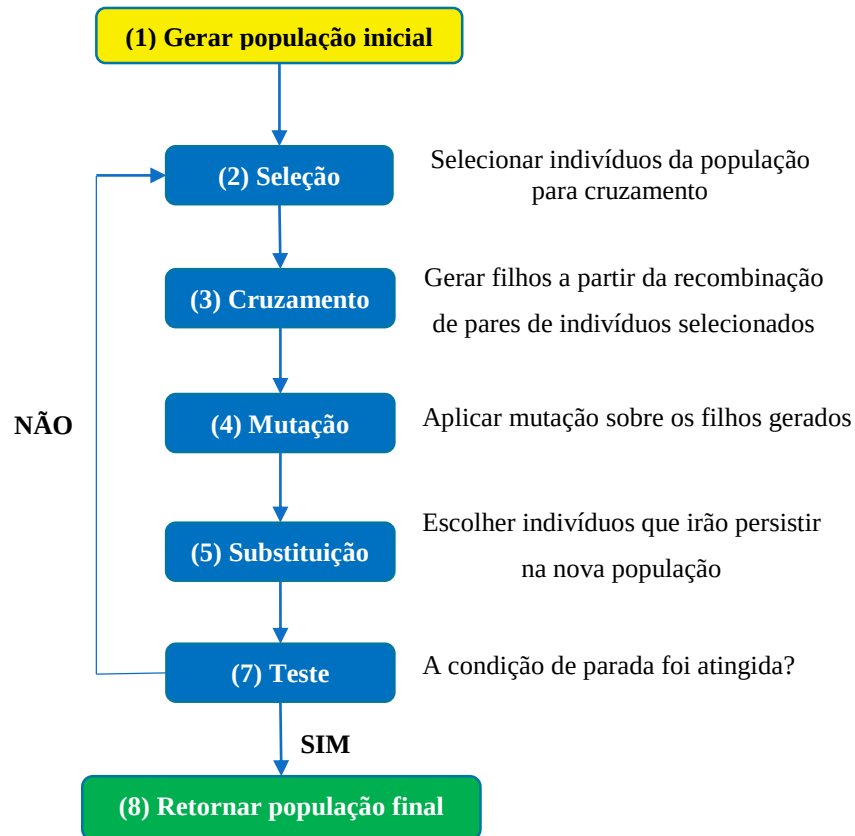


Figura 10 - Ciclo de execução de um algoritmo genético.

Uma característica interessante dos algoritmos genéticos é sua capacidade de incorporar um certo grau de aleatoriedade na sua execução, seja na geração de sua população inicial, no não-determinismo de seleção de indivíduos para cruzamento, além do não-determinismo a respeito de qual parcela do indivíduo o operador de mutação irá atuar. Além disso, os operadores de cruzamento e mutação incorporam características de busca global e local, respectivamente. Esta classe de algoritmos é adequada nas situações em que técnicas mais especializadas não forneceram resultados de boa qualidade, ou quando não existe muito conhecimento disponível sobre o espaço de busca em razão destas características especiais dos algoritmos genéticos.

3.1.1. Background da Biologia

Os algoritmos genéticos estão inseridos na classe dos algoritmos bio-inspirados, no âmbito dos algoritmos evolucionários inspirados na Teoria da Evolução das Espécies de Charles Darwin. Nesta teoria, a seleção natural consiste na sobrevivência dos indivíduos que possuem características mais favoráveis de se adaptarem ao ambiente em que eles estão inseridos. Adicionalmente, existe uma competição por recursos escassos como alimento, abrigo, assim como uma disputa de indivíduos da mesma espécie para atrair pares para reprodução. Neste

contexto, podemos observar que os indivíduos da população que tenham um desempenho ruim neste ambiente possuem menor propensão a sobreviverem e gerar descendentes que herdarão suas características, sendo deste modo descrito como um indivíduo que possui um **fitness** baixo.

O conceito de **Computação Evolucionária** foi inicialmente concebido na década de 1960 por I. Rechenberg no trabalho “Estratégias Evolucionárias” (RECHENBERG, 1989). Este trabalho possibilitou que, mais tarde, em 1975, fosse desenvolvido um livro denominado “Adaptação em sistemas naturais e artificiais” por John Holland, concebendo assim os primeiros **Algoritmos Genéticos** (*Genetic Algorithms, GAs*) como um método capaz de resolver problemas de busca e otimização (HOLLAND, 1992).

A informação genética de um indivíduo é armazenado em **cromossomos**, normalmente construída em uma molécula de DNA e constituída por pares. Os cromossomos, por sua vez, são divididos em **genes**, em que cada gene pode codificar uma propriedade da espécie do indivíduo, como, por exemplo, a cor dos olhos. Cada gene pode assumir diferentes configurações formadas por **alelos**, seguindo o exemplo da cor dos olhos, podem existir alelos para representar a cor verde, azul ou castanho. Todos os alelos possíveis de uma população formam o *pool* genético, determinando assim todas as variações possíveis para as futuras gerações. Portanto, conforme maior a variedade de alelos presentes na população, maior será a diversidade da população.

No contexto biológico, existe uma separação entre cromossomos e **genoma**, no qual o genoma representa a totalidade de genes possíveis para uma espécie. Cada gene possui um **locus**, indicando sua posição no genoma. Normalmente, um organismo vivo armazena seu genoma em diferentes cromossomos, mas, no contexto computacional dos GAs, estes genes são armazenados em um único cromossomo. Portanto, no contexto dos Algoritmos Genéticos, genoma e cromossomo são conceitos semelhantes.

Seguindo a fundamentação biológica, podemos observar o **genótipo** como sendo a combinação total dos genes de um indivíduo, enquanto que o **fenótipo** representa algum aspecto físico que se manifesta a partir da decodificação deste genótipo. Quando as espécies procuram um par para realizar a reprodução, a seleção normalmente se dá pela identificação de características desejáveis no fenótipo, mas um novo indivíduo herda a recombinação do genótipo de seus pais. Conforme discutido, os cromossomos normalmente são codificados em pares, constituindo genes diplóides, no qual o gene dominante em cada par irá decodificar uma característica fenotípica. Entretanto, ainda existe a possibilidade do gene recessivo ser repassado para as gerações futuras. Esta codificação diplóide significa uma maior diversidade

nos alelos e serve como um mecanismo de memória, representando uma vantagem para a espécie por resguardar uma maior diversidade genética. Porém, no contexto dos Algoritmos Genéticos, os genes geralmente são codificados através de genes haplóides por uma maior simplicidade de implementação, além de ser suficiente para resolver o problema em questão.

A reprodução de um indivíduo pode ocorrer por **mitose** ou **meiose**. O primeiro método gera uma cópia do indivíduo e é normalmente identificada na reprodução de células de indivíduos. O segundo método, por sua vez, representa a reprodução sexual, no qual dois gametas são combinados a fim de gerar um novo indivíduo distinto.

A origem das espécies é baseada no princípio de “preservação de variações favoráveis e na rejeição de variações não favoráveis”, denotando que o indivíduo que apresentar as melhores características possui uma maior probabilidade de sobrevivência no ambiente.

A Tabela 1 estabelece uma comparação entre a fundamentação biológica dos GAs e como estes conceitos foram mapeados na prática.

Tabela 1- Mapeamento dos fundamentos da biologia para os algoritmos genéticos.

Evolução natural	Algoritmos Genéticos
Cromossomo	String
Gene	<i>Feature</i> / característica
Alelo	Valor da característica
Locus	Posição na String
Genótipo	Estrutura ou string codificado
Fenótipo	Conjunto de parâmetros ou estrutura decodificada

3.1.2. Definição de um algoritmo genético clássico

Os algoritmos genéticos partem de um princípio estocástico, no qual a seleção, reprodução e mutação de indivíduos devem possuir algum grau de aleatoriedade, concedendo, é claro, que os indivíduos com características com um maior *fitness* possuam uma maior

probabilidade de emergirem nestas etapas. Além disso, em vez de lidarmos com uma única solução, os GAs tratam com um conjunto de indivíduos em uma população, permitindo assim uma recombinação destas soluções a partir de técnicas de cruzamento entre pares de indivíduos. A presença de um conjunto de soluções também permite que os GAs sejam construídos para execução em diferentes threads, admitindo paralelismo.

A idéia básica é que o *pool* genético de uma determinada população contém a solução para um dado problema, porém, é pouco provável que a população inicial obtida aleatoriamente durante a etapa de inicialização do algoritmo contenha a solução ótima para o problema. Esta solução portanto não é “ativa”, dependendo de uma série de recombinações de indivíduos, além da realização de diversas mutações que possam emergir características inexistentes na população. Todavia, a mutação deve ser um evento raro, caso contrário, o algoritmo passaria a assumir um comportamento aleatório. Esta variação deve ser realizada em baixa-escala, pois, conforme observado na natureza, as espécies não surgem repentinamente, além disso, é preferível que esta variação seja não-direcionada.

Um algoritmo genético clássico adota um conjunto de princípios:

- O espaço de busca deve ser definido de maneira tal que o algoritmo possa potencialmente visitar todas as soluções possíveis dentro desta região. Portanto, a codificação do cromossomo deve refletir esta **completude**. Adicionalmente, a codificação deve ser única, ou seja, um mesmo cromossomo não pode ser codificado de diferentes maneiras para **evitar redundâncias**;
- O algoritmo deve incluir operadores de **cruzamento** (*crossover*) entre cromossomos e um operador de **mutação**. Assim sendo, os filhos obtidos no cruzamento devem herdar boas características dos pais;
- O operador de seleção de indivíduos deve ser derivado a partir dos valores obtidos a partir do *fitness* destes;
- A população inicial deve ser gerada aleatoriamente para garantir a diversidade de características.

A iteração básica de um algoritmo genético possui as seguintes etapas:

- a. Seleção de indivíduos para cruzamento: baseado em uma função de probabilidade relativa ao *fitness* dos indivíduos;

- b. Reprodução de indivíduos: novos indivíduos devem ser gerados a partir da recombinação dos indivíduos selecionados na etapa anterior. Também se deve admitir uma etapa de mutação, que potencialmente poderá emergir novas características nestes indivíduos novos;
- c. Avaliação: o *fitness* destes novos cromossomos devem ser calculados;
- d. Substituição: indivíduos da população antiga podem ser substituídos pela nova geração. Dependendo da técnica de substituição utilizada, indivíduos antigos podem persistir na nova população de acordo com seu valor de *fitness*.

Finalmente, com estas etapas em mãos, podemos esboçar um pseudocódigo de um algoritmo genético, ilustrado no Algoritmo 1.

Algoritmo 1- Pseudocódigo de um algoritmo genético.

01	[Início] Gerar população inicial aleatória
02	[Fitness] Avaliar o fitness de cada indivíduo
03	[Nova população] Repetir os procedimentos abaixo até gerar nova população
04	[seleção] selecionar dois pais a partir de seu fitness
05	[crossover] Gerar filhos a partir de uma probabilidade de cruzamento, se o cruzamento não ocorrer, os filhos serão uma cópia exata dos pais
06	[mutação] Dada uma probabilidade, gerar uma mutação no filho em cada posição (locus)
07	[aceitação] Inserir filho na nova população
08	[Substituição] Usar população na próxima iteração
09	[Teste] Se condição de parada for satisfeita, encerre a execução do algoritmo e retorne a melhor solução obtida na população
10	[Loop] Retorne ao passo 02, caso a condição de parada não seja satisfeita

A estratégia de busca do algoritmo relativo ao *fitness* dos cromossomos podem alterar o comportamento do algoritmo. Por exemplo, um algoritmo puramente determinista, que sempre buscará a melhor solução possível, pode ocasionar uma grande variância na solução, mas, em contrapartida, pode estar preso em uma região de busca para o pior caso. Uma estratégia oposta, de completo não-determinismo pode facilitar que o algoritmo escape deste problema, admitindo uma estratégia estocástica, mas ainda assim é recomendável que a busca seja direcionada com um grau razoável de determinismo, garantindo que a solução possa convergir em um quantidade razoável de iterações, agilizando a execução do algoritmo.

3.1.3. Comparativo com diferentes metaheurísticas

Outros algoritmos não-evolucionários adotam estratégias de busca local, tais como o algoritmo de **Subida de Colina** (*Hill Climbing*) ou o algoritmo **guloso** (*Greedy*), no qual eles observam os pontos vizinhos da região de busca e tentam avançar em direção do nó com melhor avaliação de *fitness*. Entretanto, este tipo de algoritmo, em um problema de maximização de valores, pode cair no problema dos valores máximos locais, restringindo o algoritmo em um espaço sem que ele possa melhorar a solução em direção ao valor ótimo global, conforme ilustrado na Figura 11. Para resolver este problema, é exigido que o algoritmo de subida de colina seja reinicializado diversas vezes para que ele possa iniciar uma nova busca em uma região diferente.

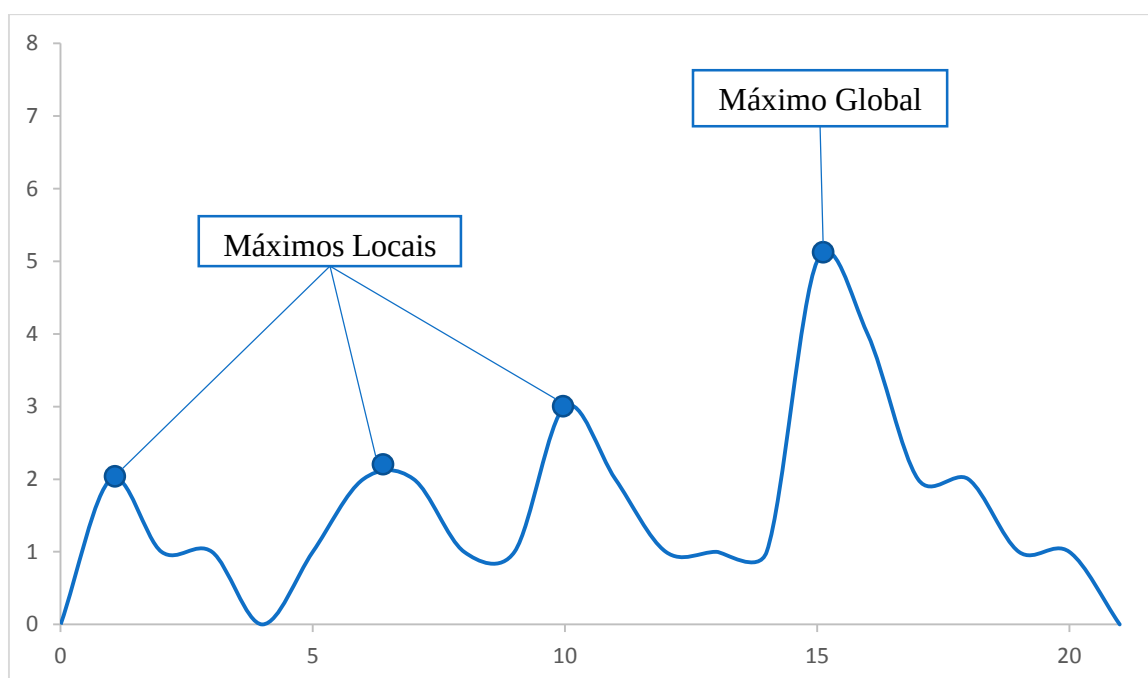


Figura 11 - Exemplos de máximos locais e máximo global.

A metaheurística de Arrefecimento Simulado (***Simulated Annealing***) estabelece uma analogia com as técnicas da metalurgia, em que o metal é tratado inicialmente em uma temperatura elevada, que é decaída lentamente. Computacionalmente falando, o algoritmo inicial com uma “temperatura” elevada admite que o algoritmo pode direcionar sua busca para um nó vizinho que não melhore o valor *fitness* encontrado até o momento. Com o decaimento da “temperatura”, a probabilidade de um nó vizinho que piore a função de avaliação ser selecionado na próxima iteração diminui, aproximando seu comportamento ao de um algoritmo clássico de subida de colina. Esta seleção aleatória no início é adequada para possibilitar que a metaheurística escape de pontos ótimos locais, sendo que um resfriamento infinitamente lento

garante que a solução ótima global seja encontrada em alguma iteração do problema. A principal dificuldade na configuração deste algoritmo, portanto, consiste em definir uma boa taxa de decaimento da temperatura, garantindo simultaneamente uma boa solução e que seja encontrada em um tempo hábil.

O *Simulated Annealing* é um bom competidor aos Algoritmos Genéticos, apresentando por diversas vezes resultados de qualidade semelhante e que podem ser utilizados para comparação. Entretanto, os GAs ainda possuem a vantagem de explorar uma região maior de busca, além de serem mais adaptáveis à implementações com paralelismo de execução. Além disso, podemos observar a ausência de um operador de recombinação de solução, presente nos GAs. Uma metaheurística denominada *Genetic Annealing* combina ambas as abordagens (YAO, 1991). A idéia desta combinação consiste em, após gerar uma população inicial aleatória de m indivíduos, a busca é iniciada utilizando *Simulated Annealing* por m iterações, ao fim desta etapa, o algoritmo verifica se a qualidade das soluções encontradas satisfaz uma condição pré-configurada, caso não seja satisfeita, o algoritmo retorna ao procedimento de *Simulated Annealing*. Caso existam soluções com um alto valor de *fitness*, os operadores genéticos são acionados nestas soluções, com uma probabilidade de que o operador de mutação seja acionado inicialmente alta, sendo progressivamente reduzido de acordo com a quantidade de iterações. Deste modo, após o acionamento dos operadores genéticos e retornando ao procedimento de *Simulated Annealing*, o algoritmo encontrará uma população potencialmente mais favorável em se aproximar da solução ótima, reduzindo o tempo de convergência.

Além das técnicas de busca local e global, existe a possibilidade de admitirmos uma estratégia de **busca aleatória**, em que qualquer nó do espaço de soluções pode ser escolhido ao acaso com iguais possibilidades. Inicialmente, este tipo de busca pode ser rejeitado de imediato por não possuir uma estratégia definida em direção de uma solução, o que pode implicar em uma convergência demasiadamente lenta, mas este algoritmo pode ser útil para o que se define como um “teste de sanidade” (HARMAN *et al.*, 2012). Este teste é muito utilizado durante o processo de validação de um algoritmo e consiste em demonstrar que a estratégia de busca deste algoritmo é capaz de fornecer um resultado suficientemente superior em comparação com uma busca aleatória.

Finalmente, em relação às metaheurísticas aqui descritas, podemos elencar uma série de vantagens dos algoritmos genéticos:

- A existência de descontinuidade na superfície de busca causa pouco impacto na performance geral do algoritmo, diferentemente das outras técnicas citadas;
- Os GAs são resistentes ao problema dos ótimos locais por trabalhar com uma população em vez de observar um único ponto em relação aos seus pontos vizinhos;
- Apresenta uma boa performance para problemas de otimização em larga escala, admitindo implementações paralelas. Também não possui grandes dificuldades para lidar com funções ruidosas, espaços de busca densos ou difíceis de compreender, o que causaria problemas em uma abordagem mais específica ao domínio do problema;
- Os GAs são indicados para problemas definidos como funções multimodais;
- Não está restrito a um único domínio de problemas, diferentemente de técnicas que empregam IA simbólica ou técnicas de otimização exatas. Portanto, são facilmente adaptáveis, além de admitir modificações para lidar com problemas com múltiplos objetivos a serem otimizados.

Entretanto, é imprudente considerar os GAs como uma panacéia capaz de resolver qualquer problema de otimização. Podemos também observar uma série de limitações nesta abordagem:

- Pode ocorrer uma convergência prematura da solução em que o *pool* genético ainda está muito distante da solução ideal;
- Inadequado caso a função *fitness* não seja facilmente identificável;
- Pode existir dificuldade em modelar a representação do problema;
- Não possui um critério de parada efetivo, diferentemente do *Simulated Annealing* que encerra após o resfriamento e identificar que a função não possa mais ser melhorada em relação à vizinhança;
- Requer um grande número de avaliações da função *fitness*;
- A configuração do algoritmo não é trivial, pois diversos parâmetros tais como o tamanho da população, a taxa de mutação, a taxa de cruzamento, assim como a escolha da técnica de mutação, cruzamento e seleção de pais devem ser testados para encontrar os melhores resultados;
- Não é adequado para problemas em que o espaço de busca é definido como uma função unimodal;

- O algoritmo não garante que a solução ótima seja encontrada. Para aumentar a chance de encontrar ótimos globais, é necessário que os GAs sejam utilizados em combinação com alguma técnica de busca local;
- Técnicas de otimização específicas ao domínio do problema que fazem uso de informações auxiliares podem ser mais rápidas e mais precisas.

3.1.4. Operadores

Conforme discutido, existem quatro operadores essenciais na formulação de um algoritmo genético, são eles: seleção, cruzamento, mutação e substituição. Com o passar do tempo, diferentes técnicas surgiram para implementar estes operadores. As seções a seguir irão dar enfoque às técnicas mais conhecidas. É importante notar que, dada a natureza deste trabalho, em que o cromossomo será codificado para refletir a seleção de um conjunto de requisitos, os operadores deverão ser responsáveis por realizar operações sobre dados binários, sinalizando a seleção ou não destes requisitos conforme demonstrado na Figura 12.

0	1	1	0	1	0	1	0
---	---	---	---	---	---	---	---

Figura 12 - Codificação de uma string de 8 bits.

Portanto, técnicas que se baseiam em problemas codificados em números reais ou em permutação de valores serão omitidos por um questão de simplificação de modo a evitar a fuga do tema central deste trabalho.

3.1.4.1. Operador de seleção

O operador de seleção realiza a tarefa de escolher os indivíduos que participarão do cruzamento e irão gerar novos indivíduos que poderão ser incluídos na população. A seleção normalmente se dá com base no valor de *fitness* dos cromossomos, conforme apresentado nas técnicas descritas a seguir.

A **seleção aleatória** é a técnica mais simples de todas, os indivíduos são selecionados aleatoriamente, com igual possibilidade de qualquer indivíduo ser selecionado. Esta técnica possui a desvantagem natural de produzir grande ruído na população e tornar a convergência da solução mais lenta.

Na **técnica da roleta**, a probabilidade de seleção é proporcional ao *fitness* de cada indivíduo. Inicialmente, o algoritmo computa a soma ‘S’ do *fitness* de todos os ‘N’ cromossomos da população. Em seguida, os indivíduos são alocados em uma porção do intervalo (0,S) da roleta, proporcional aos seus respectivos valores de fitness, conforme pode

ser observado na Figura 13. Para a seleção de um dado indivíduo, um número aleatório ' r ' é gerado dentro deste intervalo. Após esta etapa, o cromossomo que ocupa a porção da roleta referente ao valor ' r ' é escolhido para posterior cruzamento.

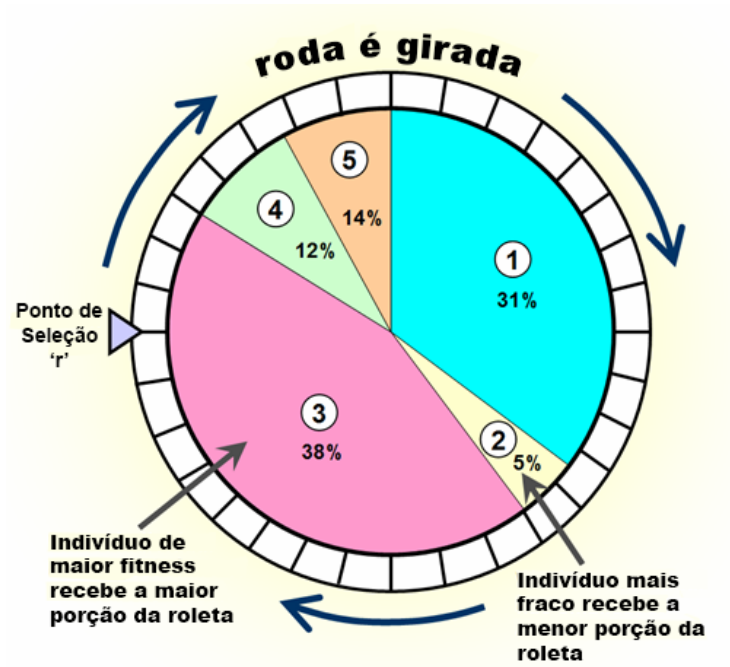


Figura 13 - Técnica de seleção da roleta (EDC, 2015).

Esta técnica não garante que o indivíduo de maior *fitness* seja sempre selecionado e é fácil de implementar. A técnica da roleta apresenta dificuldade quando existe uma grande variância no *fitness* da população, como é o caso ilustrado na Figura 11, onde um indivíduo possui alto valor de *fitness*, enquanto que os demais cromossomos possuem um valor de avaliação muito baixo. Dada esta variância, um conjunto reduzido de indivíduos com *fitness* alto ocuparão a maior parte da roleta, enquanto que os demais cromossomos ocuparão uma parcela reduzida da roleta, diminuindo suas chances de seleção, tornando o processo de seleção elitista, o que potencialmente pode fazer com que a população permaneça restrita a um espaço de busca reduzido, com possibilidade remota de que a solução seja perturbada e possibilite encontrar uma solução melhor do que as soluções já exploradas.

Esta técnica de **seleção por amostragem estocástica** é semelhante à seleção por roleta, em que os cromossomos recebem espaços relativo ao *fitness*, só que, em vez de selecionar um valor aleatório ' r ' dentro do intervalo $(0, S)$, são definidos ponteiros fixados de cruzamento. A quantidade de ponteiros é relativo à quantidade de cromossomos necessários para cruzamento.

Por exemplo, considere um problema em que são necessários 6 cromossomos para realizar cruzamento em uma população com 10 indivíduos. Sabendo que $1/6 = 0.167$, a primeira

seleção será um valor aleatório entre $[0, 0.167]$, o intervalo seguinte obterá um valor entre $[0.168, 0.334]$ e assim sucessivamente. Esta técnica é considerada mais justa que a roleta e está ilustrada na Figura 14. Note que o ponteiro 1 na Figura 14 representa justamente o primeiro intervalo compreendido entre $[0, 0.167]$, deste modo, um número aleatório dentro deste intervalo é gerado, como não existe nenhum outro indivíduo além do indivíduo 1 neste espaço dado seu alto valor de *fitness*, ele será selecionado. Em contrapartida, o espaço compreendido entre o ponteiro 6 e a posição final de seleção, definido entre o intervalo $[0.834, 1]$ permite que um dos três indivíduos com um baixo valor de *fitness* possam ser selecionados, sendo eles os indivíduos 8, 9 ou 10.

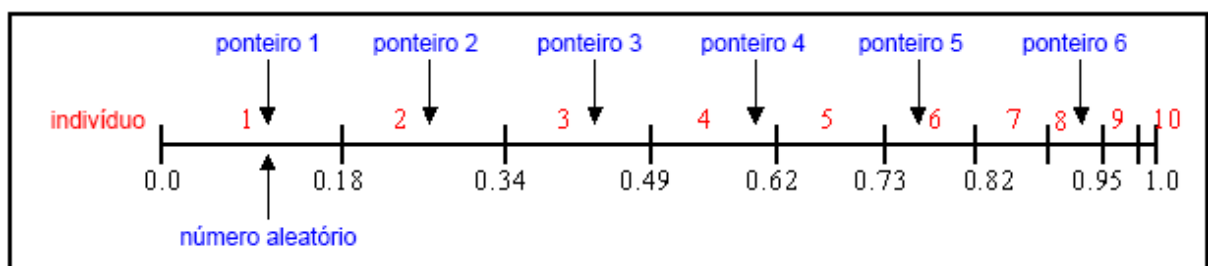


Figura 14 - Exemplo de seleção por amostragem estocástica (SIVANANDAM e DEEPA, 2008).

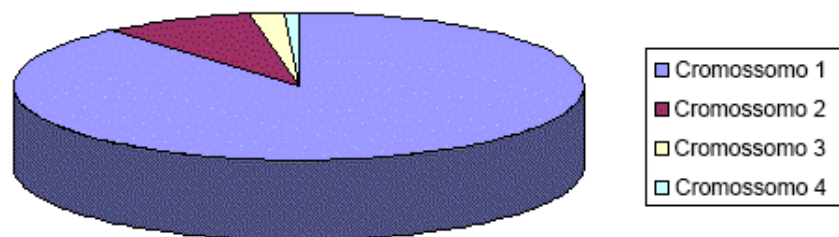


Figura 15 - Limitação da técnica da roleta (OBITKO, 1998).

Para resolver as limitações da técnicas da roleta, a **técnica de seleção por ranqueamento**, em vez de distribuir um valor proporcional ao *fitness*, os ‘N’ cromossomos de uma dada população recebem um valor fixado relativo ao seu desempenho na seguinte forma:

- Pior cromossomo recebe valor 1
- Cromossomos seguintes recebem valores $2 \dots N-1$
- Cromossomo de maior *fitness* recebe valor N

O resultado final do ranqueamento dos cromossomos da roleta na Figura 15 são apresentados de acordo com seu ranqueamento na Figura 16.

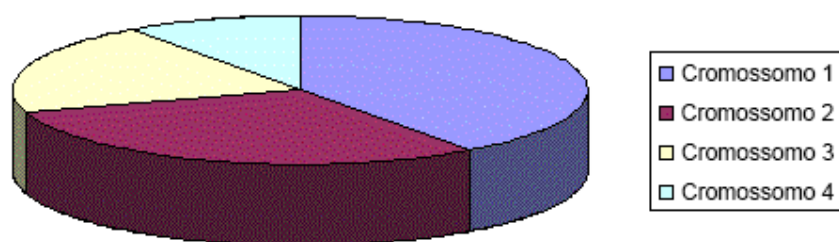


Figura 16 - Cromossomos ranqueados (OBITKO, 1998).

Apesar da vantagem em relação à técnica anterior, a técnica da roleta ainda possui uma convergência lenta.

A **seleção por torneio** é um método mais eficiente e possui maiores chances de alcançar a solução ótima (CHUDASAMA, SHAH e PANCHAL, 2011). Nesta técnica, dois ou mais indivíduos são aleatoriamente selecionados para participar do torneio. Neste torneio, o indivíduo que possuir o maior *fitness* será selecionado para a próxima rodada de cruzamentos. Estes torneios são repetidos até que o número de vencedores seja semelhante à quantidade de pais necessários para cruzamento da população nova.

A **seleção de Boltzmann** é inspirado na idéia da metaheurística de *Simulated Annealing*, a temperatura inicial admite que cromossomos de pior avaliação possam ser escolhidos com maior facilidade, porém, conforme esta temperatura decai, o algoritmo possui uma tendência em selecionar apenas cromossomos de maior *fitness* para cruzamento.

3.1.4.2. Operador de cruzamento

Após terminada a etapa de escolha dos indivíduos, é necessário então adotar alguma técnica para cruzamento dos pais, afim de gerar uma nova geração de indivíduos. Para isto, existem diversos operadores de cruzamento, que serão apresentados em detalhes nas seções seguintes.

Um ponto importante a ser lembrado é que o operador de cruzamento possui uma probabilidade associada, quando esta probabilidade atinge 0%, os novos filhos gerados serão idênticos aos pais, sem aplicar nenhuma recombinação. Em contrapartida, quando esta probabilidade atinge 100%, todos os novos indivíduos serão fruto desta recombinação. Segundo os autores (SIVANANDAM e DEEPA, 2008), é recomendável alocar uma porcentagem alta para criar uma diversidade de indivíduos, porém, ainda reservar uma pequena possibilidade de que os novos indivíduos não sejam afetados por cruzamento e sejam fruto de uma cópia idêntica dos pais. Note que, após o cruzamento, um algoritmo genético executa em seguida o operador de mutação sobre os filhos gerados. Portanto, caso o operador de cruzamento não seja acionado

e os filhos gerados sejam cópias dos pais, isto significa que apenas o operador de mutação é aplicado sobre estes cromossomos. Um filho idêntico ao pai que sofre mutação sem cruzamento pode ser um fator desejável, visto que uma solução perturbada pode gerar um *fitness* melhor em comparação com os pais que não sofreram esta mutação.

O **cruzamento de ponto único** é a técnica mais clássica em que dois pais são selecionados, um ponto de corte é escolhido aleatoriamente no cromossomo, daí os novos filhos são gerados a partir de uma recombinação dos pais, conforme ilustrado na Figura 17.

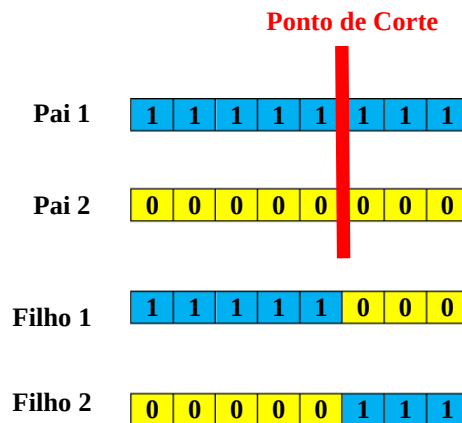


Figura 17 - Cruzamento de ponto único.

Esta técnica possui a desvantagem natural de que, caso as características distintas de ambos os pais estejam concentradas em uma parte extrema da String, existe uma menor probabilidade de que o ponto de corte seja localizado nesta posição, diminuindo a chance de que ambas as características distintas dos pais sejam herdadas por ambos os filhos.

Para tentar resolver esta limitação do cruzamento de ponto único, foi concebido a técnica de **cruzamento de dois pontos**, ilustrado na Figura 18. Apesar da técnica diminuir sensivelmente o desempenho do algoritmo genético, o espaço de busca é investigado com uma profundidade maior, causando uma maior perturbação à solução.

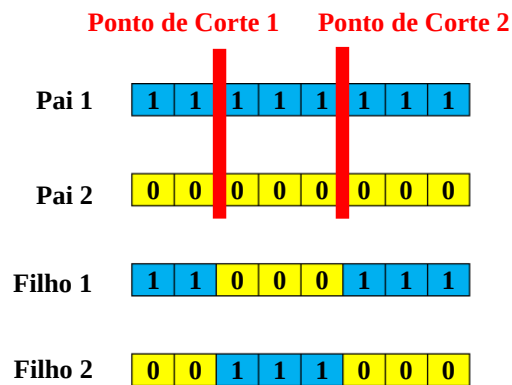
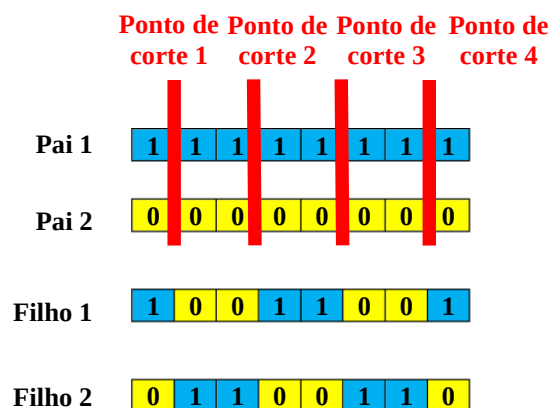


Figura 18 - Cruzamento de dois pontos

O **cruzamento de n -pontos** se trata de uma generalização da técnica anterior, em que, além dos pontos de corte serem selecionados aleatoriamente, desta vez o número de pontos de corte também é escolhido de modo aleatório. Podemos observar um exemplo disto na Figura 19.

Figura 19 - Cruzamento de n -pontos.

Esta técnica incorpora uma diversidade ainda maior à busca, porém, torna a convergência do algoritmo mais lenta.

O **cruzamento com substituto reduzido** se trata de uma variação do cruzamento de n -pontos. Nesta abordagem, os pontos de corte são fixados nas posições em que existem variância nos valores dos genes, de modo a evitar gerar clones dos pais.

Na técnica de **cruzamento uniforme** (*uniform crossover, UX*), inicialmente, cria-se uma máscara de bits do mesmo tamanho dos cromossomos selecionados para cruzamento. Esta máscara será preenchida com valores aleatórios, onde o valor '1' sinaliza a cópia do referido gene do primeiro pai, enquanto que o valor '0' sinaliza a cópia do respectivo gene do segundo pai, sendo esta lógica então invertida para a geração do segundo filho. Esta técnica promete

reduzir o viés associado com o comprimento da string, além de permitir fixar probabilidades individuais a cada posição para torná-lo tendencioso a ser preenchido com valores específicos de modo a se beneficiar de um possível conhecimento do domínio do problema. Outra possibilidade é fixar uma taxa de mistura, em que, caso este valor seja 50%, a máscara será enviesada de modo a copiar exatamente metade de genes de ambos os pais. Esta técnica está ilustrada na Figura 20.

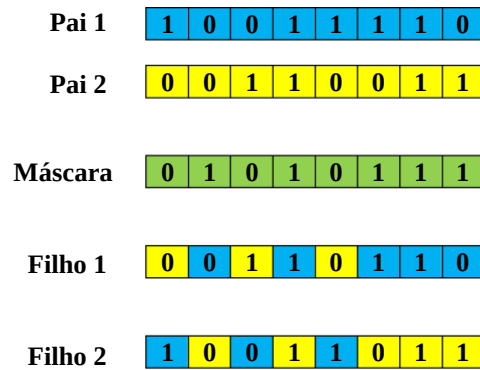


Figura 20 - Cruzamento uniforme.

O **cruzamento de metade uniforme** (*half uniform crossover, HUX*) ignora os elementos semelhantes em ambos os pais na mesma posição do cromossomo. O cruzamento é realizado apenas nas posições em que os valores são distintos entre os pais. Para isto, é calculado uma distância de Hamming para estimar a quantidade de genes distintos e esse número é dividido por dois, portanto, determinando que a taxa de mistura será sempre de 50% entre esses genes diferenciados. A Figura 21 ilustra o funcionamento desta técnica.

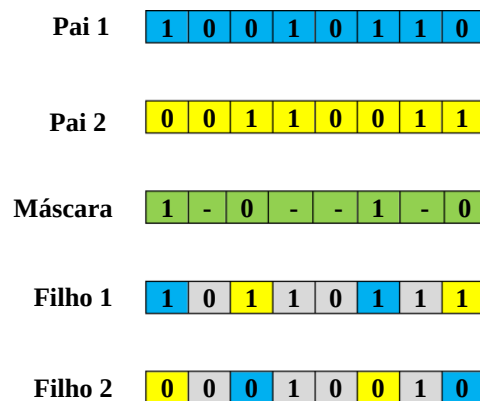


Figura 21 - Cruzamento de metade uniforme.

3.1.4.3. Operador de mutação

Sabemos que o operador de cruzamento não é suficiente para explorar o espaço de busca do problema. Também é necessário que novas características sejam levantadas pelo operador de mutação, largamente responsável por evitar que a solução se restrinja aos valores ótimos locais. Além disso, este operador é bastante relevante por garantir um certo grau de diversidade da população. Portanto, obtemos esta relação complementar entre cruzamento e mutação, no qual o primeiro é responsável por buscar melhores alternativas dentro do espaço de busca atual, representado pelos indivíduos da população, enquanto que o último auxilia na perturbação da população, acrescentando características inexistentes na população presente.

Além da taxa de cruzamento, o operador de mutação também deve ser configurado com uma taxa de mutação, que sinaliza a probabilidade do operador atuar sobre um dado cromossomo, a depender da técnica de mutação utilizada. Os autores divergem a respeito da taxa de mutação. Segundo (SIVANANDAM e DEEPA, 2008), este valor deve ser usualmente fixado em $1/L$, sendo ‘L’ o comprimento da string do cromossomo. Werner (WERNER, 2011), entretanto, apresenta um estudo que mostra que os melhores resultados são obtidos quando a taxa de mutação é fixada entre 1% a 10%. Vale lembrar que uma taxa de mutação de 100% determina que todas as posições de um cromossomo sofrerão mutação.

A técnica de **flipping** consiste em, dada uma taxa de mutação, este valor é testado a cada posição, sendo o bit invertido caso o operador de mutação seja acionado. Esta técnica é ilustrada na Figura 22. Pelo fato de cada bit ser testado de acordo com a probabilidade de mutação, a principal vantagem desta técnica consiste em ter um maior controle a respeito da quantidade média de bits que serão afetados por este operador. Por exemplo, para um problema com String de 10 bits e uma taxa de mutação de 10%, isto sinaliza que, em média, para cada cromossomo afetado pelo operador de mutação, ele terá um bit modificado.

Cromossomo original	1	0	1	1	0	1	0	1
Ativação da mutação	1	0	0	0	1	0	0	0
Cromossomo modificado	0	0	1	1	1	1	0	1

Figura 22 - Mutação por flipping de bits.

Também existe uma variação elitista desta técnica, no qual o operador de mutação é ativado quando a variação melhorar o valor de *fitness* de cromossomo, mas isto faz com que a solução seja afetada pelo problema dos ótimos locais, comprometendo o propósito da mutação.

A **intercambiação** pode ser utilizada em operadores binários, mas é mais adequada para problemas com codificação que admite valores não-binários. Ela consiste em selecionar duas posições aleatórias da string e inverter a posição destes bits. Portanto, a probabilidade de mutação define apenas se este operador deverá selecionar 2 bits a serem modificados no cromossomo, independentemente do tamanho da String, o que pode ser problemático no caso de problemas com String de bits muito grandes.

A **reversão** consiste em, caso o operador de mutação seja acionado, os valores das posições vizinhas são invertidos em conjunto com a posição escolhida. Isto é ilustrado na Figura 23. Naturalmente, a média de bits afetados pelo operador de mutação é dobrado em comparação com o operador de *flipping*. Para o mesmo caso anterior, de uma String com 10 bits e uma taxa de mutação de 10%, no caso da reversão implica que, em média, 2 bits dos cromossomos serão modificados. Além disso, outra possível desvantagem se trata da proximidade dos valores afetados que sempre serão separados por apenas um bit intercalado. Note que, no caso de um *flipping* com probabilidade de 20% para uma String de 10 bits, os 2 bits que, em média, serão afetados não serão necessariamente próximos um do outro, qualquer posição da String poderá ser modificada. Este comportamento não-determinístico a respeito da posição da String que deverá sofrer mutação pode ser vista como uma outra vantagem do operador de *flipping* em comparação com a reversão.

Cromossomo original	1	0	1	1	0	1	0	1
Ativação da mutação	0	0	0	0	1	0	0	0
Cromossomo modificado	1	0	1	0	1	0	0	1

Figura 23 - Exemplo de mutação por reversão.

3.1.4.4. Operador de substituição

Após o cruzamento e a mutação, devemos empregar alguma técnica responsável por inserir estes novos indivíduos na população que serão armazenados para a próxima iteração do algoritmo.

A **substituição geracional** é a técnica mais simples em que N pais geram N filhos e todos estes filhos irão compor a nova população. Isto ocasiona a perda de boas soluções encontradas em iterações passadas. Esta técnica, apesar de reproduzir fielmente o comportamento biológico das espécies, é raramente usada.

A **substituição estacionária** garante que os cromossomos da população antiga possam persistir na nova geração. As métricas de substituir os piores membros da população ou os membros mais velhos pode ser empregado, mas ocasiona uma alta pressão de seleção de cromossomos em que os cromossomos de piores *fitness* sempre serão descartados, o que pode ocasionar em uma convergência prematura da solução, sem que ela se aproxime do valor ótimo, portanto, é raramente utilizado (SIVANANDAM e DEEPA, 2008). Esta técnica é mais comumente empregada usando técnicas de substituição por torneio, idêntico à seleção por torneio, ou substituir por similaridade entre cromossomos pais e filhos.

No caso da **substituição aleatória**, em um problema que possua uma população pequena, pode ser interessante substituir aleatoriamente uma determinada quantidade de cromossomos da população, selecionados tanto entre pais como filhos. Mas, naturalmente, dada a aleatoriedade do processo esta técnica não fornece garantia de persistência de cromossomos com maior *fitness*.

Em técnicas de cruzamento responsáveis por gerar dois filhos, a **substituição do pai mais fraco** ocorre com a seleção do pai e filho que possuam o maior valor de *fitness*. Porém, esta técnica pode ser redundante quando cromossomos fracos raramente são selecionados para reprodução.

No caso da **substituição de ambos os pais**, em contraste com a técnica anterior, esta substituição somente se aplica a técnicas de cruzamentos que geram um único filho, sendo assim, ‘N’ pais geram ‘N/2’ filhos, enquanto que o restante da população são selecionados dentre os cromossomos da população anterior.

Finalmente, o **elitismo** não é uma técnica de substituição *per se*, mas uma estratégia que pode ser combinada a qualquer outra técnica de substituição. Neste caso, o elitismo irá separar uma determinada porcentagem de cromossomos de maior *fitness* que deverão persistir na próxima iteração. O restante dos indivíduos são substituídos por qualquer outra técnica existente. A motivação por trás do elitismo é garantir que soluções de grande interesse não sejam perdidas devido à relativa aleatoriedade presente no processo de substituição.

3.1.5. Critérios de parada

Conforme discutido anteriormente, os algoritmos genéticos possuem uma limitação na definição do critério de parada em relação ao *Simulated Annealing*, em que este último encerra sua operação quando ele não consegue melhorar a solução obtida em relação à vizinhança.

Portanto, para os algoritmos genéticos, é necessário sinalizar algum critério manual a ser configurado e testado de modo a encontrar o melhor valor possível para o problema a ser resolvido.

Dentre os critérios existentes, o mais simples se trata de definir um número máximo de iterações (gerações) do algoritmo. O limite de gerações pode ser aplicado em conjunto com um tempo-limite, encerrando sua execução ao satisfazer o primeiro critério alcançado.

Outras técnicas existentes se tratam de encerrar a execução após um valor ‘N’ de gerações sem obter uma mudança no valor geral do *fitness* da população. Uma variação deste critério pode ser obtida pela presença de uma geração estagnada, em que o algoritmo pode ser encerrado caso a mudança no valor geral de *fitness* seja menor que um dado valor de tolerância configurável.

3.2. Algoritmos Genéticos Multiobjetivos

Apesar das diversas vantagens levantadas em favor dos algoritmos genéticos, note que, de maneira simplista, a solução consiste em se aproximar de um valor ótimo global para um dado problema. Em outras palavras, o algoritmo está em busca de um ponto dentro de uma região de possíveis soluções. Para uma modelagem clássica do problema do próximo release, conforme apresentada no capítulo anterior, não haveria problemas. Porém, caso o problema seja modificado para representar múltiplos objetivos a serem otimizados, este algoritmo é insuficiente para realizar esta tarefa. Portanto, o algoritmo genético deve ser modificado, de modo a ser capaz de lidar com este desafio.

Felizmente, denotamos que, dentre as vantagens dos GAs, eles podem ser facilmente modificados para tratar este tipo de problemática multiobjetiva. Nas próximas seções, elencamos três algoritmos distintos que utilizam os conceitos de algoritmos genéticos apresentados na seção anterior e são modificados de forma a refletir esta nova realidade de otimização multiobjetiva. Novamente, ressaltamos que não basta um algoritmo ser capaz de lidar com vários objetivos simultâneos, mas que também é desejável que este algoritmo produza um conjunto de soluções com um bom espalhamento e que se aproxime da Fronteira de Pareto formada pelo conjunto ótimo de Pareto, conforme discutido no capítulo anterior deste documento.

Os algoritmos selecionados foram o NSGA-II (DEB *et al.*, 2002), SPEA2 (ZITZLER, LAUMANN e THIELE, 2001) e MOCell (NEBRO *et al.*, 2009). O primeiro algoritmo, conforme observado na Tabela 2 em que foi levantado uma série de trabalhos relacionados com

o tema, é comumente utilizado como um algoritmo de referência nos problemas do próximo release com objetivos diversos, sendo sempre utilizado como fator de comparação a um novo algoritmo capaz de tratar múltiplos objetivos. O SPEA2, por sua vez, argumenta ser capaz de obter resultados tão competitivos quanto os do NSGA-II. Finalmente, o MOCell possui uma característica singular de incorporar estratégias de busca local na sua implementação. Conforme discutido por Harman *et al.* (HARMAN *et al.*, 2012), é salutar, quando estabelecer comparações de resultados entre diferentes algoritmos, escolher, adicionalmente a um algoritmo de busca global, algum outro que utilize uma estratégia de busca local. O MOCell neste sentido nos oferece uma opção interessante que incorpora ambas as estratégias de maneira explícita.

Tabela 2 - Propostas para o problema do próximo release ou para o problema de planejamento de releases que utilizam os algoritmos selecionados.

Algoritmo	Total	Trabalhos que o utilizam
NSGA-II	8	(DURILLO <i>et al.</i> , 2011), (DURILLO <i>et al.</i> , 2009), (CAI, WEI e HUANG, 2012), (LI <i>et al.</i> , 2014), (PITANGUEIRA <i>et al.</i> , 2016), (COLARES <i>et al.</i> , 2009), (LI <i>et al.</i> , 2016), (SARAIVA <i>et al.</i> , 2016)
SPEA2	1	(CAI, WEI e HUANG, 2012)
MOCell	3	(DURILLO <i>et al.</i> , 2011), (DURILLO <i>et al.</i> , 2009), (SARAIVA <i>et al.</i> , 2016)

Outros pesquisadores independentes confirmam essa hipótese. Podemos citar o trabalho de Pitangueira *et al.* (PITANGUEIRA, MACIEL e BARROS, 2015) que realizou uma revisão sistemática de 39 trabalhos a respeito do histórico problema do próximo release, desde sua concepção original em 2001 (BAGNALL, RAYWARD-SMITH e I. M. WHITTLEY, 2001), e do problema de planejamento de releases, demonstrando os principais algoritmos utilizados por eles, reproduzido na Figura 24. Esta figura apresenta uma quantidade diferente de artigos, visto que esta revisão sistemática apresenta um histórico de 39 trabalhos de próximo release, mas que nem todos se relacionam diretamente com o tema proposto.

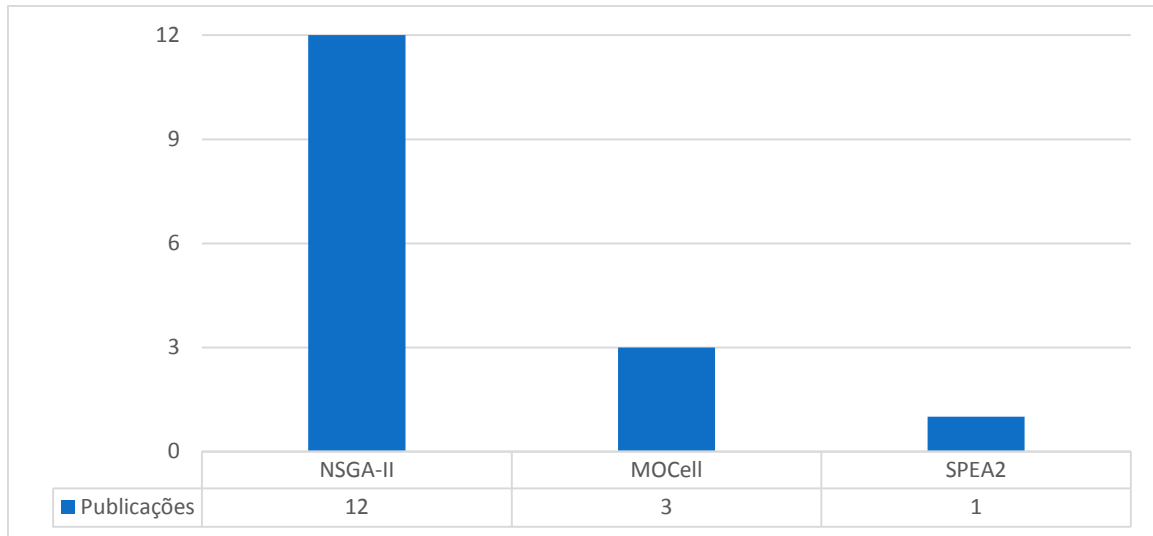


Figura 24 - Ocorrência dos algoritmos selecionados dentre os 39 estudos listados em (PITANGUEIRA, MACIEL e BARROS, 2015).

O trabalho de Colanzi *et al.* (COLANZI *et al.*, 2012) realiza uma revisão sistemática dos trabalhos de SBSE no Brasil, citando que 48% dos estudos selecionados adotam uma abordagem multiobjetiva, em que o NSGA-II é o principal algoritmo utilizado. Esta revisão adicionalmente faz menção aos algoritmos MOCell e SPEA2, descrevendo este último como similar ao NSGA-II.

3.2.1. NSGA-II

O algoritmo NSGA-II (DEB *et al.*, 2002) surgiu da necessidade de melhorar algumas limitações presentes no NSGA original (SRINIVAS e DEB, 1995), um dos primeiros algoritmos evolucionários capazes de lidar com múltiplos objetivos. Dentre as limitações existentes no NSGA, os autores afirmam:

1. Alta complexidade computacional: o algoritmo original possuía uma complexidade na ordem $O(MN^3)$, em que ‘M’ é o número de objetivos, enquanto que ‘N’ representa o tamanho da população. O algoritmo NSGA-II consegue melhorar este aspecto com um algoritmo com complexidade da ordem $O(MN^2)$;
2. Ausência de elitismo: os autores demonstram que incorporar alguma técnica de elitismo no algoritmo original pode agilizar a performance, garantindo que boas soluções não sejam perdidas assim que forem encontradas;
3. Necessidade de configuração do espalhamento: uma variável de compartilhamento deve ser configurada e usada no algoritmo original para ajustar o espalhamento da solução. Observando a grande quantidade de parâmetros a serem configurados em um algoritmo

genético, é desejável que esta variável seja eliminada, sendo esta técnica substituída por alguma outra que não precisa de configuração prévia.

O ciclo de execução do NSGA-II em sua inicialização possui quatro passos distintos:

1. Gerar a população inicial;
2. Atribuição de dominância dos cromossomos conforme a regra de dominância de Pareto discutida no capítulo 2 deste trabalho;
3. Separação dos cromossomos em fronteiras (*fronts*), onde as melhores soluções são alocadas no primeiro front, as soluções que são dominadas pelas soluções do primeiro front são classificadas no segundo front e assim sucessivamente;
4. Classificação de espalhamento: atribuindo um valor de distância para cada cromossomo em relação aos seus vizinhos, de modo que os cromossomos com maior valor de distância possuam a melhor avaliação.

A geração da população é um passo trivial, portanto, iniciaremos detalhando a segunda e terceira etapas do algoritmo, responsáveis por realizar a atribuição de dominância da população inicial, assim como sua classificação em algum front.

3.2.1.1. Atribuição de dominância e separação em fronts

No início do passo de atribuição de dominância, denominada “*fast non-dominated sort*”, para cada indivíduo $\mathbf{p}$ da população $\mathbf{P}$, calcula-se um valor n_p , que representa o número de soluções que dominam este elemento $\mathbf{p}$. Além disso, também deve-se formar um subconjunto S_p , que indica quais elementos da população são dominados pelo indivíduo $\mathbf{p}$. Este passo inicial tem complexidade $O(MN^2)$, o que define a complexidade maior do algoritmo.

É intuitivo observarmos que, ao final deste passo de atribuição de dominância, os indivíduos que não são dominados por nenhuma outra solução possuirá um valor n_p igual a zero. Por tanto, já iniciando o passo de separação em fronteiras, este conjunto de valores são classificados no primeiro front da solução. Adicionalmente, cada elemento do subconjunto S_p para todos os indivíduos do primeiro front devem ter seus respectivos valores n_p decrementados em 1. Estes elementos com valores reduzidos, caso atinjam um valor igual a zero, fazem agora parte do segundo front da solução. Este procedimento é então repetido até que todos os valores sejam classificados em algum front. O Algoritmo 2 detalha este procedimento.

Algoritmo 2 - Atribuição de dominância e classificação em fronts do NSGA-II.

01	Fast-non-dominated-sort(P)
02	for each $p \in P$
03	$S_p = \emptyset, n_p = 0$
04	for each $q \in P$
05	if ($p < q$) then //se p domina q
06	$S_p = S_p \cup \{q\}$ //adicionar q ao conjunto de soluções dominadas por p (S_p)
07	else if($q < p$) then //se p for dominado por q , então o contador n_p deve ser incrementado
08	$n_p = n_p + 1$
09	if $n_p = 0$ then //ao final do loop for, todos os indivíduos p que possuírem valor $n_p = 0$ pertencem ao primeiro front
10	$p_{rank} = 1$
11	$F_1 = F_1 \cup \{p\}$
12	$i = 1$ //inicializar o contador de fronts
13	while $F_i \neq \emptyset$
14	$Q = \emptyset$ //usado para armazenar elementos do próximo front
15	for each $p \in F_i$
16	for each $q \in S_p$
17	$n_q = n_q - 1$
18	if $n_q = 0$ then //q pertence ao próximo front
19	$q_{rank} = i + 1$
20	$Q = Q \cup \{q\}$
21	$i = i + 1$
22	$F_i = Q$

3.2.1.2. Classificação de espalhamento

O cálculo de espalhamento para qualquer ponto é estimado em relação aos pontos vizinhos, sendo a distância dada pela média normalizada com base em todos os ' m ' objetivos do problema. A região final que representará a soma total das distâncias entre pontos vizinhos é denominada "cubóide". A Figura 25 ilustra um exemplo para dois objetivos ' f_1 ' e ' f_2 '. É importante destacar que os indivíduos que se encontrarem nos pontos extremos da região de busca recebem um valor de distância infinito, de modo a tornar desejável uma otimização do espalhamento, mesmo que necessariamente estas soluções nos extremos não possuam um valor de *fitness* razoável em comparação com os demais pontos encontrados.

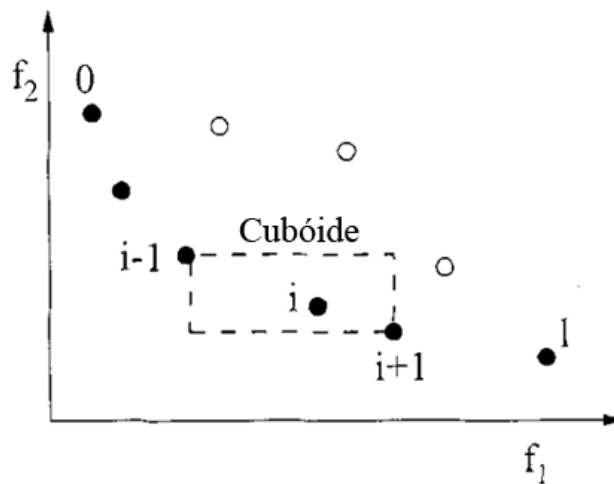


Figura 25 - Definição de um cubóide para um ponto 'i' com dois objetivos f_1 e f_2 .

O Algoritmo 3 descreve o cálculo do espalhamento para cada cromossomo dentro de um conjunto não-dominado L .

Algoritmo 3 - Cálculo de distância do NSGA-II.

```

01 crowding-distance-assignment( $L$ )
02    $l = |L|$       //o número de soluções em  $L$ 
03   for each  $i$ , set  $L[i]_{distance} = 0$       //inicializar o cálculo de
                                                distância
04   for each objective  $m$ 
05        $L = sort(L, m)$       //ordenar os elementos de  $L$  de acordo com o
                                valor do objetivo  $m$ 
06        $L[1]_{distance} = L[l]_{distance} = \infty$       //atribuir valor infinito aos
                                                        pontos extremos
07       for  $i = 2$  to  $(l - 2)$ 

```

08

$$L[i]_{distance} = L[i]_{distance} + \frac{L[i+1].m - L[i-1].m}{f_m^{max} - f_m^{min}}$$

Na linha 08 do Algoritmo 3, o valor $L[i].m$ se refere ao m -ésimo valor da função objetivo do i -ésimo cromossomo no conjunto L . Portanto, a equação realiza uma diferença entre os valores de função dos cromossomos vizinhos. Este valor é dividido pela diferença dos valores máximo e mínimo de função para o objetivo ' m ', que, naturalmente, podemos intuir que estes valores estão presentes nos cromossomos 1 e ' l ' para aquele dado conjunto ' L '. Finalmente, este valor calculado é acrescido ao valor de distância do cromossomo ' i ' em relação aos demais objetivos já calculados, afim de obter o valor de distância final. Segundo os autores, este procedimento possui uma complexidade na ordem de $O(MN \log N)$, novamente admitindo ' M ' como sendo o número de objetivos, enquanto que ' N ' representa o tamanho da população.

Note ainda que este operador de distância não é executado em relação à todos os elementos da população, mas apenas dentre os elementos de um mesmo front, classificados na etapa anterior. Isto ocorre pois, conforme observado na Figura 26, dada a característica elitista do algoritmo, os elementos classificados nos primeiros fronts são naturalmente aceitos na população futura.

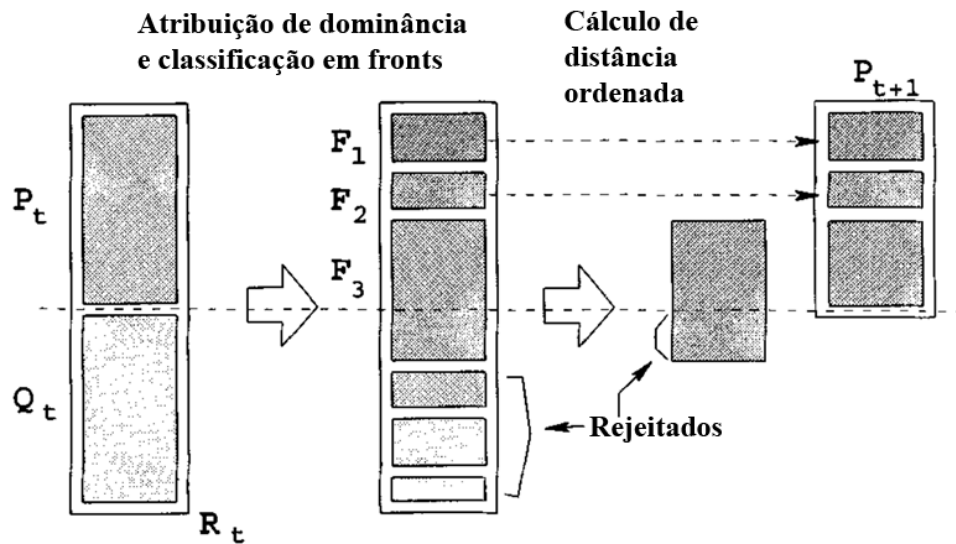


Figura 26 - Etapas de classificação e substituição de cromossomos no NSGA-II a partir de uma população P_t e um conjunto de filhos Q_t .

Conforme a solução for convergindo após sucessivas iterações, poderemos observar que todos os elementos irão progressivamente participar dos primeiros fronts, exigindo que o operador de distância seja utilizado como um critério de desempate para gerar o novo conjunto de pais para criar o conjunto de filhos da próxima geração.

Para substituir a população, o operador de comparação de densidade ($<_n$) é definido a fim de selecionar os elementos presentes na próxima iteração, recebendo como entrada o valor do front e a distância de espalhamento. Uma dada solução ' i ' é melhor que uma outra solução ' j ', ou, $i <_n j$ se:

- Front $i < \text{front } j$;
- (Front $i = \text{front } j$) $\wedge$ (distância $i > \text{distância } j$)

3.2.1.3. Loop principal do NSGA-II

Finalmente, em posse de todos estes conceitos definidos nas seções anteriores, podemos descrever o ciclo de execução do NSGA-II no Algoritmo 4.

Algoritmo 4 - Ciclo principal do NSGA-II.

01	$R_t = P_t \cup Q_t$	//criar um conjunto único de pais e filhos de tamanho 2N
02	$F = \text{fast-non-dominated-sort}(R_t)$	//atribuição de dominância
03	$P_{t+1} = \emptyset$ and $i = 1$	
04	until $ P_{t+1} + F_i \leq N$	//incluir fronts na nova população de pais até que o tamanho do fronts selecionados até o momento exceda o tamanho N da população
05	$\text{crowding-distance-assignment}(F_i)$	//cálculo de distância
06	$P_{t+1} = P_{t+1} \cup F_i$	//incluir o front na nova população de pais
07	$i = i + 1$	//incrementar e testar o próximo front i+1
08	$\text{sort}(F_i, <_n)$	//ordenar os elementos do último front i a fim de preencher exatos N elementos na população de pais
09	$P_{t+1} = P_{t+1} \cup F_i[1:(N - P_{t+1})]$	//escolha os primeiros (N - P _{t+1}) elementos de F _i para preencher o restante da solução
10	$Q_{t+1} = \text{make-new-pop}(P_{t+1})$	//utilizar os operadores de seleção, cruzamento e mutação para gerar a nova população Q _{t+1}
11	$t = t + 1$	//incrementar o contador de geração

3.2.2. SPEA2

De maneira semelhante ao NSGA-II, o SPEA2 (ZITZLER, LAUMANNNS e THIELE, 2001) representa uma modificação em relação ao algoritmo original proposto em 1999

(ZITZLER e THIELE, 1999). O algoritmo original garantia o elitismo utilizando um **arquivo**, definido por um conjunto de soluções separado da população, que contém todos os cromossomos não-dominados em relação aos demais. Para manter o arquivo, o algoritmo elimina soluções repetidas e indivíduos dominados por algum outro membro da população. O SPEA2 promete melhorar o método de truncamento do arquivo, de modo a garantir soluções presentes nos extremos da curva de Pareto. Além disso, a função de *fitness* é modificada, assim como também a função de avaliação de vizinhança, de modo a auxiliar o processo de busca.

Originalmente, o SPEA admitia duas métricas, sendo elas uma medida de **‘Força’** e outra de **‘Fitness’**, definidas respectivamente nas Equações 8 e 9.

$$\text{Força: } S(i) \in [0, 1) = \frac{\text{soluções dominadas ou equivalentes a } i}{\text{tamanho da população} + 1} \quad (8)$$

$$\text{Fitness: } F(i) = 1 + \sum s(j), \text{ onde } j \text{ são todos os indivíduos dominados ou equivalentes ao cromossomos } i \quad (9)$$

Durante o cruzamento, os membros do arquivo possuem mais chances de serem selecionados do que os membros da população. A população é renovada sempre a cada iteração, enquanto que o arquivo é persistente.

Apesar destas características, o SPEA apresenta uma série de desvantagens:

- Indivíduos dominados por soluções semelhantes do arquivo possuem fitness similar. No pior caso em que o arquivo contém apenas um único indivíduo, o algoritmo possui um comportamento igual à uma busca aleatória;
- Se muitos indivíduos de uma dada geração não dominam uns aos outros, pouca informação pode ser extraída dos dados de *fitness* e força. Um valor de espalhamento se torna necessário a fim de melhor informar quais soluções devem persistir no arquivo, porém, o SPEA considera apenas o espalhamento interno do arquivo, desprezando os elementos da população regular, diminuindo a diversidade;
- O SPEA possui uma técnica de clusterização com objetivo de manter as características das soluções do arquivo, porém, isto prejudica o espalhamento das soluções, caso elas fujam do padrão de características.

3.2.2.1. Loop principal do SPEA2

Para resolver estes problemas, o SPEA2 modifica a função de atribuição de *fitness* e introduz um novo operador de truncamento, realizado durante o processo de seleção ambiental, conforme definido no Algoritmo 5.

Algoritmo 5 - Etapas de execução do SPEA2.

01	Dados de Entrada	N , tamanho da população
02		$\overline{N}$, tamanho do arquivo
03		T , número de iterações
04	Dados de Saída	A , conjunto de soluções não-dominadas
05	Inicialização: Gere uma população inicial $P_0 = \emptyset$, um arquivo $\overline{P}_0 = \emptyset$, e defina a iteração inicial $t = 0$	
06	Atribuição de fitness: o valor de fitness dos indivíduos do arquivo $\overline{P}_t$ e da população regular P_t são calculados	
07	seleção ambiental: copie todos os elementos não dominados em P_t e $\overline{P}_t$ para $\overline{P}_{t+1}$. se $\overline{P}_{t+1}$ exceder $\overline{N}$, então reduza $\overline{P}_{t+1}$ por meio de um operador de truncamento. Caso contrário, selecione elementos dominados da população P_t e do arquivo $\overline{P}_t$ para preencher o arquivo.	
08	Encerramento: se $t \geq T$, ou se algum outro critério de parada for atingido, então retorne os valores de $\overline{P}_{t+1}$ como o arquivo final A e encerre a execução do algoritmo.	

3.2.2.2. Atribuição de fitness

Diferentemente do algoritmo original, o valor final de *fitness* leva em consideração as soluções presentes tanto no arquivo quanto na população regular. Isto é feito de modo a evitar valores de *fitness* similares. O cálculo de força ‘ S ’ é modificado, indicando a quantidade de soluções que dominam o indivíduo ‘ i ’. O valor de *fitness* é modificado para um novo “*raw fitness*” definido na Equação 10.

$$R(i) = \sum_{j \in P_t \cup \overline{P}_t, j > i} S(j) \quad (10)$$

Esta equação realiza a soma das forças ‘ S ’ de todos os elementos que dominam ‘ i ’. Portanto, o *fitness* é um valor a ser minimizado, assumindo valor igual a zero quando nenhuma outra solução atual domina o indivíduo. Este novo cálculo elimina o primeiro problema do SPEA. A Figura 27 ilustra, para um mesmo problema com duas funções objetivo, como o cálculo de *fitness* é capaz de gerar valores distintos entre o SPEA original e o SPEA2.

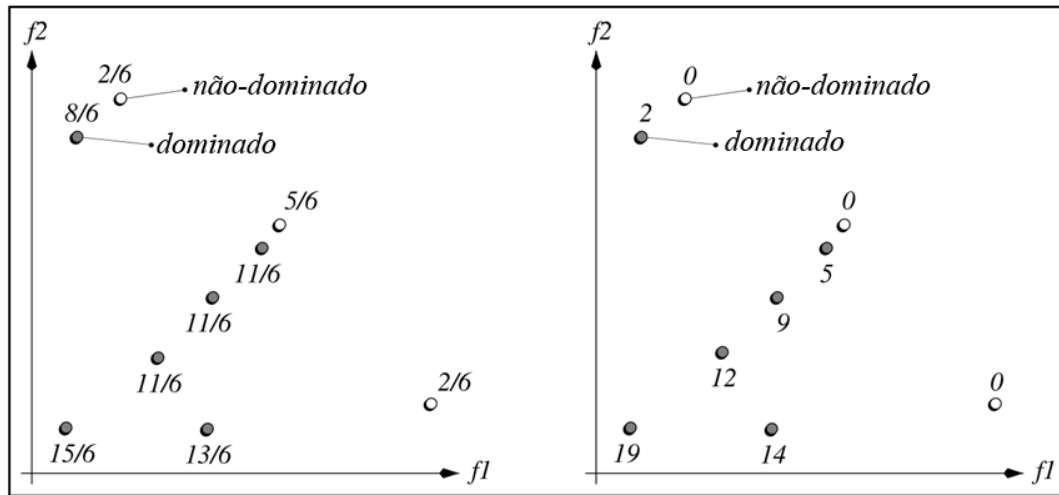


Figura 27 - Comparação da atribuição de fitness entre o SPEA e SPEA2 para dois objetivos.

Apesar da vantagem do novo cálculo de *fitness* evitar produzir valores distintos, ele ainda não resolve o segundo problema do algoritmo original, já que os valores de *fitness* ainda serão similares, como ainda podemos notar na Figura 24, ao visualizarmos os elementos não-dominados. Portanto, para uma situação em que muitos indivíduos são não-dominados, é necessário alguma informação adicional capaz de diferenciá-los. Deste modo, a função *fitness* incorpora um novo valor de densidade, somado ao “*raw fitness*”, visto na Equação 11.

$$F(i) = R(i) + D(i) \quad (11)$$

Este novo valor de densidade calcula o valor inverso à distância dos k -vizinhos mais próximos, de maneira similar ao algoritmo k -NN, *k-th Nearest Neighbor* (SILVERMAN, 1986), conforme podemos observar na Equação 12. O valor de distância no denominador é somado ao valor dois para evitar uma divisão por zero e garantir que o valor de densidade seja sempre menor que 1.

$$D(i) = \frac{1}{\sigma_i^k + 2}, k = \sqrt{N + \bar{N}} \quad (12)$$

3.2.2.3. Seleção ambiental

Adicionalmente aos passos descritos no loop principal do algoritmo, o operador de truncamento dá uma atenção especial para o valor de densidade. Caso o tamanho do novo arquivo exceda o tamanho máximo definido na entrada, os elementos com as menores distâncias para os k -vizinhos são removidos. Caso os elementos excedentes possuam uma distância similar, o valor de distância é alterado para se relacionar apenas com o vizinho mais próximo. Caso o empate persista, a comparação é realizada em relação ao segundo vizinho mais próximo, posteriormente, com o terceiro vizinho mais próximo, caso a situação permaneça indefinida, e assim sucessivamente até que um valor diferente seja encontrado.

3.2.3. MOCeII

Os algoritmos genéticos observados até o momento tratam a população como uma entidade singular, em que os operadores estocásticos são aplicados de maneira uniforme sobre este conjunto completo. Esta técnica é denominada de panmixia.

Entretanto, observamos que uma das vantagens dos GAs é a capacidade de paralelização, o que deu origem aos algoritmos genéticos distribuídos, capazes de trabalhar em diferentes regiões do espaço de busca, sendo estas soluções compartilhadas quando elas possuem alguma relação com o espaço de busca sendo investigado por outra instância do algoritmo. Esta classes de GAs recebe a denominação de Estratégias Evolucionárias Estruturadas, estes algoritmos distribuídos são capazes de fornecer uma melhor amostragem do espaço de busca, visto que o espaço é dividido em uma quantidade considerável de sub-regiões em comparação com um algoritmo genético clássico que trata o espaço de busca completo como uma entidade singular no modo de panmixia.

Finalmente, no caso mais extremo de segmentação do espaço de busca, observamos os Algoritmos Genéticos Celular, ou cGAs. Neste cenário, os cromossomos estão restritos a interagir apenas com os indivíduos existentes na vizinhança durante o processo de reprodução. Esta sobreposição de microrregiões vizinhas favorece a diversificação das soluções, que, no contexto dos algoritmos multi-objetivos, é representada pelo espalhamento das soluções, por conta de uma difusão lenta de boas soluções entre os membros da população. Em contrapartida, a intensa atividade entre cromossomos vizinhos auxilia nas buscas locais, aproximando-se do valor ótimo local para cada vizinhança estabelecida. A Figura 28 estabelece uma comparação entre estas três abordagens descritas até então. Outra vantagem apontada pelos autores no uso de algoritmos genéticos celulares e que podemos observar na Figura 28 se trata em criar um

mecanismo implícito de migração, caso seja encontrado um conjunto reduzido de soluções que possuam um *fitness* muito mais elevado em comparação com o restante da população atual, isto não irá influenciar significativamente o comportamento geral do algoritmo e causando uma convergência prematura do algoritmo. Sabendo que as vizinhanças são sobrepostas, em um processo de seleção de pais para reprodução, uma solução muito melhores que as demais podem afetar, no pior caso, apenas sua própria vizinhança e as vizinhanças adjacentes, não afetando então o processo de seleção para os indivíduos que não façam parte de alguma destas vizinhanças afetadas.

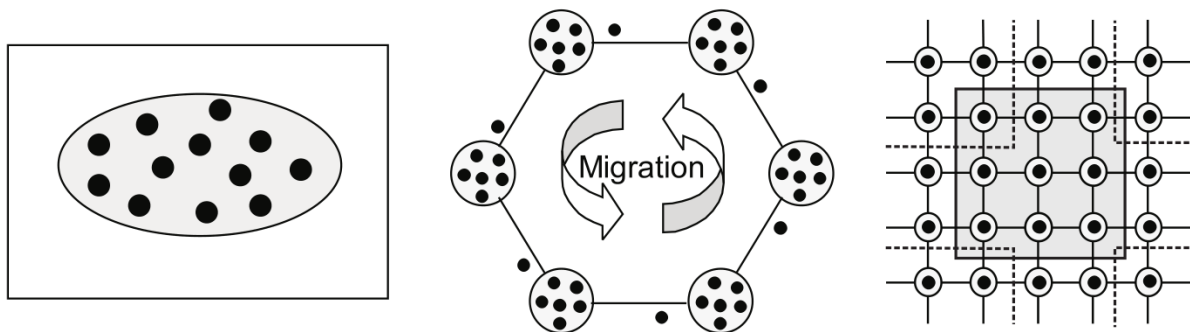


Figura 28 - Técnicas de panmixia, distribuição e organização celular do espaço de busca.

Estes algoritmos genéticos celulares foram propostos inicialmente para serem empregados em máquinas com um alto grau de paralelismo, apesar que o modelo em si é perfeitamente adaptável para o uso de máquinas com apenas um único núcleo de processamento. Além disso, a vizinhança é definida em razão da proximidade das soluções candidatas encontradas até o mesmo, o que não quer dizer necessariamente que um problema de otimização deve ser quebrado em partes distintas que são executadas por processadores diferentes.

Neste contexto, o MOCcell (NEBRO *et al.*, 2009) é um algoritmo multi-objetivo que emprega a distribuição celular da região de busca. De maneira semelhante ao SPEA2, o MOCcell também emprega um arquivo com objetivo de armazenar as melhores soluções encontradas de modo a garantir o elitismo das soluções, enquanto que a população regular é inteiramente renovada a cada ciclo de execução. Porém, diferentemente do algoritmo anterior, o MOCcell possui um mecanismo de *feedback*, em que algumas soluções existentes na população são trocadas aleatoriamente por membros do arquivo.

Um algoritmo genético celular qualquer é definido de maneira tal que a população é usualmente estruturada em uma grade com d dimensões ($d = 1, 2, 3$) e uma vizinhança é definida

sobre esta população. O algoritmo avalia iterativamente cada indivíduo na grade de vizinhança, então, de acordo com a técnica de seleção empregada, os pais são escolhidos dentre estes vizinhos. Após a seleção, os operadores de cruzamento e mutação são acionados condicionadas a alguma probabilidade de cruzamento e uma outra probabilidade de mutação. Após isso, o algoritmo deve computar o *fitness* de cada indivíduo gerado e recoloca-lo na posição da população em que os pais se encontram em uma população auxiliar. A nova população é obtida selecionando membros tanto da população atual quanto os indivíduos contidos na população auxiliar de acordo com algum método de substituição empregado. Após esta etapa, uma condição de parada é testada e, caso ela não seja satisfeita, uma nova etapa de seleção de indivíduos em cada grade para compor um novo conjunto de pais é reiniciado.

O Algoritmo 6 mostra o ciclo de execução principal do MOCeII. Apesar da grande quantidade de semelhanças com um algoritmo genético celular apresentado acima, dada a natureza do MOCeII em resolver problemas de otimização multi-objetivo, a fronteira de Pareto com as melhores soluções não-dominadas conhecidas até o momento devem ser armazenadas. Para isto, o arquivo previamente mencionado é utilizado, de maneira semelhante ao SPEA2 e que, ao final do ciclo de execução do algoritmo, este arquivo será considerado a saída do MOCeII.

Algumas considerações importantes devem ser consideradas. A primeira é que o número de vizinhos pode ser configurável, e, caso este parâmetro não seja definido explicitamente, o algoritmo assume que a vizinhança contém 8 indivíduos. Outra configuração típica é utilizar uma vizinhança com 4 indivíduos. Um detalhe importante do MOCeII a ser considerado é que, apesar do formato padrão vizinhança conter 8 indivíduos, o indivíduo selecionado para ser o elemento central desta vizinhança também faz parte da vizinhança, totalizando 9 membros da população por vizinhança. Este formato de vizinhança que considera um elemento central e seus vizinhos ao norte, sul, leste, oeste e as diagonais do elementos é denominado de “nove compacto” ou vizinhança de Moore (DORRONSORO e ALBA, 2008).

Na linha 10 do Algoritmo 6, os filhos gerados após as etapas de cruzamento e mutação (linhas 07, 08) são reinseridos na mesma posição dos pais na “população auxiliar”, que representa a população que será utilizada na próxima iteração do algoritmo, compondo a população regular (linha 12). Para decidir se os cromossomos devem ser inseridos no arquivo (linha 11), o MOCeII utiliza um operador de comparação de densidade semelhante ao descrito no NSGA-II, descrito na seção 3.2.1.2.

Algoritmo 6 - Loop principal do MOCeLL.

```

01 Proc Inicio(mocell)      //Inicializar parâmetros do algoritmo
02 Front-Pareto = Criar-Front()      //Criar front vazio
03 while not CondicaoParada() do
04     for each indivíduo, 1 to mocell.tamanho_população do
05         n_lista := Retornar_vizinhança(mocell,posição(indivíduo))
06         pais := Selecao(n_lista)
07         filhos := Crossover(propCrossover,pais)
08         filhos := Mutação(probMutacao,filhos)
09         Avaliar_fitness(filhos)
10         Inserir(posição(indivíduo),filhos,mocell,pop_aux)
11         Inserir_Front-Pareto(indivíduo)      //inserir no arquivo
12     mocell.população := pop_aux;
13     Mocell.população := Feedback(mocell,Front_Pareto)

```

Também notamos que o procedimento de *feedback* é acionado com objetivo de selecionar um número fixo de indivíduos escolhidos aleatoriamente dentre o arquivo e inseridos na nova população (linha 13). Este mecanismo, segundo (DORRONSORO e ALBA, 2008), é uma inclusão importante em auxiliar uma melhor exploração do espaço de busca e evitando uma convergência prematura da solução, visto que, na próxima iteração, é possível que estas soluções que até o momento eram consideradas não-dominadas passem a ser dominadas por uma nova solução e não retornem ao arquivo na iteração seguinte. Todos estes passos são ilustrados na Figura 29.

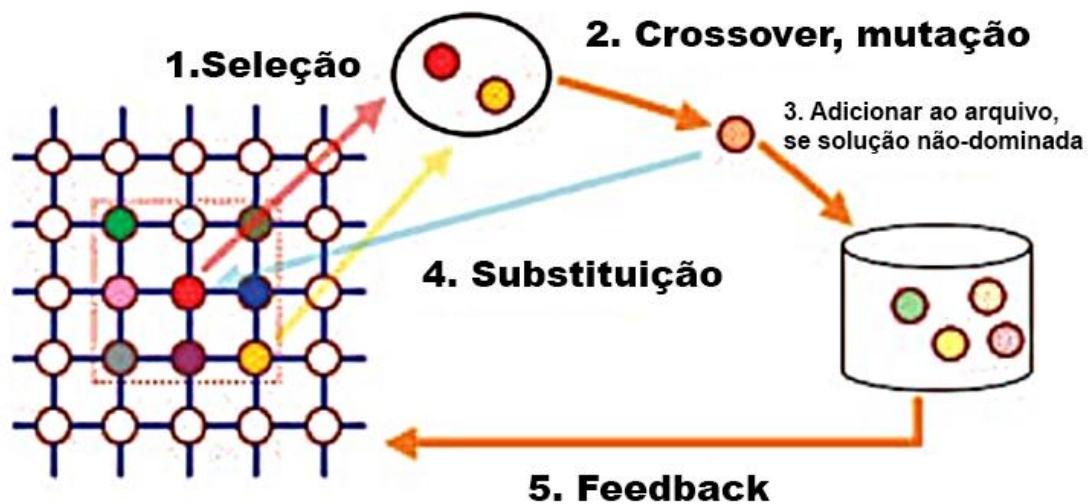


Figura 29 - Etapas de execução do MOCeL.

Segundo os resultados práticos apresentados pelos autores, apesar do MOCeL não ser capaz de atingir os mesmos valores de otimalidade alcançados pelo NSGA-II, o algoritmo é capaz de convergir mais rapidamente para um melhor espalhamento de soluções.

3.3. Considerações Finais

Neste capítulo, foram apresentados todos os fundamentos relacionados às metaheurísticas necessárias para resolver um problema de otimização, no caso particular deste trabalho, um problema do próximo release multi-objetivo. Iniciamos a discussão apresentando os algoritmos genéticos em comparação com algumas das principais metaheurísticas existentes na literatura. Na sequência, definimos os principais conceitos relativos aos algoritmos genéticos, sua relação com a teoria da evolução das espécies e como a solução é gerada inicialmente aleatoriamente, sendo sucessivamente aplicados os operadores de seleção, cruzamento, mutação e substituição. Operadores estes que foram apresentados em detalhes, destacando os principais tipos de abordagens utilizadas na implementação dos operadores genéticos, além dos possíveis critérios de parada que podem ser utilizados nos algoritmos genéticos. Em seguida, demonstramos como um algoritmo genético pode ser modificado para resolução de problemas multi-objetivos, destacamos alguns dos algoritmos mais utilizados na literatura, sendo o NSGA-II o mais bem referenciado e largamente utilizado para comparação de resultados, vimos como o NSGA-II realiza a categorização das soluções através da atribuição de dominância e classificação das soluções em diferentes *fronts*. O SPEA2 que, segundo seus autores, apresenta um desempenho similar ao NSGA-II e é definido por seu cálculo de atribuição de *fitness*, responsável por estabelecer a relação de dominância entre soluções, assim

definindo as soluções de melhor qualidade que foram encontradas pelo algoritmo. Finalmente o MOCcell é detalhado, que consiste um algoritmo de busca celular que incorpora boas práticas das abordagens de busca local durante o processo de seleção, em que oito das soluções mais próximas são definidas como vizinhos potencialmente selecionáveis como pares para cruzamento.

4. ABORDAGEM PROPOSTA

Conforme observado no segundo capítulo deste trabalho, o esquema clássico para o problema do próximo release adota uma abordagem mono-objetiva. No entanto, também foi observado que esta problemática evoluiu em relação à quantidade de objetivos a serem otimizados, tornando-se portanto uma abordagem multi-objetiva. Adicionalmente, o problema do próximo release também evoluiu para o problema de planejamento de releases, no qual os requisitos candidatos são selecionados para uma quantidade indefinida de iterações de desenvolvimento do produto de software. No contexto específico deste trabalho, também foi possível delinear 11 estudos que possuem alguma relação à temática desta proposta, no âmbito de medidas de risco ou incertezas a serem incorporadas ao problema do próximo release.

Posteriormente, também necessitamos designar alguma metaheurística capaz de solucionar o problema proposto. Para isto, descrevemos os algoritmos genéticos e entramos em detalhes a respeito da sua estratégia de busca global, assim como examinamos os operadores genéticos associados a esta técnica. Vimos também que os algoritmos genéticos possuem uma fácil adaptabilidade para lidar com problemas que admitem múltiplos objetivos, e, nesta direção, foram apresentadas três técnicas candidatas a serem utilizadas nesta proposta: NSGA-II, SPEA2 e por fim, MOCell.

O estudo destas metaheurísticas capazes de lidar com múltiplos objetivos ainda não é suficiente para lidar com o problema, portanto, necessitamos de uma implementação destas metaheurísticas capazes de atender aos interesses desta proposta. Para isto, apresentamos o jMetal como um framework capaz de executar estas metaheurísticas, codificar o problema do próximo release em um representação de string binária e apresentar resultados com base em medidas de desempenho pertinentes como o hipervolume e espalhamento da solução obtida com base na população de cromossomos. Estas informações complementares se encontram disponíveis no Anexo A no final deste documentos.

Enfim, fundamentado pelo embasamento conceitual relacionado ao problema do próximo release, incluindo a análise dos trabalhos relacionados à proposta aqui apresentada, e, em complemento, apoiado por implementações das metaheurísticas capazes de lidar com este problema multi-objetivo, podemos então apresentar e detalhar a abordagem proposta para o problema do próximo release, denominada *SR²* (*Selection of Requirements based on Software Risks*), que incorpora uma análise de riscos para estimar o impacto dos riscos no custo dos requisitos e na satisfação dos clientes.

4.1. Caracterização do Problema

Inicialmente, o cenário para o problema do próximo release deve ser caracterizado, tornando possível a criação de um modelo computacional capaz de gerar resultados quantificáveis para que possamos ser capazes de avaliar sua qualidade. Conforme observamos no Capítulo 2, propostas SBSE mais atuais para o problema do próximo release adotam a versão multi-objetivo deste problema, portanto, o modelo proposto também deve seguir este padrão. Deste modo, no contexto de NRP, o trabalho propõe uma abordagem multi-objetiva, baseada em riscos e automatizada para seleção de requisitos de software, em que a análise de risco é incorporada na estimação do impacto dos riscos de software no custo total de desenvolvimento e na satisfação dos stakeholders. Conforme ilustrado na Figura 30, a proposta adota uma arquitetura em camadas estrutura em três fases.

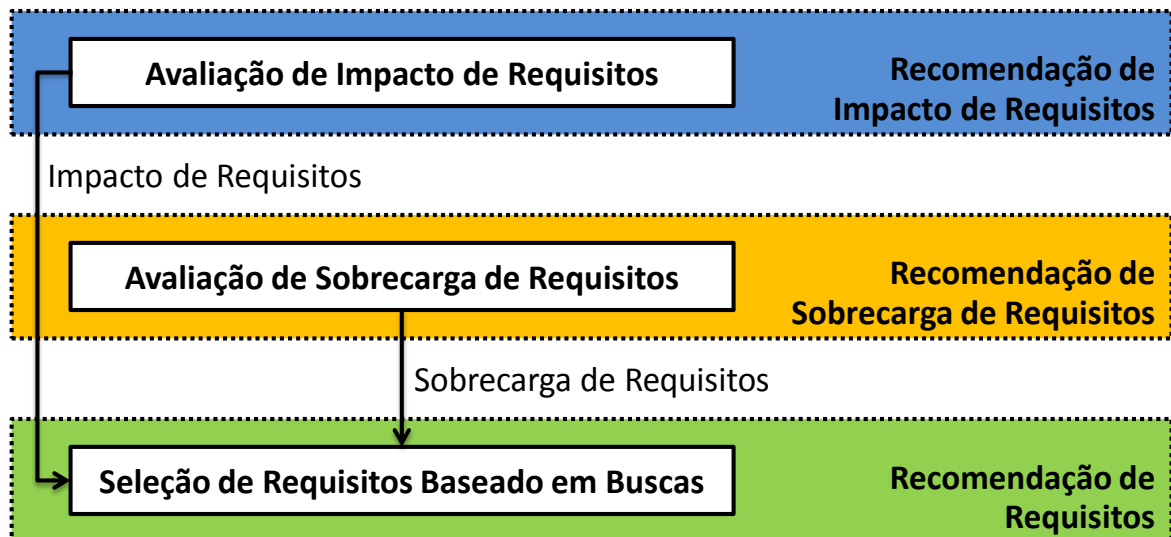


Figura 30 - Arquitetura de três camadas da abordagem proposta.

A primeira fase, denominada de **Recomendação de Impacto de Requisitos**, avalia os requisitos de software conjuntamente com seus riscos de software associados, produzindo como uma avaliação de impacto de requisitos, estabelecendo uma penalidade a ser aplicada para cada requisito de software ao estimar a satisfação de clientes. A principal saída desta fase consiste em produzir um vetor que representa o impacto de requisitos, extraído da informação relativa à rastreabilidade de riscos.

A segunda fase, **Recomendação de Sobrecarga de Requisitos**, é responsável por avaliar os requisitos de software e suas técnicas de mitigação associadas de modo a estimar a sobrecarga de requisitos, que representam um tipo de penalidade a ser aplicada para cada

requisito de software devido a adoção de suas técnicas de mitigação associadas. Como saída principal desta segunda etapa, uma matriz tridimensional que representa a sobrecarga de requisitos é obtida a partir da informação da rastreabilidade de riscos e técnicas de mitigação.

Recebendo as informações relativas a impacto e sobrecarga de requisitos como entrada, a terceira fase, denominada **Recomendação de Requisitos**, adota três algoritmos de otimização multiobjetivo, sendo eles o NSGA-II (Seção 3.2.1), SPEA2 (Seção 3.2.2) e MOCell (Seção 3.2.3) já apresentados em detalhe no Capítulo 3. Estes algoritmos são responsáveis por selecionar um subconjunto dentre os requisitos candidatos existentes em que, no contexto de NRP, os riscos de software podem introduzir penalidades ao resultado final, impondo um impacto na satisfação de clientes e introduzindo um custo final adicional ao custo de implementação destes requisitos. É importante destacar que esta terceira fase, ao produzir este subconjunto de requisitos candidatos selecionados, podem ser adotados pelo gerente de projeto para seleção de requisitos de software a serem entregues no próximo release, possivelmente levando em consideração quaisquer outras restrições de projeto ou organizacional, tais como o tamanho da equipe de desenvolvimento, assim como suas habilidade técnicas e a disponibilidade dos membros da equipe, impondo um limite na quantidade de requisitos que podem ser entregues dentro destas restrições.

As seções seguintes apresentam os detalhes de cada etapa desta arquitetura em três camadas: como estimar o impacto dos requisitos; como estimar a sobrecarga de requisitos; e como formular o problema do próximo release baseado em riscos de software.

4.1.1. Recomendação de Impacto de Requisitos

De forma a identificar o impacto de requisitos, é assumido que os riscos de software associados a estes requisitos candidatos implicando em um impacto no processo de implementação destes requisitos, que possuem o potencial de afetar a qualidade do software sendo desenvolvido. Portanto, a primeira fase recebe como entrada as informações relativas aos riscos de software, explorando a rastreabilidade de riscos associados aos requisitos candidatos, assim como também a severidade responsável por definir o impacto da ocorrência deste risco.

Na abordagem proposta, os **requisitos de software**, que são levantados junto aos stakeholders durante o processo de engenharia de requisitos, são representados na Equação 13. Observamos nesta Equação que cada rq_i representa um requisito candidato para o próximo release.

$$RQ = \{rq_1, rq_2, \dots, rq_n\} \quad (13)$$

Por sua vez, os **riscos de software**, que são identificados pelo gerente de projeto durante o processo de gerenciamento de riscos, são representados na Equação 14, em que cada termo rk_j representa um risco identificado no projeto e que está associado a algum requisito candidato.

$$RK = \{rk_1, rk_2, \dots, rk_m\} \quad (14)$$

A informação a respeito de riscos é necessário pois, diversos fatores de risco associados ao processo de engenharia de requisitos podem emergir durante o planejamento do próximo release do sistema. Estes fatores podem representar o fracasso ou sucesso de um novo release de acordo com a visão dos clientes e usuários finais do sistema (BOEHM, 1991) (ISLAM e HOUMB, 2010).

Nesta direção, a abordagem proposta lida com os *riscos de software* associados com os requisitos do software. Dentre os exemplos de riscos de software, podem ser indicadas as mudanças inevitáveis em projetos de software de grande porte devido à constante evolução do modelo de negócio dos clientes. Além disso, como um exemplo adicional, a falta de compreensão a respeito da definição dos requisitos, incluindo a incorreta interpretação dos reais interesses dos clientes, ou até mesmo a omissão de recursos importantes ou a falta de restrições associadas aos requisitos. Esta falta de compreensão ou incompletude pode estar relacionada a um contexto específico, em que a linguagem utilizada para definir os requisitos pode possuir um significado especial para o domínio de negócio do cliente. Ademais, o produto de software pode passar por um crescimento descontrolado devido a problemas pertinentes ao escopo e fronteiras do sistema. Isto pode ser o resultado de descrições vagas ou abstratas dos requisitos, algumas vezes definidos de maneira deliberada, de modo a conciliar as visões conflitantes dos diversos stakeholders envolvidos (RAJAGOPAL *et al.*, 2005) (ARNUPHAPTRAIRONG, 2011) (DEMARCO e LISTER, 2003).

A elicitação de requisitos não deve ser realizada de maneira muito ambiciosa, mas deve ser feita de modo realista e alcançável. Além disso, uma falta de rastreabilidade durante a fase de validação de requisitos também pode se tornar um fator de risco. Do ponto de vista dos usuários finais, eles podem rejeitar o produto de software caso não sejam apropriadamente envolvidos durante o processo de desenvolvimento de software, gerando uma resistência a mudanças, especialmente quando o produto transmite uma percepção de imposição aos

mecanismos utilizados em substituição ao novo software a ser usado. De fato, vários outros fatores de risco mais específicos podem ser considerados. Tais riscos podem gerar uma série de eventos indesejáveis que podem impactar no grau de satisfação percebido pelos clientes, além de aumentar o custo final do próximo release a ser entregue. Todos estes fatores podem estar relacionados a uma seleção inadequada dos requisitos candidatos (HUMA HAYAT KHAN e NAZ'RI BIN MAHRIN, 2014).

Conseqüentemente, levando em consideração o contexto mutável relacionado aos processos de desenvolvimento de software, numerosos recursos e restrições relacionados ao projeto de software, produto de software, ambiente de desenvolvimento, hardware de implantação e plataformas de software (DEMARCO e LISTER, 2003), possuem o potencial de afetar os requisitos candidatos de maneira direta ou indireta (ISLAM e HOUMB, 2010).

Usualmente, um risco, em uma abordagem pessimista, é definido como uma perda financeira ou material, ou qualquer outro evento que deve ser evitado em um processo de gestão de riscos (BOEHM, 1991), (YANG, JONES e YANG, 2006) (FEATHER e CORNFORD, 2003). Cada risco possui uma *severidade* associada, que sinaliza as conseqüências da ocorrência de um dado risco e seu impacto ou perda associados (YANG, JONES e YANG, 2006) (FEATHER e CORNFORD, 2003). Ademais, cada risco possui uma *probabilidade* associada, que define o quão provável é deste evento indesejável ocorrer.

Na abordagem proposta, os **riscos** são definidos pelo conjunto ' RK ', conforme indicado na Equação 14. Conforme mencionado, cada risco rk_j é associado a um valor de probabilidade e um valor de severidade. Entretanto, dada a dificuldade de estimar valores precisos aos graus de probabilidade e severidade de um risco, na abordagem proposta, uma representação orientada por um modelo *fuzzy* é utilizada, em que valores textuais são empregados de modo a classificar a probabilidade e severidade como valores discretos conforme representados na Tabela 3.

Tabela 3 - Valores de probabilidade e severidade de risco.

Classe	Muito Baixo	Baixo	Médio	Alto	Muito Alto
Valor	0.05	0.25	0.50	0.75	0.95

Conforme podemos notar na Tabela 3, uma escala Likert de cinco níveis foi adotada por se tratar de uma representação comum de riscos utilizada na literatura (BANNERMAN, 2008). Deste modo, os valores atribuídos à probabilidade e severidade de risco apenas podem assumir valores definidos no conjunto FT , definido na Equação 15.

$$FT = \{0.05, 0.25, 0.50, 0.75, 0.95\} \quad (15)$$

Finalmente, a **probabilidade de risco** e a **severidade de risco** são definidos respectivamente através dos conjuntos RKP e RKS , conforme demonstrado nas Equações 16 e 17. Note que os termos correspondentes rkp_j e rks_j apenas podem assumir valores representados pelos termos *fuzzy* definidos na Tabela 3 e formalizado pelo conjunto FT na Equação 15.

$$RKP = \{rkp_j | \exists rk_j \in RK \wedge rkp_j \in FT\} \quad (16)$$

$$RKS = \{rks_j | \exists rk_j \in RK \wedge rks_j \in FT\} \quad (17)$$

Para representação da rastreabilidade entre requisitos e riscos de software, é definido na abordagem aqui proposta com **rastreabilidade de riscos**, onde cada requisito rq_i pode estar associado com um ou mais riscos rk_j . Deste modo, conforme indicado na Equação 18, o relacionamento $RT_{RQ,RK}$ é utilizado para representar esta rastreabilidade, de forma que o termo $rt_{i,j}$ denota esta rastreabilidade entre o requisito rq_i e o risco rk_j , assumindo valor igual a um ou zero para sinalizar a existência ou não deste relacionamento.

$$RT_{RQ,RK} = \{rt_{i,j} | \exists rq_i \in RQ, \exists rk_j \in RK, rt_{i,j} \in \{0,1\}\} \quad (18)$$

Com esta equação, conseguimos observar não apenas que um requisito pode estar associado a um ou mais riscos, mas o inverso também é verdadeiro, um único risco pode estar atrelado a uma série de requisitos distintos. Portanto, a ocorrência de um evento de risco podem afetar uma série de funcionalidades do sistema.

Este conceito de rastreabilidade de riscos pode ser observado em propostas tais como a de (CLELAND-HUANG *et al.*, 2005). Seu modelo de “Rastreabilidade Centrada em Metas” parte de uma abordagem ampla, em que são avaliados possíveis mudanças em requisitos não-funcionais definidos de “*softgoals*”. A partir de cada requisitos não-funcional, uma série de funcionalidades do sistema são afetadas. Estas funcionalidades são mapeadas em “metas” que o sistema deve cumprir. Em seguida, estas metas são decompostas em “operacionalizações”. Estas operacionalizações detalham as etapas específicas que o sistema de software deve efetuar para cumprir as metas propostas para o sistema, sendo então mais próximos da definição de requisitos funcionais para o sistema, que são rastreadas retroativamente para os requisitos não-funcionais a partir destas etapas intermediárias. Estas operacionalizações, por sua vez, são

definidas através de um grafo de interdependência de *softgoals*. Com auxílio de uma ferramenta de modelagem UML aponta possíveis interdependências entre estes modelos funcionais e não-funcionais do sistema, daí então o tomador de decisão aceita ou rejeita cada uma das interdependências entre estas operacionalizações. Os autores informam que este procedimento é análogo à realização de uma busca na internet, em que o buscador deve classificar os links retornados como relevantes ou irrelevantes aos termos de busca utilizados. Para o caso em que uma mudança é proposta ao sistema. Uma etapa de reavaliação de metas é empregada de modo a avaliar como cada aspecto do modelo funcional do sistema deve ser modificado de forma a satisfazer os aspectos não-funcionais do sistema. Esta reavaliação fornece como saída um relatório de análise de impacto identificando como todas estas metas foram afetadas positivamente ou negativamente pela mudança proposta. Finalmente, caso a mudança seja aceita e deva ser implementada em uma nova versão do sistema, documentos de design são modificados para incorporar esta mudança e planos de mitigação de riscos são identificados de modo a minimizar ou remover quaisquer possíveis impactos negativos ao sistema em razão desta mudança. Os autores validaram a proposta utilizando artefatos de um sistema crítico meteorológico responsável por detectar o congelamento de estradas com base em sensores instalados nas estradas e a partir de dados recebidos por estações de monitoramento de clima locais. Possíveis mudanças foram propostas e com base nestes artefatos, as mudanças foram então aceitas ou rejeitadas. Dentre as mudanças aceitas, planos de mitigação de risco foram propostos. Por exemplo, no caso em que uma estrada muito ampla exigisse uma grande quantidade de sensores, estes sensores causariam um grande impacto na unidade de processamento responsável por enviar estes dados, sendo que possíveis atrasos no envio desta informação poderiam prejudicar a segurança dos motoristas que trafegam pela região. Deste modo, duas possíveis ações de mitigação deste impacto negativo com a adição de novos sensores ao sistema poderiam ser resolvidos: utilizar os dados das estações climáticas mais próximas ou acrescentar novas unidades de processamento de modo a aliviar a carga de dados recebida. Além disso, a inclusão de novos sensores e novas unidades de processamento pode ser uma ação desejável em garantir uma redundância no sistema, uma característica desejável em um sistema crítico responsável pela segurança de motoristas.

O modelo observado em (SÁNCHEZ *et al.*, 2011) também realiza um procedimento de rastreabilidade de requisitos de segurança, sendo identificados potenciais riscos do sistema e as possíveis formas de mitigação destes riscos. Esta proposta utiliza um processo de desenvolvimento orientado a modelos, em que os requisitos de segurança são mapeados em

design, sendo o design por sua vez correspondente à implementação do sistema. Sendo esta proposta realizada para o desenvolvimento de equipamentos robóticos responsáveis por reparar e pintar o casco de navios. Portanto, se trata de um sistema crítico, pois, uma ação incorreta do robô pode resultar em danos para o navio ou em ferimentos para qualquer profissional que esteja próximo do equipamento robótico, os autores argumentam que o uso extensivo de modelos de design rastreáveis é uma abordagem adequada para este sistema. Para cada requisito de segurança, um meta-modelo em que são mapeados informações relativas a potenciais perigos envolvidos a este requisito são identificados, cada perigo, por sua vez, é associado a um grau de risco, sua severidade, sua categoria de redução de risco, responsável por identificar quaisquer salvaguardas e circuitos que são responsáveis por mitigar este risco, além de possíveis políticas de segurança que devem ser adotadas para evitar este risco. É importante notar que esta categoria de redução de risco segue um padrão ANSI/RIA de automação robótica industrial. Portanto, novamente a importância do uso de rastreabilidade de requisitos em relação aos seus riscos identificados e os riscos por sua vez, são atrelados a alguma metodologia de redução ou eliminação do risco se mostra relevante no uso de sistemas críticos.

Após a definição de todos estes conceitos e, levando em consideração a medida baseada em severidade de riscos de software associados ao requisito rq_i , o **impacto de requisitos** pode ser definido na Equação 19.

$$IMP = \{imp_i | \exists rq_i \in RQ, imp_i \in \mathbb{R}\} \quad (19)$$

Cada termo individual imp_i pode ser estimado na Equação 20. Representando o impacto de todos os riscos associados ao requisito rq_i . Esta fórmula estabelece a relação entre os seguintes termos: (i) a rastreabilidade $rt_{i,j}$ entre o requisito rq_i e seus riscos associados rk_j ; e (ii) a severidade de cada risco (rks_j) associada ao risco rk_j . De fato, o termo imp_i representa a soma da severidade de todos os riscos rk_j associados ao requisito rq_i .

$$imp_i = \sum_{rk_j \in RK} rks_j \cdot rt_{i,j} \quad (20)$$

4.1.2. Recomendação de Sobrecarga de Requisitos

Como uma maneira de estimar a sobrecarga de requisitos, é assumido que técnicas de mitigação sejam empregadas para lidar com os riscos. A utilização destas técnicas implicam em uma sobrecarga na implementação dos requisitos, que possui o potencial de afetar o custo do software a ser desenvolvido. Portanto, a segunda fase recebe como entrada as informações

relevantes sobre os requisitos de software, riscos de software e técnicas de mitigação de riscos, explorando a rastreabilidade de riscos em relação aos requisitos, assim como a rastreabilidade de técnicas em relação aos riscos, provendo um meio de monitorar e estimar a possível sobrecarga sobre os requisitos.

Em um processo de gerenciamento de riscos, caso ocorra um evento de risco, devem ser adotadas **técnicas de mitigação de riscos** (FEATHER e CORNFORD, 2003), afim de reduzir ou eliminar as conseqüências de um risco. Na abordagem proposta, estas técnicas mitigatórias são definidas pelo conjunto T na Equação 21.

$$T = \{t_1, t_2, \dots, t_q\} \quad (21)$$

Cada técnica de mitigação de riscos possui um custo associado com sua adoção, definida pelo conjunto TC e seus respectivos termos tc_k na Equação 22, definindo assim o **custo de técnica**. É importante enfatizar que, com base em algum método de estimação de custos abordado durante a fase de planejamento do projeto, a equipe de desenvolvimento deve ser capaz de providenciar as estimativas de custo de técnica tc_k para cada técnica de mitigação de riscos correspondente t_k .

$$TC = \{tc_k | \exists t_k \in T \wedge tc_k > 0\} \quad (22)$$

Durante o processo de desenvolvimento, cada técnica de mitigação pode auxiliar na resolução de um ou mais riscos. De maneira a representar a rastreabilidade entre riscos e suas técnicas de mitigação associadas, também necessitamos definir o relacionamento $TT_{RK,T}$, denominado **rastreabilidade de técnicas** e representado na Equação 23. Este relacionamento define cada termo $tt_{j,k}$ que associa o risco rk_j à técnica de mitigação t_k , assumindo valores iguais a um ou zero para indicar a existência ou inexistência deste relacionamento, respectivamente.

$$TT_{RK,T} = \{tt_{j,k} | \exists rk_j \in RK \wedge \exists t_k \in T \wedge tt_{j,k} \in \{0,1\}\} \quad (23)$$

Novamente, de maneira análoga ao conceito de rastreabilidade de riscos, do mesmo modo que um risco deve ser resolvido por uma série de técnicas de mitigação, o oposto também pode ser observado: uma única técnica pode ser capaz de resolver uma série de eventos de risco identificados no projeto. Está técnica polivalente é um efeito desejável visto que, para uma única ação mapeada em uma técnica, ela aumenta consideravelmente as chances de sucesso do conjunto de requisitos candidatos afetados por estes riscos. Na direção contrária, uma técnica

que age apenas sobre um único risco e este risco, por sua vez, está associado a um único requisito representa um esforço adicional. Visto que uma única ação mapeada em uma técnica de mitigação de risco aumenta as chances de sucesso de uma parcela restrita do subconjunto de requisitos selecionados para o próximo release, portanto representando um menor impacto no sucesso do incremento como um todo.

Geralmente, em processos de gerenciamento de riscos, as técnicas de mitigação podem ser classificadas em dois tipos:

- **Preventiva:** esta técnica busca evitar a ocorrência do risco. É aplicada durante o processo de implementação do release, independentemente da ocorrência ou não de um evento associado ao risco;
- **Corretiva:** em contrapartida, este tipo de técnica mitigatória é utilizada em busca de diminuir ou eliminar a consequência de um risco. É aplicada após a ocorrência do evento de risco, durante um estado de emergência, de modo a permitir que o projeto de software retorne ao seu estado desejado, eliminando ou reduzindo as consequências associadas a um ou mais riscos.

Finalmente, com a definição de todos os conceitos apresentados até o momento, é possível definir a **sobrecarga de requisitos** do modelo através do relacionamento definido na Equação 24.

$$OVR_{RQ,RK,T} = \{ovr_{i,j,k} | \exists rq_i \in RQ \wedge \exists rk_j \in RK \wedge \exists t_k \in T \wedge ovr_{i,j,k} \in \mathbb{R}\} \quad (24)$$

Cada termo individual $ovr_{i,j,k}$ pode ser estimado na Equação 25, que representa a sobrecarga de aplicar uma dada técnica t_k , relacionado ao risco rk_j , que por sua vez é associado ao requisito rq_i . Note que, conforme definido na Equação 25, se existe o relacionamento $(rt_{i,j})$ entre o requisito rq_i e o risco rk_j , e também se existir o relacionamento $(tt_{j,k})$ entre o risco rk_j e a técnica t_k , a sobrecarga $ovr_{i,j,k}$ é definida baseada no custo de aplicar a técnica t_k ao risco rk_j .

$$ovr_{i,j,k} = rt_{i,j} \cdot tt_{j,k} \cdot tc_k \cdot tp_{j,k} \quad (25)$$

Por sua vez, o termo $tp_{j,k}$ dado pela Equação 26, diferencia o tipo da técnica a ser aplicada. O custo de técnicas preventivas são considerados integralmente, já que elas sempre serão aplicadas independentemente da ocorrência do evento de risco ou não. Em contraste, o

custo das técnicas corretivas são dependentes da probabilidade de ocorrência do risco (rkp_j), visto que elas não são aplicadas até que o evento de risco ocorra, o que ainda é dado como um evento incerto durante o processo de desenvolvimento do release. Deste modo, o custo de técnicas corretivas são dadas com base em uma estimativa.

$$tp_{j,k} = \begin{cases} 1, & \text{se técnica } \textbf{preventiva} \\ rkp_j, & \text{se técnica } \textbf{corretiva} \end{cases} \quad (26)$$

Note que a severidade e a probabilidade de risco são consideradas separadamente na abordagem proposta ao afetar o impacto e a sobrecarga de requisitos, implicando que os eventos de risco não causam necessariamente a consequência esperada, pois estas consequências negativas podem ser reduzidas ou eliminadas com o uso de técnicas de mitigação de risco (ZHANG, 2007).

4.1.3. Recomendação de Requisitos

Com base no impacto e sobrecarga de requisitos, estimados previamente nas primeira e segunda etapas do modelo, a terceira fase da proposta incorpora conceitos e técnicas de SBSE para representar o problema do próximo release como uma problema de otimização multi-objetivo de forma a recomendar uma boa solução ou, caso seja encontrada, até mesmo uma solução ótima para o problema.

Conforme observamos na Seção 2.2 a resolução de um problema do próximo release multiobjetivo não consiste apenas a melhor aproximação possível da solução ótima, mas um conjunto de soluções de formam uma Fronteira de Pareto. Esta fronteira de Pareto pode ser facilmente encontrada para problema pequenos, em que um espaço de busca limitado pode ser investigado completamente por um algoritmo de busca exaustiva adaptado por testar a dominância de soluções, portanto fornecendo ao final de sua execução o conjunto de soluções não-dominadas em relação a todo o universo de soluções existentes no espaço de busca. Porém, dada a crescente complexidade computacional para problemas de com uma quantidade elevada de requisitos candidatos, a utilização de um algoritmo de busca exaustivo não é possível de ser realizada em um tempo hábil. Portanto, o uso de alguma metaheurística capaz de obter uma boa aproximação desta Fronteira de Pareto que contém um conjunto de soluções ótimas passa a ser uma meta desejável para esta terceira etapa do modelo. Para isto, os algoritmo NSGA-II, SPEA2 e MOCcell previamente apresentados em detalhe na Seção 3.2 foram empregados e, conjuntamente com a definição de um problema multi-objetivo para o próximo release com duas funções de avaliação de custo e satisfação, a abordagem proposta foi modelagem com base

nesta esquematização biobjetiva, conforme podemos observar na Figura 31. Esta figura ilustra os principais passos envolvidos em uma abordagem baseada em riscos.

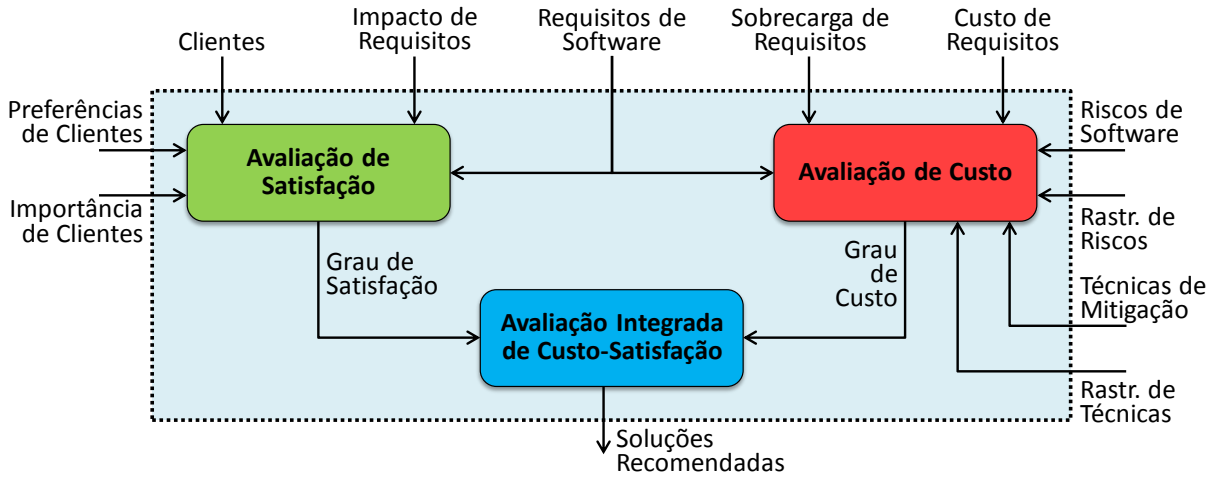


Figura 31 - Fase de recomendação de requisitos.

Conforme observado na Figura 31, a terceira etapa introduz um novo conjunto de informações a respeito de clientes, assim como também incorpora as previamente definidas informações a respeito de requisitos de software, riscos de software e técnicas de mitigação. Portanto, antes de detalhes os procedimentos relativo a estimação dos graus de satisfação de clientes e de custo de requisitos, é importante definirmos as novas informações necessárias para esta etapa.

Na abordagem proposta, os **clientes** são representados pelo conjunto ‘ U ’, conforme sinalizado na Equação 27.

$$U = \{u_1, u_2, \dots, u_p\} \quad (27)$$

Cada cliente u_l deve definir uma lista de **preferências de requisitos**, atribuindo uma preferência maior aos requisitos de maior importância para si. Por conseguinte, para cada cliente u_l , deve ser definido um grau de preferência atribuído para cada requisito rq_i . Conforme observado na Equação 28, tal preferência é representada pelo relacionamento $S_{U,RQ}$, composto por termos denominados $s_{l,i}$, que assume valores dentro do intervalo $[0, 1] \in \mathbb{R}$. Note que o valor para $s_{l,i}$ igual a um indica que o cliente u_l possui o máximo grau de interesse no requisito rq_i , enquanto que o valor zero denota uma completa falta de interesse neste requisito.

$$S_{U,RQ} = \{s_{l,i} | \exists u_l \in U \wedge \exists rq_i \in RQ \wedge s_{l,i} \in [0,1]\} \quad (28)$$

Conseqüentemente, os clientes podem ter interesses distintos ou até mesmo conflitantes a respeito dos requisitos a serem selecionados no próximo release. Para lidar com estes conflitos de interesse, a organização responsável pelo desenvolvimento do produto de software deve adotar uma estratégia conciliatória para balancear estas preferências conflitantes. Na abordagem proposta, tal estratégia é baseada no conceito de **importância dos clientes** do ponto de vista da organização desenvolvedora, indicando o quão importante cada cliente u_l atende suas estratégias de negócio. A importância dos clientes é representada pelo conjunto E , conforme demonstrado na Equação 29, em que cada termo e_l pode assumir valores dentro do intervalo $(0,1] \in \mathbb{R}$, sinalizando a importância associada a respeito de cada cliente u_l . Diferentemente das preferências de requisitos dos clientes, a importância dos clientes não pode ser completamente descartada e deve assumir um valor diferente de zero de modo a refletir este comportamento.

$$E = \{e_l | \exists u_l \in U \wedge e_l \in (0,1]\} \quad (29)$$

Para o próximo release, os clientes possuem interesse em um conjunto de requisitos candidatos, representados pelo conjunto RQ apresentado na Seção 4.1.1. Além disso, conforme discutido nas Seções 4.1.1 e 4.1.2, um dado requisito de software rq_i também está sujeito ao impacto e sobrecarga devido aos seus riscos de software e técnicas de mitigação associados, definidos respectivamente como **impacto de requisitos** e **sobrecarga de requisitos**, dados pelo conjunto IMP e pelo relacionamento $OVR_{RQ,RK,T}$ correspondentes.

Além disso, de modo a avaliar o custo do próximo release, é necessário definir os **custos de requisitos** associados a cada requisito rq_i , que é representado pelo conjunto RQC na Equação 30. É importante destacar que a equipe de desenvolvimento, com base em algum método de estimação de custos adotado durante a fase de planejamento de projeto, irá providenciar estes valores de custos de requisitos.

$$RQC = \{rqc_i | \exists rq_i \in RQ \wedge rqc_i > 0\} \quad (30)$$

Conforme também previamente discutido na Seção 4.1.1 e observado nos trabalhos relacionados (BOEHM, 1991) (ISLAM e HOUMB, 2010), o custo do requisito não pode ser utilizado como o único fator a ser avaliado em um problema, visto que outros fatores-chave associados a riscos identificados durante o processo de engenharia de requisitos podem emergir durante o planejamento do próximo release. Portanto, a incorporação de análise de risco é considerado um fator importante na entrega do próximo release na abordagem proposta.

Conseqüentemente, conforme também apresentados nas Seções 4.1.1 e 4.1.2, os **riscos de software** e suas **técnicas de mitigação**, representadas respectivamente pelos conjuntos RK e T são incorporados ao modelo. Além disso, como uma forma de rastrear os relacionamentos entre requisitos e riscos e, por sua vez, entre riscos e suas técnicas de mitigação, a abordagem proposta também incorpora os conceitos de **rastreabilidade de riscos** e **rastreabilidade de técnicas**, também discutidos nas Seções 4.1.1 e 4.1.2. É importante lembrarmos que estes elementos de rastreabilidade de riscos e rastreabilidade de técnicas são denotados por $RT_{RQ,RK}$ e $TT_{RK,T}$, respectivamente.

Finalmente, também sabemos que, dentre estes requisitos candidatos, apenas um subconjunto será de fato selecionado para o próximo release. Assim, de modo a comunicar a seleção de cada requisito rq_i , cada possível **solução recomendada** é representada pelo conjunto ' X '. Conforme definido na Equação 31, cada termo x_i , assume valores iguais a '1' ou '0' para denotar que cada requisito rq_i foi selecionado ou não, respectivamente.

$$X = \{x_i | \exists rq_i \in RQ \wedge x_i \in \{0,1\}\} \quad (31)$$

4.1.3.1. Avaliação de Satisfação

Após a definição de todos os conceitos necessários, o **grau de satisfação** é estimado pela **função de satisfação**, definida na Equação 32. Ela indica a satisfação percebida pelos clientes levando em consideração o subconjunto de requisitos selecionados. Note que o nível de satisfação é modelado pela soma total para cada requisito r_i e cada cliente u_l , considerando o produto dado pelos seguintes termos: (i) o grau de preferência $s_{l,i}$ que cada cliente u_l possui em relação ao requisito rq_i ; (ii) o grau de importância e_l que a organização responsável pelo desenvolvimento do software atribui para cada cliente u_l ; (iii) os valores de severidade de risco dado pela função imp_i , descrita posteriormente, associada ao requisito rq_i ; e (iv) o seletor x_i que representa a seleção ou não do requisito rq_i na solução avaliada por alguma metaheurística.

$$\text{maximizar } S_X = \sum_{u_l \in U} \sum_{rq_i \in RQ} s_{l,i} \cdot e_l \cdot imp_i \cdot x_i \quad (32)$$

Conforme também pode ser visto na Equação 32, observando o grau de satisfação, o problema enfrentado pelas organizações de desenvolvimento consiste em encontrar um subconjunto de requisitos que maximize a satisfação dos clientes. Portanto, quanto maior for o grau de satisfação S_X para uma dada seleção de requisitos X , conforme esta seleção for

selecionando os melhores requisitos candidatos, a solução candidata então se aproximará do conjunto ótimo de Pareto (P^*).

Note que a abordagem proposta adota a premissa que os processos de software devem lidar primeiramente com os riscos mais críticos de formar a maximizar as chances de sucesso do projeto, em decorrência de melhorar os indicadores de satisfação. Nesta direção, a abordagem proposta favorece a seleção de requisitos associados aos riscos mais severos, produzindo um efeito proporcionalmente menor caso riscos de menor relevância sejam selecionados, decorrendo em uma penalidade responsável por diminuir a satisfação. De fato, usualmente, os processos de gestão de riscos focam em riscos críticos, mitigando ou evitando a ocorrência de eventos indesejáveis que possam comprometer a satisfação dos clientes (ALAM, 2014).

4.1.3.2. Avaliação de Custo

Agora, o **grau de custo** é estimado pela **função de custo**, dada na Equação 33, que define o custo total percebido pela equipe de desenvolvimento ao implementar uma solução candidata X , que inclui um subconjunto de requisitos candidatos que foram selecionados. Como pode ser observado, o grau de custo é definido pela soma total entre o custo dos requisitos candidatos (rqc_i) conjuntamente com o custo adicional de gerenciamento de riscos ($rkci$) para cada requisito rq_i . Conforme definido, o custo de gerenciamento de riscos representa uma penalidade no custo total para o próximo release. Em projetos de software reais, lidar com riscos como uma medida adicional de custo é uma estratégia habitual (BANNERMAN, 2008). Conforme podemos notar, os custos de desenvolvimento e gerenciamento de riscos associados a um dado requisito rq_i apenas são considerados se o requisito for selecionado para o próximo release, que é representado na Equação 33 pelo termo x_i .

$$\text{minimizar } C_X = \sum_{rq_i \in RQ} (rqc_i + rkci) \cdot x_i \quad (33)$$

Neste caso, observando o grau de custo, o problema enfrentado pelos desenvolvedores consiste em encontrar um subconjunto de requisitos que minimizar o custo total de requisitos. Portanto, conforme o valor do grau de custo C_X para um dado conjunto de requisitos X diminui, isto significa que a solução está melhorando e se aproxima da solução candidata contida no conjunto ótimo de Pareto (P^*).

O custo de gerenciamento de riscos $rk c_i$ para um dado requisito rq_i pode ser definido na Equação 34. Conforme podemos observar, esta função é dada pela soma total do custo de gerenciamento de risco $rk c_{i,j}$ relativo a cada risco individual rk_j associado ao requisito rq_i .

$$rk c_i = \sum_{rk_j \in RK} rk c_{i,j} \quad (34)$$

É importante lembrar que uma dada técnica de mitigação de risco (t_k) pode estar associada a um ou mais riscos (rk_j). Deste modo, é razoável adotar a premissa que o custo tc_k de aplicar uma dada técnica t_k deve ser dividida entre todos os riscos rk_j que fazem uso desta mesma técnica. Além disso, também é assumido que um requisito rq_i pode estar associado a um ou mais riscos rk_j , que por sua vez podem estar associados a uma ou mais técnicas de mitigação t_k . Este relacionamento de custo entre requisitos, riscos e técnicas de mitigação é desmembrado no dado de sobrecarga de requisitos definido pelo relacionamento $OVR_{RQ,RK,T}$ previamente introduzido.

Agora, conforme definido na Equação 35, o custo de gerenciamento do risco $rk c_{i,j}$ para cada risco individual rk_j associado ao requisito rq_i é definido pela razão dos seguintes termos: (i) o total da soma da sobrecarga de requisito $ovr_{i,j,k}$ ao aplicar cada técnica t_k relacionada ao risco rk_j e ao requisito rq_i ; e (ii) o número de vezes, dado por ta_k , que uma dada técnica t_k é aplicada em todos os requisitos rq_i e riscos rk_j , definida posteriormente na Equação 36.

$$rk c_{i,j} = \sum_{t_k \in T} \frac{ovr_{i,j,k}}{ta_k} \quad (35)$$

Agora, retornando ao termo ta_k , definido na Equação 36, é responsável por denotar a quantidade de vezes que a técnica t_k é aplicada em todas as combinações entre requisitos e riscos ($rt_{i,j}$), assim como também entre os riscos e a técnica em questão ($tt_{j,k}$).

$$ta_k = \sum_{rq_i \in RQ} \sum_{rk_j \in RK} x_i \cdot rt_{i,j} \cdot tt_{j,k} \quad (36)$$

A partir deste ponto, o custo de gerenciamento de riscos $rk c_i$, foi completamente definido diluindo o custo de cada técnica t_k entre todos os riscos rk_j que fazem uso desta técnica. De modo a tornar esta estratégia mais clara, considere o seguinte exemplo ilustrado na Figura 32.

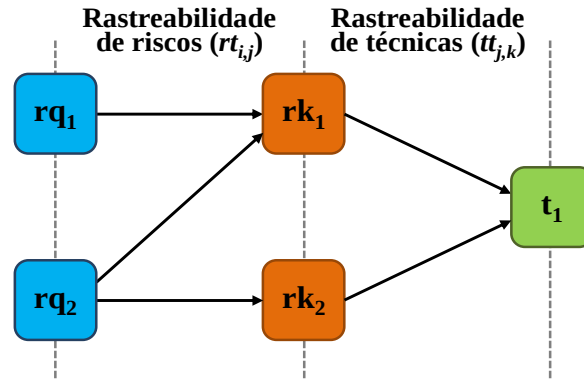


Figura 32 - Ilustração do exemplo de cálculo de riscos para dois requisitos.

Considere dois requisitos rq_1 e rq_2 . Considere também que o requisito rq_1 é associado ao risco rk_1 , enquanto que o requisito rq_2 é associado aos riscos rk_1 e rk_2 . Por conseguinte, considere que ambos os riscos são tratados por uma única técnica t_1 . Se ambos os requisitos são selecionados, o custo da técnica t_1 é particionada desproporcionalmente entre os requisitos, sendo um terço atribuído ao requisito rq_1 e dois terços alocados ao requisito rq_2 . Isto ocorre ao observarmos que existem três trilhas de relacionamento dos requisitos rq_1 e rq_2 em direção à técnica final t_1 , sendo um caminho iniciado no requisito rq_1 e dois caminhos partindo de rq_2 . Deste modo, o custo de gerenciamento de riscos rk_{c_i} para cada requisito rq_i é calculado nas formas dadas nas Equações 37 e 38. Podemos observar que $tp_{j,k}$ representa a probabilidade de aplicar a técnica t_k ao risco rk_j (Equação 26) e tc_k representa o custo de aplicar a técnica t_k .

$$rk_{c_1} = tp_{1,1} \cdot \left(\frac{tc_1}{3}\right) \quad (37)$$

$$rk_{c_2} = tp_{1,1} \cdot \left(\frac{tc_1}{3}\right) + tp_{2,1} \cdot \left(\frac{tc_1}{3}\right) \quad (38)$$

4.1.3.3. Avaliação Integrada de Custo-Satisfação

Em conclusão, assim que o grau de satisfação S_X e o grau de custo C_X são estimados para todas as soluções candidatas, explorando uma avaliação multi-objetiva de custo-satisfação, as três metaheurísticas consideradas na proposta (NSGA-II, SPEA2 e MOCcell) devem ser capazes de encontrar um conjunto bom o suficiente de **soluções recomendadas**, que possam se aproximar do conjunto ótimo de Pareto o máximo quanto for possível P^* . Note que o problema enfrentado pela organização de desenvolvimento se trata em encontrar um conjunto de soluções candidatas que maximizem a função de satisfação S_X (Equação 32) ao mesmo tempo que o valor da função de custo C_X seja minimizado (Equação 33).

4.2. Considerações Finais

Neste capítulo, o modelo proposto é apresentado em detalhes seguindo o modelo de um problema do próximo release multi-objetivo com duas funções de satisfação e custo. Estas funções de satisfação de custo avaliam uma dada solução candidata, em que um subconjunto de requisitos, selecionados dentre um conjunto de requisitos candidatos a serem entregues no próximo release devem ser implementados. Uma boa solução candidata é definida de forma tal que minimize os custos de implementação dos requisitos selecionados ao mesmo tempo que a satisfação de clientes interessados no projeto seja maximizado. Adicionalmente, em uma abordagem baseada em riscos, a função de custo é acrescida de uma sobrecarga relativa ao custo necessário para a mitigação de riscos de software relacionados a risco. Para isto, além da informação relacionada a riscos de software e suas respectivas técnicas de mitigação, é necessário obter dados de rastreabilidade entre requisitos e seus riscos associados, assim como a rastreabilidade entre riscos e suas técnicas de mitigação associadas. Finalmente, a função de satisfação é modificada de forma a refletir o impacto dos riscos para o problema, em que os riscos críticos devem ser tratados antecipadamente, ampliando a satisfação percebida pelos clientes. Esta abordagem baseada em riscos, de modo a incorporar tanto o impacto de requisitos na satisfação assim como a sobrecarga de requisitos para o custo, adota uma arquitetura de 3 fases a serem executadas. Na primeira etapa, o impacto dos requisitos são estimados, na segunda, a sobrecarga de requisitos e finalmente na terceira etapa é possível estimar a avaliação final de custo-satisfação do subconjunto de requisitos selecionados.

5. RESULTADOS E DISCUSSÃO

No Capítulo 4, descrevemos em detalhes a abordagem proposta para o problema do próximo release incorporando modelagem de riscos. Concluída esta etapa, necessitamos de algum modo validar a proposta com base em experimentos. Apesar do foco deste trabalho ser uma modelagem para o problema do próximo release multi-objetivo, inicialmente necessitamos verificar os resultados da proposta observando se a modelagem matemática descrita no Capítulo 4 está apresentando resultados numéricos consoantes com o sentido designado pelas respectivas equações. Deste modo, antes de realizarmos um experimento para uma proposta multi-objetiva utilizando alguma metaheurística anteriormente apresentada, primeiramente realizamos um experimento reduzido com um algoritmo de busca exaustiva de modo a facilitar a interpretação dos resultados obtidos e comparamos com o que é esperado do modelo proposto. Na seção posterior, apresentamos resultados práticos em dois experimentos de maior escala com base em dados semi-reais utilizando as metaheurísticas descritas no Capítulo 3. Deste modo, também é possível validarmos o modelo de acordo com o que é esperado de um problema multi-objetivo para o próximo release.

5.1. Experimento com uma busca exaustiva utilizando o dataset Motorola

Para validarmos o modelo matemático proposto e implementado de acordo com o que é esperado de suas equações formuladas, um caso de uso reduzido é proposto. Para explorarmos todas as possíveis soluções do experimento, um algoritmo de busca exaustiva em Java foi concebido. Deste modo, considerando n requisitos candidatos, existem 2^n possibilidades de subconjuntos de requisitos capazes de formar uma possível solução.

Este experimento foi conduzido usando um subconjunto de requisitos de um projeto Motorola (BAKER *et al.*, 2006). Originalmente, este projeto possui 35 requisitos, que geram um espaço de busca com cerca de 34 bilhões (2^{35}) de soluções possíveis, o que se tornaria computacionalmente inviável, visto que uma busca exaustiva demoraria cerca de 1165 dias para ser concluída. Portanto, para validarmos a modelagem proposta, foram gerados dois experimentos com 5 e 10 requisitos com base nos 35 requisitos originais, definindo dois espaços de busca, sendo um pequeno e outro médio com 32 e 1024 soluções, respectivamente.

A Tabela 4 apresenta os 10 requisitos com seus custos e riscos associados. Note que, para cada requisito identificado rq_i , o conjunto de riscos associados é representado por expressões do tipo $rt_{i,j} = 1$, que denotam a rastreabilidade entre o requisito rq_i e o risco rk_j ,

conforme definido anteriormente na Equação 18. Note também que, para o experimento proposto com 5 requisitos, apenas os requisitos rq_1 a rq_5 são considerados, sendo os demais requisitos utilizados apenas para o exemplo maior com 10 requisitos.

Tabela 4 - Requisitos candidatos para o primeiro experimento.

Requisito (rq_i)	rq_1	rq_2	rq_3	rq_4	rq_5	rq_6	rq_7	rq_8	rq_9	rq_{10}
Custo (rqc_i)	10	40	80	400	1100	20	60	100	200	500
Risco (rk_j)	rk_1, rk_2	rk_2	rk_1, rk_3	rk_2, rk_4	rk_1, rk_3, rk_5	rk_1, rk_4	rk_3	rk_2, rk_3, rk_5	rk_4, rk_5	rk_2, rk_3

Para todos os riscos identificados, conforme ilustrado na Tabela 5, é necessário definirmos suas respectivas probabilidades e severidades, assim como também necessitamos indicar quais são as técnicas de mitigação apropriadas para cada risco. Novamente, note que, para cada risco identificado, o conjunto de técnicas de mitigação associadas é representado no modelo proposto por expressões do tipo $tt_{j,k} = 1$. Esta expressão, conforme descrito na Equação 23, denota a rastreabilidade entre o risco rk_j e a técnica de mitigação t_k .

Tabela 5 - Riscos de software do primeiro experimento e suas técnicas associadas.

Risco (rk_j)	Probabilidade de risco (rkp_j)	Severidade de risco (rks_j)	Técnicas de mitigação associadas ($tt_{j,k}$)
rk_1	Baixo	Baixo	t_1, t_6
rk_2	Baixo	Médio	t_2, t_3
rk_3	Médio	Alto	t_2, t_5
rk_4	Alto	Alto	t_1, t_2, t_4
rk_5	Muito alto	Muito alto	t_1, t_2, t_5

Após isso, para todas as técnicas de mitigação definidas, necessitamos fornecer seus custos associados, assim como também necessitamos identificar estas técnicas como sendo preventivas ou corretivas. Todos estes dados estão descritos na Tabela 6.

Tabela 6 - Técnicas de mitigação de riscos propostas para o primeiro experimento.

Técnica de mitigação (t_k)	t_1	t_2	t_3	t_4	t_5	t_6
Custo da técnica (tc_k)	10	30	230	15	50	180
Preventiva				✓	✓	✓
Corretiva	✓	✓	✓			

Além disso, a proposta também requer as informações a respeito das preferências dos clientes pelos requisitos candidatos, assim como também definimos a importância destes clientes do ponto de vista da organização responsável pelo desenvolvimento do software. A Tabela 10 apresenta estas informações, novamente ressaltam que apenas os cinco requisitos foram considerados para o primeiro caso de uso, os demais requisitos, do sexto ao décimo são apenas utilizados no segundo caso de uso com 10 requisitos.

Tabela 7 - Preferências e importância de clientes com base no primeiro experimento.

Cliente (u_i)	Importância (e_i)	Preferência do cliente ($s_{l,i}$)									
		rq_1	rq_2	rq_3	rq_4	rq_5	rq_6	rq_7	rq_8	rq_9	rq_{10}
u_1	1,0	0,2	0,4	0,8	0,4	1,0	0,2	0,7	0,6	0,5	0,7
u_2	0,7	0,1	0,5	0,2	0,9	0,4	0,2	0,5	1,0	0,4	0,9
u_3	0,5	0,3	0,3	1,0	0,6	0,6	0,3	0,4	0,9	0,7	0,3
u_4	0,3	0,8	0,6	0,4	0,4	0,3	0,3	0,6	0,3	0,4	1,0

De posse de todos os dados apresentados, agora é possível aplicarmos a abordagem proposta de modo a avaliarmos os graus de custo e satisfação para todos os possíveis subconjuntos de requisitos selecionados. Os resultados de avaliação para o exemplo com 5 requisitos é ilustrado na Figura 33. Já os valores do grau de satisfação e grau de custo, assim como a identificação dos respectivos requisitos selecionados para este caso de uso é descrito na Tabela 8. A ilustração dos resultados do segundo caso de uso para 10 requisitos candidatos é ilustrado na Figura 34. Por uma questão de simplificação, uma tabela contendo todas as 1024 soluções candidatas deste exemplo não é fornecida. Note que tais resultados simulam uma busca exhaustiva para todas as possíveis combinações dos espaços de busca de ambos os casos de uso.

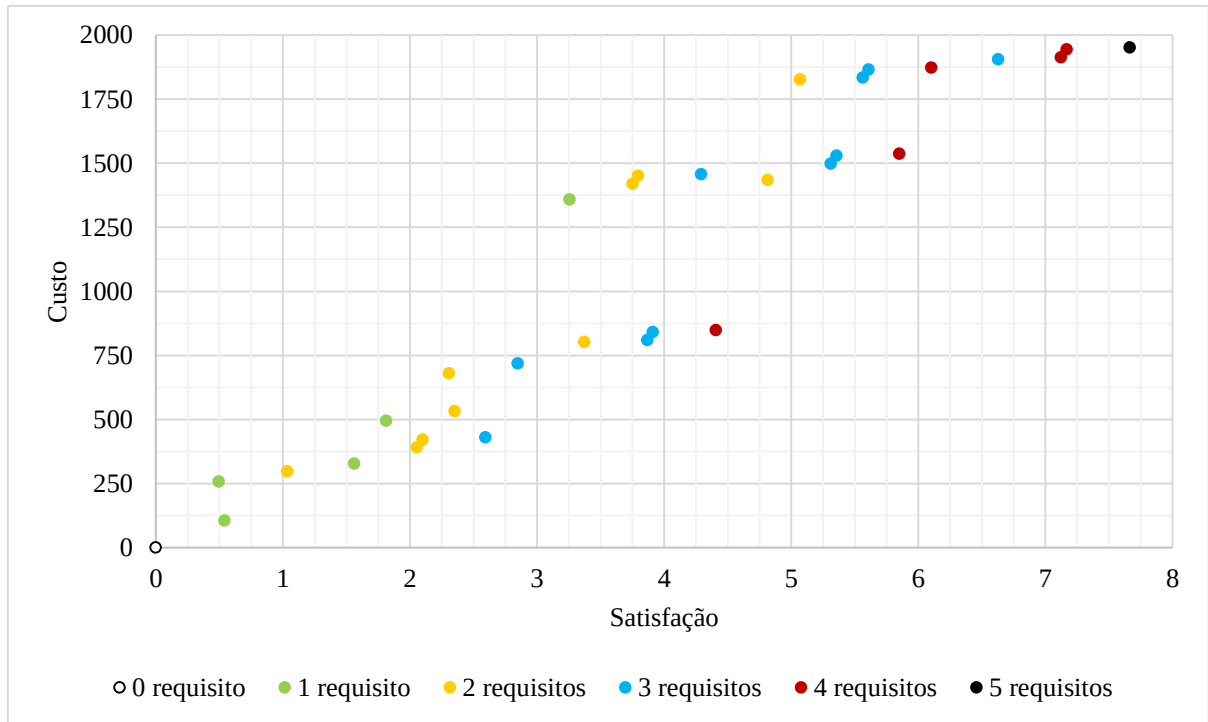


Figura 33 - Avaliação de custo-satisfação para 5 requisitos.

Tabela 8 - Seleção e valores de satisfação e custo para o caso de uso com 5 requisitos.

Seleção de requisitos ($r_{q1}, r_{q2}, r_{q3}, r_{q4}, r_{q5}$)	Satisfação	Custo
0 requisito		
00000	0,000	0,00
1 requisito		
10000	0,495	257,50
01000	0,540	105,00
00100	1,560	327,50
00010	1,812	495,00
00001	3,256	1357,75
2 requisitos		
11000	1,035	297,50
10100	2,055	391,25
01100	2,100	421,25
10010	2,307	680,00
01010	2,352	532,50
00110	3,372	802,50
10001	3,751	1419,33
01001	3,796	1450,50
00101	4,816	1434,33
00011	5,069	1827,38
3 requisitos		
11100	2,595	430,00
11010	2,847	718,75
10110	3,867	809,79
01110	3,912	840,63
11001	4,291	1456,96
10101	5,311	1498,25
01101	5,356	1528,83

10011	5,564	1834,20
01011	5,609	1865,20
00111	6,629	1905,70
4 requisitos		
11110	4,407	848,67
11101	5,851	1536,45
11011	6,104	1872,75
10111	7,124	1913,40
01111	7,169	1944,00
5 requisitos		
11111	7,664	1952,19

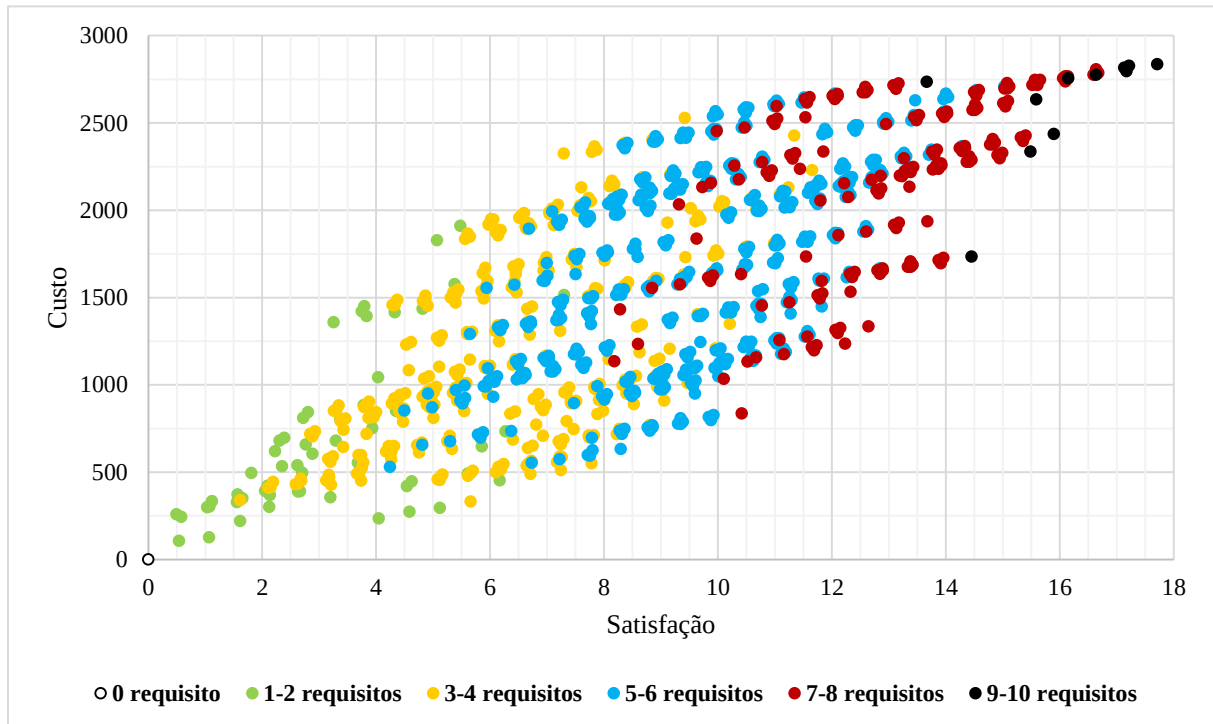


Figura 34 - Avaliação de custo-satisfação para 10 requisitos.

De modo a avaliarmos a aplicabilidade da abordagem proposta, destacamos algumas pesquisas realizadas a respeito de projetos de software reais na indústria (FELDERER, AUER e BERGSMANN, 2016) (HAN, 2014) (KAJKO-MATTSSON, LUNDHOLM e NORRBY, 2009) (NORDIN *et al.*, 2014). A percepção obtida a respeito dessas pesquisas que envolvem uma quantidade numerosa de projetos de software analisados. No caso de (HAN, 2014) em que 139 projetos de software foram avaliados é que o processo de gerenciamento de riscos não é formalmente empregado dentro do processo de desenvolvimento de software ou é realizado de maneira informal. Estas atividades são centralizadas no papel do gerente de projeto, em que os demais membros da equipe são envolvidos durante as reuniões de projeto para discutir rapidamente a respeito da evolução dos riscos detectados e monitorados. Por se tratar de uma atividade informal e por envolver uma quantidade reduzida de pessoas interessadas no processo de gestão de riscos, o processo possui uma tendência a lidar apenas com os riscos mais críticos

para o projeto. Do ponto de vista do modelo, ele é perfeitamente capaz de lidar apenas com esta categoria de riscos de maior probabilidade ou severidade, visto que os maiores valores numéricos são atribuídos justamente a estes tipos de riscos. Além disso, por reduzir a quantidade de riscos, a quantidade de possibilidades de ação para lidar com tais riscos também é diminuída sem prejuízo ao modelo proposto. Este trabalho também revela que os riscos relacionados aos objetivos de projeto são considerados os mais relevantes a serem avaliados. Segundo (FELDERER, AUER e BERGSMANN, 2016), uma pesquisa realizada em 57 empresas de software na Alemanha, Áustria e Suíça, apesar de também revelar que a maior parte das empresas não emprega uma metodologia formal de gerência de riscos, os participantes da pesquisa informaram os casos de teste são priorizados de acordo com a avaliação de risco do projeto, nos casos em que ele é realizado. Ele corrobora o dado apontado por (HAN, 2014) em que os riscos relacionados às questões técnicas do produto de software são considerados os mais relevantes a serem avaliados, sendo tão importantes quanto os riscos de projeto. Deste modo, com bases nestes dois trabalhos, é possível avaliar que os riscos se relacionam de alguma maneira com as funcionalidades esperadas pelo produto de software. Infelizmente, nenhum dos trabalhos detalhou como este relacionamento ocorre, apenas informando que tal relacionamento existe. Porém, observamos anteriormente em (CLELAND-HUANG *et al.*, 2005) (SÁNCHEZ *et al.*, 2011) dois modelos de rastreabilidade de requisitos e riscos aplicados a projetos de software crítico em que a segurança de pessoas era um fator importante. Tais modelos podem servir como base para definir este relacionamento de requisitos e riscos, devendo ser simplificado e generalizado para não se restringir a softwares críticos, mas a qualquer outra categoria de software. Finalmente, os trabalhos (KAJKO-MATTSSON, LUNDHOLM e NORRBY, 2009) (NORDIN *et al.*, 2014) declaram a dificuldade de avaliar a eficácia de um processo de gerenciamento de riscos, além que a quantidade de recursos aplicada a este fim é um fator altamente variável de acordo com o tamanho do projeto, tamanho da organização, a quantidade de pessoas envolvidas e a própria natureza do produto de software. Por conta dessa dificuldade em estabelecer a quantidade de recursos a ser gasta em tal processo e a incapacidade de medir eficazmente o quanto este processo de gerência de riscos beneficia o sucesso do projeto é um fator que impacta negativamente o processo de gerência de riscos, sendo muitas vezes abandonado ou realizado apenas de maneira informal, apesar das empresas reconhecerem a importância de tal processo. Para solucionar este problema, a pesquisa realizada em (NORDIN *et al.*, 2014) com especialistas de domínio e membros da indústria local na Malásia aponta que o uso de ferramentas de gerência de risco que demonstram claramente como o tratamento de risco influencia no sucesso do projeto seria um fator importância para que mais

empresas adotassem um processo formal de gerência de riscos. Deste modo, para que o modelo proposto seja aplicável em cenários reais de desenvolvimento de software, ele deve ser convertido em uma ferramenta de gerência de risco, acrescido de uma interface intuitiva de pré-processamento dos dados de entrada, estabelecendo as informações relevantes a respeito de requisitos, riscos e possibilidades de ação para tratamento de tais riscos. Como saída, a ferramenta, além de fornecer os gráficos de custo-satisfação e valoração de custo e satisfação de acordo com cada possibilidade de seleção de requisitos candidatos, seria desejável destacar um dado à parte dedicado a informar como os riscos selecionados contribuem isoladamente para maximizar a função de satisfação do modelo, obtido através da Equação 20 dada por imp_i . Além disso, a quantidade de recursos estimados a serem empregados no processo podem ser obtidos a partir da Equação 34, dada através da função rkc_i . Portanto, o modelo permite a criação de uma ferramenta capaz de estimar a quantidade de recursos gastos no processo de gerenciamento de riscos, além de também ser capaz de informar o quanto este tratamento de riscos afeta o sucesso do projeto de software.

Resultados em ambos os experimentos mostram que, conforme mais requisitos são progressivamente selecionados, existe uma maior tendência de compartilhamento de riscos entre requisitos, aliviando os custos das penalidades introduzidas pelas técnicas de mitigação de riscos dimensionados pelas probabilidades de seus riscos associados. Tal comportamento possui o potencial de tornar progressivamente mais baratos os custos individuais de cada requisito. Tomemos como exemplo a solução que seleciona apenas o requisito rq_1 , que está associado aos riscos rk_1 e rk_2 , que por sua vez se associam às técnicas t_1, t_2, t_3, t_6 . Neste caso, individualmente, o requisito rq_1 possui um custo individual de 257,50, visto que o requisito será responsável por custear todas estas quatro técnicas de mitigação dos riscos associados ao requisito rq_1 , conforme observado na Equação 39. Agora, tomemos como exemplo contrário, a solução em que todos os cinco requisitos do primeiro caso de uso são selecionados. Neste caso, o requisito rq_1 possui um custo individual de 90,74. Conforme podemos observar na Figura 35, isto ocorre pelo fato do requisito rq_1 compartilhar riscos com os demais requisitos rq_2, rq_3, rq_4 e rq_5 . Outro fator que corrobora com este fenômeno é que, mesmo os riscos que não são compartilhados com rq_1 , possuem técnicas de mitigação semelhantes às técnicas associadas aos riscos rk_1 e rk_2 , o que também contribui para minimizar a avaliação final de custo de tratamento de riscos para o requisito 1 (rkc_1). Visto que o custo final de rkc_1 é dado com base na fórmula destacada na Equação 40.

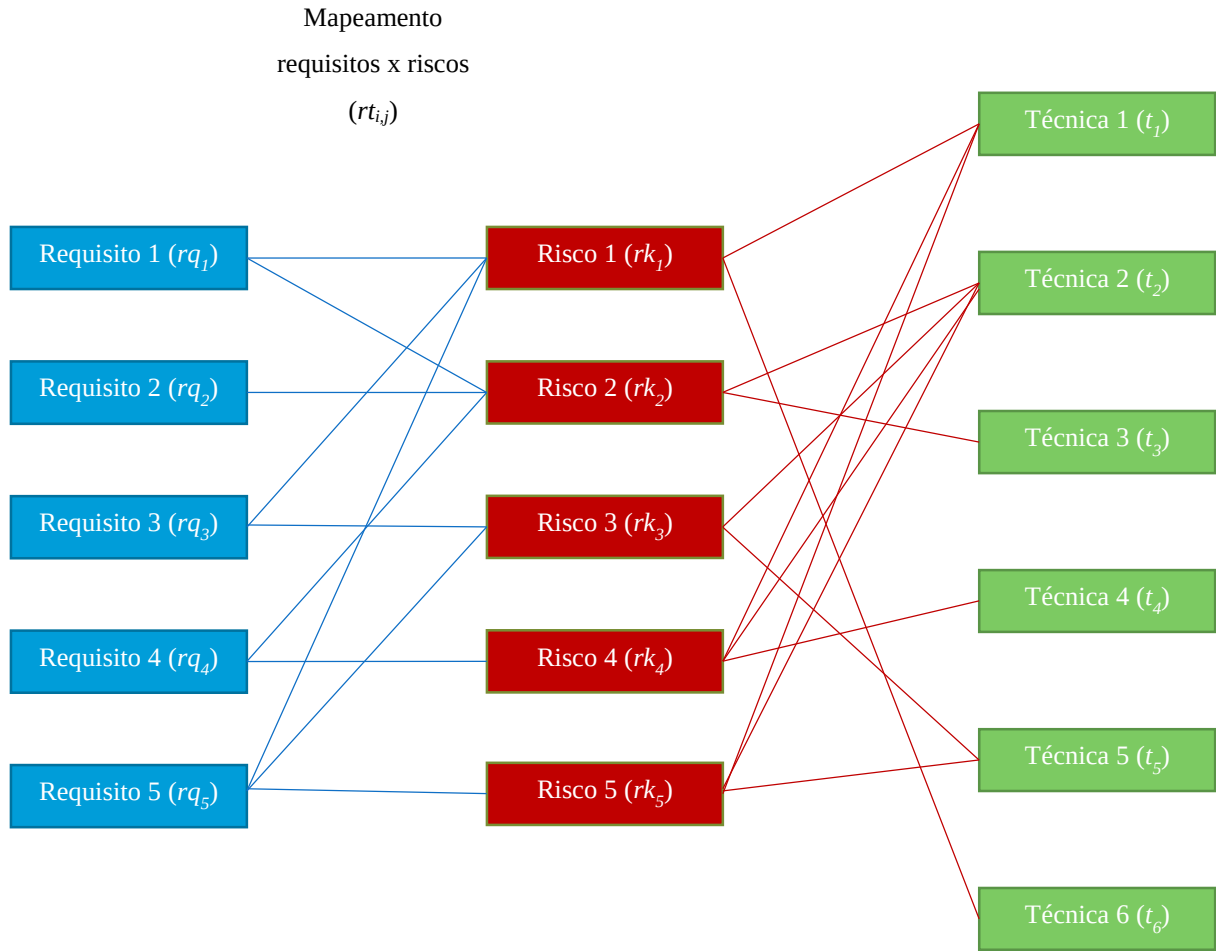


Figura 35 - Mapeamento dos relacionamentos $rt_{i,j}$ e $tt_{j,k}$ para o caso de uso com 5 requisitos.

$$rk_{c_1} = tp_{1,1} \cdot (tc_1) + tp_{1,6} \cdot (tc_6) + tp_{2,2} \cdot (tc_2) + tp_{2,3} \cdot (tc_3) \quad (39)$$

$$rk_{c_1} = tp_{1,1} \cdot (tc_1/5) + tp_{1,6} \cdot (tc_6/3) + tp_{2,2} \cdot (tc_2/7) + tp_{2,3} \cdot (tc_3/3) \quad (40)$$

Continuando a análise sobre o requisito rq_1 , tomando como base a solução em que apenas este requisito é selecionado, observamos que, apesar de possuir um custo de implementação baixo ($rqc_1 = 10$), ele ainda sofre uma grande penalidade no seu valor final devido ao custo associado ao tratamento de riscos. Este fenômeno se torna ainda mais impactante quando os riscos possuam uma grande probabilidade associada. Deste modo, a estimativa de custo para tratamento de riscos aproxima-se de seu valor integral, sendo o mesmo fenômeno observando quando os riscos associados estão relacionados às técnicas de mitigação da categoria preventiva.

Como uma consequência adicional, os resultados mostram que, em soluções recomendadas cujos requisitos selecionados possuam um grande grau de severidade de seus riscos associados, o impacto destes riscos na avaliação de satisfação da solução se torna mais

intensa. Por exemplo, considere o requisito rq_2 , que está associado apenas ao risco rk_2 , que possui um grau de severidade média. Caso a função de satisfação levasse em consideração apenas a satisfação dos clientes, a função retornaria um valor igual a 1,08. Entretanto, ao incluir a função de impacto de riscos de requisitos na função de satisfação (Equação 33), a avaliação final de satisfação para uma solução candidata que seleciona apenas o requisito rq_2 retorna um valor igual a 0,54. Em contrapartida, considere uma solução candidata em que o requisito rq_5 é selecionado, requisito este que está associado aos riscos rk_1 , rk_3 e rk_5 que possuem severidades baixa, alta e muito alta, respectivamente. Nesta situação, observando apenas a avaliação de satisfação de clientes na seleção deste requisito, possuiríamos uma avaliação de satisfação final 1,67. Porém, ao novamente incluirmos a função de avaliação de severidade, a avaliação de satisfação sobe para 3,2565.

Em resumo, este experimento também revela a importância de introduzirmos o conceito de análise de risco nos custos de requisitos e na satisfação de clientes. Por exemplo, em uma abordagem tradicional que não considera uma análise de risco, o custo total de selecionarmos todos os requisitos é igual à 1630. Em contrapartida, na abordagem proposta, este custo total cresce para 1952, sendo mais preciso e realista devido à inclusão dos custos relacionados à adoção de um processo de gerenciamento de riscos.

5.2. Experimento multi-objetivo utilizando os datasets Microsoft Word e ReleasePlanner

O experimento anterior demonstrou que o modelo apresentado se comporta conforme descrito no Capítulo 4, mas ainda resta validarmos o modelo do ponto de vista de uma abordagem para o problema do próximo release multi-objetivo. Para isto, dois novos experimentos foram realizados com dois conjuntos de dados de projetos reais de software (KARIM e RUHE, 2014). Estes dados estão disponíveis para download na internet¹. O conjunto de dados utilizado para o Experimento I contém 25 requisitos e 8 stakeholders relacionado à uma ferramenta denominada ReleasePlanner². O conjunto de dados utilizado para o Experimento II, por sua vez, conta com 50 requisitos e 4 stakeholders, desta vez associado à um projeto do editor de texto Microsoft Word. Podemos diretamente encontrar nestes conjuntos de dados das informações relativas aos custos dos requisitos, preferências dos clientes em relação aos requisitos e a importância de cada cliente para o projeto. Entretanto, não foram

¹ <https://sites.google.com/site/mrkarim/data-sets>

² <http://www.expertdecisions.com>

encontrados dados relativos à riscos e técnicas de mitigação de riscos. Deste modo, estes dados foram estimados indiretamente de maneira sintética com base em outras informações disponíveis nestes conjuntos de dados. Assim, o Experimento I conta com 4 riscos e 4 técnicas de mitigação, enquanto que o Experimento II possui 8 riscos e 6 técnicas de mitigação.

Foram realizadas outras buscas em artigos sobre gerenciamento de riscos afim de obter dados reais relativos a riscos e técnicas, entretanto, nenhum artigo disponibilizava estes dados. A melhor aproximação que obtivemos destes dados foi encontrada em (NEVES *et al.*, 2014), que realizou uma pesquisa em quatro empresas de pequeno porte em Minas Gerais a respeito da metodologia de gestão de riscos adotada por estas organizações. Apesar dos dados coletados não estarem disponíveis abertamente para uso no experimento, consideramos esta pesquisa relevante ao demonstrar que as empresas de fato estabelecem esta relação entre os fatores de risco do projeto e como eles podem ser tratados por técnicas de mitigação. Também observamos que os riscos são priorizados de acordo com seus valores de probabilidade e severidade, e que estes valores estavam correspondendo às listas de riscos mais importantes fornecidas pelos membros destas empresas. Portanto, um modelo capaz de levar em conta tais dados ao formar soluções com boas sugestões de seleção de requisitos é desejável.

Para execução dos experimentos, três metaheurísticas multi-objetivas foram selecionadas: NSGA-II, SPEA2 e MOCell. Adicionalmente, um algoritmo de busca aleatória também foi empregado de modo a realizar um “teste de sanidade” (HARMAN *et al.*, 2012). Para confirmar que os algoritmos superaram satisfatoriamente os resultados obtidos pelo algoritmo de busca aleatório, utilizamos o teste estatístico de Wilcoxon-Mann-Whitney (FAY e PROSCHAN, 2009) com um valor-*p* igual a 0,05, denotando um grau de confiança de 95%.

Sabemos que os algoritmos genéticos utilizam operadores de seleção, cruzamento, mutação e substituição. Para isto, configuramos os três algoritmos para utilizar a técnica de torneio para seleção que não recebe nenhum parâmetro adicional. A primeira justificativa se dá pelo operador de torneio ser considerado o operador de seleção padrão definido na implementação do NSGA-II, facilitando assim uma comparação entre os algoritmos, a segunda justificativa se dá, ao observarmos as vantagens apresentadas pela técnica de torneio em comparação com as técnicas de seleção da roleta que possui uma desvantagem em alocar um espaço muito grande para os indivíduos de maior *fitness*, fazendo com que o algoritmo genético adote um viés muito elitista especialmente nos casos que exista uma grande variância no *fitness* da população. A seleção por *ranking*, apesar de amenizar este comportamento do processo de seleção, ainda não resolve o viés de seleção em que os cromossomos de maior *fitness* possuem

uma grande probabilidade de seleção. A seleção por torneio, apesar de garantir uma maior possibilidade de escolha em pais de maior *fitness*, realiza inicialmente uma seleção aleatória sem considerar o *fitness*, após este procedimento, cada par de cromossomos selecionados são então comparados em relação ao seu *fitness*, sendo então destacado o indivíduo melhor avaliado. Portanto, existe uma possibilidade que este par comparado contenha dois pais de *fitness* baixo dada a aleatoriedade da primeira etapa de seleção. Em sequência, a técnica de cruzamento adotada foi o cruzamento de metade uniforme (*HUX Crossover*) com uma probabilidade de cruzamento de 90%. Esta técnica garante que 50% das características distintas de ambos os pais sejam herdadas, independentemente da posição destas características no cromossomo, o que não é garantido nas tradicionais técnicas de corte único ou duplo. A probabilidade de 90% foi fixada após a realizados testes com uma probabilidade de 100% apresentou resultados piores, visto que, caso o operador de cruzamento não seja selecionado, os filhos gerados serão cópias exatas dos pais, sendo em seguida submetidos ao operador de mutação, que pode melhorar uma solução pré-selecionada que já apresentava um bom *fitness*, diferentemente da ativação do operador de cruzamento que gera um indivíduo novo sem garantia de que este indivíduo gerado seja bem avaliado como solução do problema. O operador de mutação selecionado foi o de *flipping* de bits com uma probabilidade de $1/n$, em que n corresponde ao número de requisitos de cada experimento, portanto formando probabilidade de cruzamento de $1/25$ para o Experimento I e $1/50$ para o Experimento II. O operador de *flipping* foi selecionado de modo a obter um melhor controle a respeito das mutações realizadas no cromossomo, visto que uma probabilidade de $1/25$, como no caso do Experimento I afetaria 10% das posições do cromossomo afetado. O operador de intercambiação fixa este valor em duas posições do cromossomo, não permitindo um grau de aleatoriedade a respeito da quantidade de posições afetadas enquanto que o operador de reversão altera o valor das posições vizinhas em relação à posição selecionada, aumentando a quantidade de valores afetados, o que gera uma maior desconexão em relação à probabilidade de mutação utilizada.

A parametrização dos algoritmos é descrita na Tabela 9, cujos valores foram obtidos após sucessivas calibrações nos parâmetros. Conforme podemos observar, a população para cada experimento é quadruplicado em relação ao número de requisitos candidatos. A respeito do número de gerações, que representa o critério de parada do NSGA-II, SPEA2 e MOCcell, esta é configurada para cada experimento considerando o número de requisitos candidatos (n) e o tamanho do espaço de busca (2^n), calibrados utilizando a expressão $5 \cdot n \cdot 2^{n/25}$. Note que, como a saída dos algoritmos SPEA2 e MOCcell não são a população, mas o arquivo de soluções

não-dominadas, significa que o tamanho do arquivo deve ser igual à população regular. O feedback, por sua vez, é um parâmetro exclusivo do MOCell, que move aleatoriamente soluções do arquivo para a população. O algoritmo aleatório recebe como parâmetro apenas a quantidade de soluções a serem avaliadas, que é equivalente ao tamanho da população multiplicada pela quantidade de gerações dos demais algoritmos. Finalmente, é necessário apontarmos que o MOCell não foi capaz de executar o Experimento II com uma população igual a 200 cromossomos, ocasionando um erro de posição inválida de array ao buscar as oito soluções na vizinhança. Deste modo, a população do MOCell para o Experimento II foi reduzida para 196 indivíduos, que não prejudica de maneira significativa a saída do algoritmo, pois ela é obtida pelo arquivo, que possui um tamanho máximo de 200 indivíduos. Curiosamente, para o primeiro experimento com população de 100 indivíduos o MOCell não acusou erro. Após inspeção no código, foi verificado um erro no cálculo de vizinhança ao gerar a população inicial, mas, visto que, mesmo com uma população ligeiramente menor em comparação com os outros dois algoritmos, a quantidade de iterações de todos os algoritmos são iguais e portanto, os três algoritmos iriam explorar a mesma quantidade de soluções no espaço de busca, foi preferível não adulterar a codificação do algoritmo pré-existente no jMetal. Para cada algoritmo, ambos os experimentos foram executados 100 vezes.

Tabela 9 - Parametrização dos Experimentos.

	Experimento I (25 requisitos)	Experimento II (50 requisitos)
População	100	200
Arquivo	100	200
Feedback	4	4
Gerações	250	1000
Soluções Avaliadas	25.000	200.000

Os experimentos foram avaliados utilizando os indicadores de qualidade de hipervolume normalizado (*HVR*) e espalhamento (Δ), descritos anteriormente no Capítulo 2. Devemos lembrar que estes indicadores de qualidade exigem algum conhecimento a respeito da Fronteira de Pareto (PF^*), que contém o conjunto de soluções ótimas ou pelo menos de uma fronteira de referência com as melhores soluções conhecidas. No Experimento I, considerando o espaço de busca reduzido de 2^{25} soluções possíveis, a Fronteira de Pareto foi encontrada com

uso de um algoritmo de busca exaustiva, em que 105 soluções não-dominadas foram encontradas. No Experimento II, entretanto, considerando o espaço de busca igual a $2^{50} = 1.125.899.906.842.624$ soluções possíveis, apenas foi possível obter uma fronteira de referência com as melhores soluções obtidas pelas metaheurísticas adotadas, totalizando 386 soluções não-dominadas no conjunto de referência.

A plataforma de execução de todos estes algoritmos (busca aleatória, busca exaustiva, NSGA-II, SPEA2 e MOCell) foi o jMetal, codificados em Java. Estes algoritmos foram executados em um microcomputador equipado com um processador de quatro núcleos Intel Core i5 2400 de 3.40 GHz com 6MB de memória cache e memória RAM DDR3 com dois pentes de 4GB, totalizando 8GB com frequência de 667MHz. Considerando que cada algoritmo foi executado 100 vezes para cada experimento, estas execuções foram alocadas em 4 threads concorrentes, correspondente ao número de núcleos do processador.

Após a execução dos experimentos, os resultados obtidos foram compilados na Tabela 10. Podemos observar que todos os três algoritmos obtiveram uma performance significativamente melhor que o algoritmo aleatório, em todos os aspectos exceto no tempo de execução. Mas, tomando como exemplo a execução do NSGA-II no Experimento I que tomou apenas 1 segundo a mais que a busca aleatória e obteve o melhor desempenho para os parâmetros de Hipervolume e Espalhamento, esta diferença de tempo quase imperceptível não justificaria o uso de um algoritmo com performance muito pior. Ademais, ao notarmos que, no Experimento I, o algoritmo exaustivo encontrou a Fronteira de Pareto com todos os 105 valores ótimos em 38 minutos, uma metaheurística capaz de se aproximar destes valores em cerca de 9 segundos é um resultado bastante interessante.

Ao omitirmos a busca aleatória e compararmos as três metaheurísticas selecionadas, observamos que o SPEA2 obteve os piores resultados em ambos os experimentos, com um tempo de execução muito mais lento em contraste com o NSGA-II e MOCell. Isto se justifica ao observarmos que o algoritmo possui uma complexidade de $O(M^2 \log M)$, em que M é a soma da população regular e do arquivo (ZITZLER, LAUMANNNS e THIELE, 2001). No caso do NSGA-II, este possui uma complexidade $O(MN^2)$, onde M é a quantidade de gerações e N , o tamanho da população (DEB *et al.*, 2002). No caso do MOCell, os autores não especificaram a complexidade do algoritmo (NEBRO *et al.*, 2009), mas visto que o operador de dominância é semelhante ao utilizado pelo NSGA-II, podemos inferir que a complexidade deste algoritmo se aproxima do desempenho do NSGA-II, conforme demonstrado nos resultados.

No Experimento I, o NSGA-II se mostrou melhor nos três indicadores de desempenho. No Experimento II, entretanto, o MOCcell apresentou um melhor indicador de espalhamento, o que é um resultado típico para esta metaheurística, conforme apontado pelos autores do algoritmo e confirmado por um dos trabalhos relacionados que realizou experimentos comparando o NSGA-II e MOCcell (SARAIVA *et al.*, 2016). Além disso, no Experimento II, o MOCcell também obteve um Hipervolume ligeiramente melhor que o NSGA-II. Podemos inferir que, dado que o conjunto de valores ótimos se encontram densamente agrupados ao longo da Fronteira de Pareto, uma metaheurística que tem como princípio observar valores de vizinhança durante a etapa de seleção de pais para cruzamento se mostrou vantajoso para o experimento proposto. Adicionalmente, o uso de um arquivo para armazenar as soluções ótimas permitiu que a população regular se concentrasse apenas em gerar novas soluções candidatas para o conjunto final obtido como saída do algoritmo.

Tabela 10 - Hipervolume normalizado, espalhamento e tempo de execução.

	NSGA-II	SPEA2	MOCcell	Busca Aleatória
	HIPERVOLUME NORMALIZADO. Média e desvio padrão			
Experimento I	$9,99e - 01_{2,1e-04}$	$9,59e - 01_{7,7e-02}$	$9,91e - 01_{7,7e-02}$	$4,49e - 01_{1,4e-01}$
Experimento II	$9,98e - 01_{1,7e-04}$	$9,83e - 01_{3,3e-02}$	$9,99e - 01_{1,3e-04}$	$2,66e - 01_{1,2e-01}$
	ESPALHAMENTO. Média e desvio padrão			
Experimento I	$3,56e - 01_{1,8e-02}$	$5,24e - 01_{1,3e-02}$	$4,63e - 01_{2,2e-02}$	$6,73e - 01_{6,8e-02}$
Experimento II	$3,76e - 01_{2,5e-02}$	$3,88e - 01_{1,8e-02}$	$3,25e - 01_{7,0e-03}$	$7,49e - 01_{5,5e-02}$
	TEMPO DE EXECUÇÃO			
Experimento I	9"	32"	12"	8"
Experimento II	2'04"	9'28"	2'22"	1'49"

Podemos observar os gráficos com as saídas dos Experimentos I e II na Figura 36 e na Figura 37, respectivamente. É importante notar que, por uma questão de clareza nos dados apresentados nos gráficos, as soluções aqui apresentadas representam o melhor valor de Hipervolume Normalizado (*HVR*) para cada algoritmo dentro de todas as 100 execuções de ambos os experimentos. Nota-se uma clara dificuldade em distinguir os resultados das três metaheurísticas em relação aos valores ótimos e de referência para ambos os experimentos. Somente no Experimento II conseguimos, com um certo grau de dificuldade, notarmos valores levemente díspares entre as metaheurísticas utilizadas. Isto pode ter ocorrido devido ao fato

que, como o conjunto de referência do Experimento II contém 386 soluções candidatas, cada algoritmo com uma saída máxima de 200 cromossomos permitiu que estes algoritmos potencialmente encontrassem valores diferentes entre si, além de não ser possível atingir todos os 386 valores de referência.

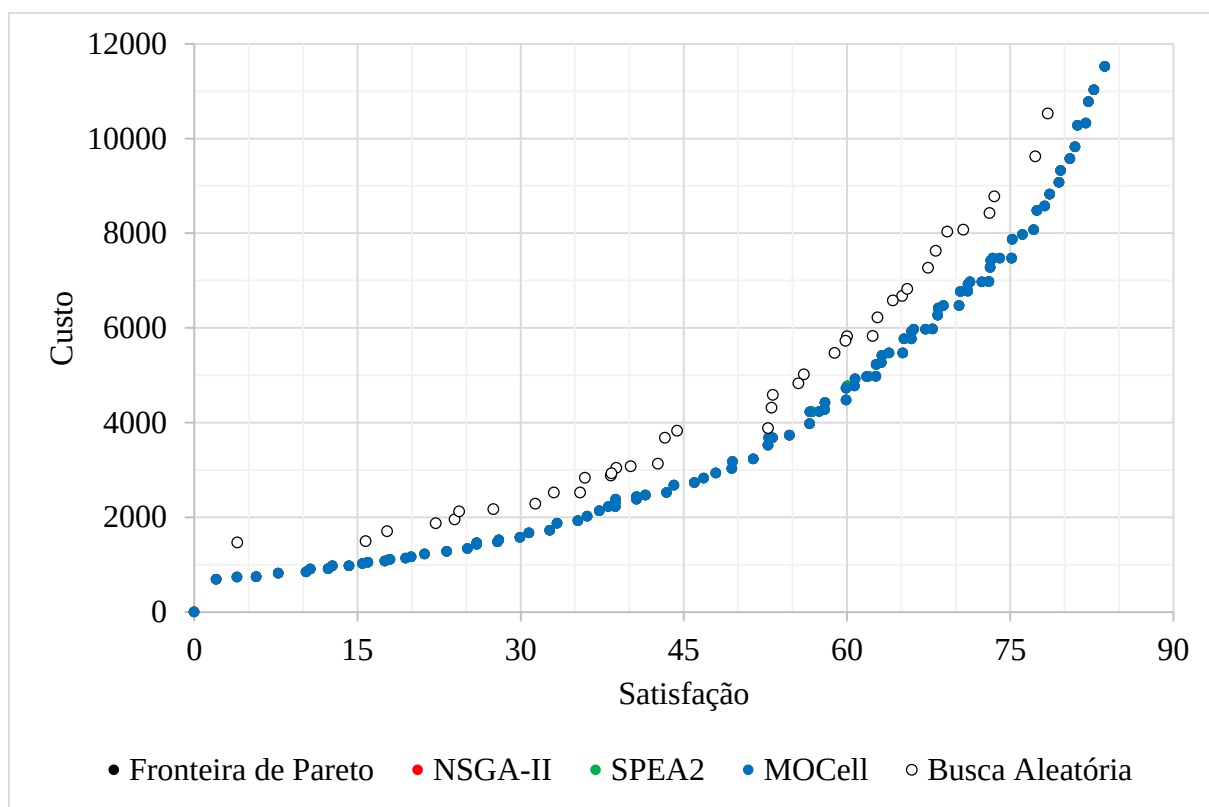


Figura 36 - Resultados para o Experimento I com 25 requisitos candidatos.

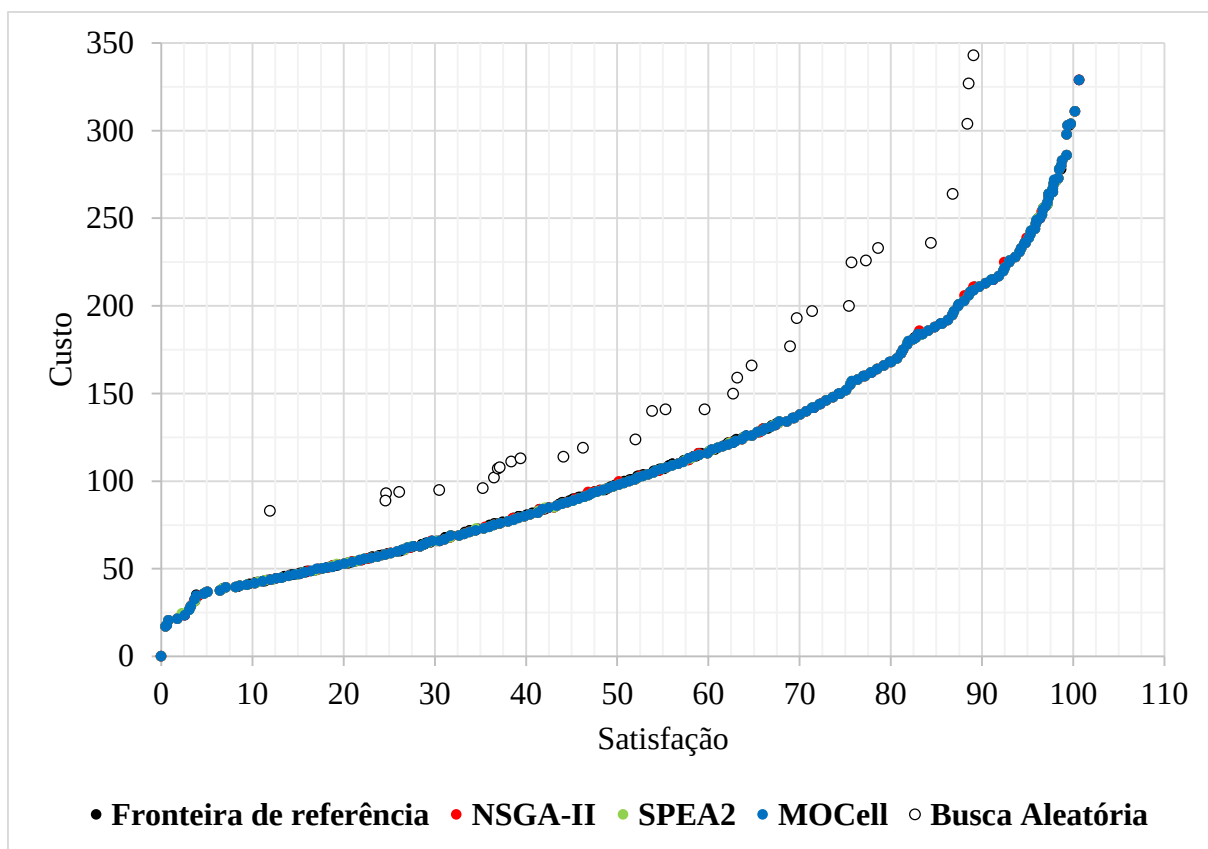


Figura 37 - Resultados para o Experimento II com 50 requisitos candidatos.

Finalmente, a Tabela 11 nos apresenta o valor médio e o desvio padrão da quantidade de soluções que cada algoritmo encontrou em todas as execuções e que também estão presentes no conjunto ótimo para o Experimento I ou no conjunto de referência para o Experimento II. Um indício do que foi discutido ao demonstrarmos os resultados de hipervolume normalizado para o Experimento II se confirma aqui, ou seja, a ausência de um arquivo capaz de armazenar as soluções não-dominadas encontradas pelo NSGA-II prejudicou seu desempenho final, visto que, dentro de um único conjunto de população, este algoritmo deve armazenar as soluções não-dominadas e também gerar novas soluções candidatas. No caso do SPEA2 e MOCcell, adicionalmente à população regular, responsável exclusivamente por gerar soluções novas, um arquivo de tamanho igual à população possuía uma atribuição exclusiva de armazenar as melhores soluções encontradas, o que claramente beneficiou estas duas metaheurísticas ao observarmos os resultados do Experimento II. A busca aleatória apresentou um resultado claramente insatisfatório, ao quase não encontrar valores ótimos no Experimento I e, de maneira ainda pior, não encontrou uma única solução equivalente ao conjunto de referência no Experimento II.

Tabela 11 - Média e desvio padrão de quantidade de soluções encontradas que estão presentes no conjunto ótimo ou de referência.

	Experimento I	Experimento II
NSGA-II	$9,78e + 01_{1,27e+00}$	$1,63e + 02_{5,18e+00}$
SPEA2	$9,85e + 01_{1,58e+00}$	$1,89e + 02_{2,82e+00}$
MOCcell	$9,92e + 01_{1,33e+00}$	$1,92e + 02_{2,71e+00}$
Busca Aleatória	$0,07e + 00_{2,56e-01}$	$0,00e + 00_{0,00e+00}$

Finalmente, os resultados para o teste estatístico de Wilcoxon-Mann-Whitney corroboram com a hipótese levantada acima a respeito do uso de um arquivo pelos algoritmos SPEA2 e MOCcell para armazenar as melhores soluções encontradas, conforme ilustrados especialmente na Tabela 12 e de maneira secundária na Tabela 13 que observa o indicador de espalhamento. Note que um símbolo $\blacktriangle$ sinaliza que o algoritmo da linha obteve uma performance melhor que o algoritmo da coluna. O símbolo ∇ sinaliza que o algoritmo da coluna obteve uma performance melhor que o algoritmo da linha. Finalmente, o símbolo $-$ confirma a hipótese nula, em que os algoritmos, para o dado indicador de qualidade avaliado, não obtiveram diferenças significativas nas suas distribuições de valores para este indicador. É importante destacar os resultados de ambos os experimentos I e II são apresentados nas tabelas, de maneira seqüencial para cada par de algoritmos comparados. Tanto o SPEA2 quanto o MOCcell obtiveram mais valores de pico para o hipervolume, apesar do NSGA-II sinalizar um valor médio de hipervolume maior no Experimento I, conforme observamos anteriormente na Tabela 10. Isto também se confirma ao observarmos o desvio padrão maior para o SPEA2 e MOCcell para o Experimento I, também descrito na Tabela 10. Apesar disto, consideramos que uma melhor regularidade de desempenho observado no NSGA-II com um valor de hipervolume médio maior é um resultado mais desejável do que uma maior variação de valores nos casos do SPEA2 e MOCcell. No Experimento II, o SPEA2 não obteve uma diferença de desempenho significativa em relação ao NSGA-II e, considerando o tempo bastante elevado de execução deste algoritmo, consideramos que o NSGA-II aponta um resultado mais aceitável. Porém, de maneira incontestável, o MOCcell obteve o melhor desempenho no Experimento II sob qualquer ponto de vista, seja hipervolume, espalhamento ou quantidade de soluções encontradas que também estão presentes no conjunto de referência. O tempo de execução ligeiramente mais lento do MOCcell em relação ao NSGA-II não é justificativa suficiente por optar em utilizar NSGA-II em detrimento do MOCcell. O comportamento observado em um algoritmo de busca

celular como o MOCcell claramente foi benéfico no caso do Experimento II em que tivemos um cenário maior com 50 requisitos candidatos, visto que o teste de Wilcoxon sinalizou um desempenho melhor tanto no indicador de hipervolume normalizado quanto no espalhamento, enquanto que o NSGA-II obteve um melhor desempenho em relação ao espalhamento em um cenário médio tal como observado no Experimento I para 25 requisitos candidatos. Finalmente, todos os algoritmos obtiveram uma performance melhor que a busca aleatória sob qualquer aspecto avaliado, excetuando o tempo de execução.

Tabela 12 - Resultados do teste de Wilcoxon para o hipervolume normalizado.

	SPEA2	MOCcell	Busca Aleatória
NSGA-II	▽–	▽▽	▲▲
SPEA2		▽▽	▲▲
MOCcell			▲▲

Tabela 13 - Resultados do teste de Wilcoxon para o indicador de espalhamento.

	SPEA2	MOCcell	Busca Aleatória
NSGA-II	▲▲	▲▽	▲▲
SPEA2		▽▽	▲▲
MOCcell			▲▲

Dentre as principais ameaças à validade destes experimentos, podemos elencar como principal *ameaça interna*, a vulnerabilidade inerente à adoção de conjuntos de dados semi-reais, exigindo a inclusão de dados sintéticos. Apesar da calibração ser obtida empiricamente, uma afinação paramétrica permitiria melhores recomendações de requisitos em cenários mais complexos. Dentre as *ameaças externas*, ao considerarmos os experimentos com 25 e 50 requisitos, uma maior quantidade de replicações e adaptações para uma maior gama de datasets com mais requisitos, riscos, técnicas de mitigação e clientes são desejáveis de forma a permitir uma maior generalização a respeito da aplicabilidade do modelo proposto e dos resultados obtidos, permitindo a identificação de algum viés no modelo ou nos experimentos. De fato, tais resultados podem ser aplicáveis em situações em que suposições similares são mantidas. Em

contrapartida, o modelo precisa ser adaptado de maneira correspondente aos diferentes cenários encontrados, por exemplo, no caso de requisitos interdependentes. Em relação às *ameaças de construção*, o modelo proposto adota conceitos e modelos de informação que foram aplicados de maneira bem sucedida nos trabalhos relacionados. As suposições e fundamentos aplicados no modelo fazem sentido para os tipos de experimentos discutidos, entretanto, esta formulação pode ser alterada no caso em que diferentes abstrações para riscos são percebidas e adotadas. Finalmente, em relação às *ameaças de conclusão*, como uma maneira de combater a natureza estocástica dos algoritmos de busca e garantir uma comparação justa, NSGA-II, SPEA2, MOCell e a busca aleatória foram executados múltiplas vezes em relação a cada um dos dois experimentos. Isto foi feito com objetivo de superar a aleatoriedade inerente em tais técnicas.

5.3. Considerações Finais

Neste capítulo foi apresentado dois experimentos, o primeiro experimento para um conjunto de dados com 5 requisitos e o segundo experimento com dois conjuntos de dados para 25 e 50 requisitos. O primeiro experimento reduzido em que foi realizado uma busca exaustiva foi útil para demonstrar o comportamento esperado do modelo concebido na proposta, por exemplo a questão de compartilhamento de riscos entre requisitos e como este compartilhamento reduz o custo individual de cada requisito, visto que esse custo é partilhado entre todos os requisitos que estão associados a um mesmo risco e que foram selecionados para entrega na próxima versão do software. Isto é interessante por evidenciar que requisitos com riscos exclusivos denotam que um esforço adicional será exigido para resolver estes riscos que afetam um conjunto reduzido de riscos. O segundo experimento utilizou dois conjuntos de dados semi-reais, de 25 e 50 requisitos, aplicados a três metaheurísticas de busca multi-objetivo: NSGA-II, SPEA2 e MOCell. Os resultados destes três algoritmos foram comparados em relação a uma busca aleatória afim de realizar um “teste de sanidade”, no caso do primeiro experimento, também foi comparado em relação a uma busca exaustiva, demonstrando o quão bem estas metaheurísticas se aproximam da solução ótima. No caso do experimento para o conjunto de 50 requisitos candidatos, entretanto, não foi possível obter um resultado através de busca exaustiva dada a crescente complexidade computacional necessária para explorar a totalidade do espaço de busca. Portanto, um conjunto de referência foi composto com base nas melhores soluções encontradas por todos os algoritmos. Com base nos resultados, observamos uma boa qualidade em soluções do NSGA-II e MOCell no primeiro caso com 25 requisitos, obtendo os melhores índices de hipervolume normalizado e espalhamento. No segundo cenário com 50 requisitos candidatos, o MOCell se destacou isoladamente, dada as características

especiais de um algoritmo de busca celular, em contraste com o NSGA-II e SPEA2 que possuem um modelo mais tradicional de busca sem considerar a vizinhança das soluções encontradas. Este dado é confirmado pelos indicadores de qualidade de hipervolume normalizado, espalhamento, pelo teste estatístico de Wilcoxon com grau de confiança de 95% e também ao observarmos a quantidade média e desvio padrão de soluções para cada algoritmo que mais obteve soluções similares ao conjunto de referência de melhores soluções conhecidas para o cenário.

6. TRABALHOS RELACIONADOS

Após apresentada a abordagem multi-objetivo baseada em riscos, proposta para o problema do próximo release, podemos então descrever em detalhes os aspectos mais relevantes dos trabalhos pré-existentes na literatura que possuem alguma relação com o modelo proposto.

Para tal, um critério de busca se fez necessário para reduzirmos a quantidade de trabalhos para um nível aceitável. Deste modo, foi realizada uma revisão sistemática na literatura, conforme critérios de busca apresentados na Seção 6.1. Nesta revisão sistemática, conseguimos identificar onze trabalhos relacionados à temática abordada neste trabalho, identificados na Seção 6.2. Finalmente, na Seção 6.3, descrevemos todas os estudos selecionados em maior grau de detalhe para que se torne possível realizarmos uma comparação com o modelo proposto na Seção 6.4.

6.1. Estratégia de Busca por Trabalhos Relacionados

A revisão de literatura foi formulada na forma de uma revisão sistemática, de acordo com as diretrizes formalizadas em (KITCHENHAM e CHARTERS, 2007), que provê um meio de identificar, avaliar e interpretar toda a pesquisa disponível e relevante a respeito de algum tema de pesquisa, tópico ou fenômeno de interesse. Estas trabalhos selecionadas para a revisão consistem de estudos primários.

Adicionalmente, a revisão sistemática também prevê uma estratégia de pesquisa padronizada e bem definida, de forma a ser replicável por terceiros e prover uma boa cobertura a respeito do tema a ser contemplado. Uma ressalva a ser feita é que, apesar da declaração explícita do método de busca realizado, a pesquisa feita a partir de bases de artigos digitais não possui garantias de que seu resultado seja replicável.

Esta estratégia de busca deve ser realizada em razão de uma questão de pesquisa a ser satisfeita, no caso específico deste trabalho, a pergunta é: *“Quais os principais estudos primários inseridos no contexto do problema do próximo release que incorporam métricas de risco ou incerteza?”*.

Esta pesquisa se justifica pela relativa pequena quantidade de estudos encontrados em (PITANGUEIRA, MACIEL e BARROS, 2015) que incorporam métricas de incerteza ou risco: (PAIXÃO e SOUZA, 2013a) e (PAIXÃO e SOUZA, 2013b). Sendo assim, a revisão procura responder se houve um crescimento de publicações neste tema em particular, e, sendo a resposta afirmativa, a revisão deve destacar as principais contribuições ao tema.

6.1.1. Estratégia de busca por estudos primários relacionados à proposta

A pesquisa consistiu primariamente na busca em bibliotecas digitais de artigos científicos, dentre elas, podemos destacar a IEEE Xplore³, a biblioteca digital da ACM⁴, Springer⁵ e Science Direct⁶. Além disto, foi utilizado o repositório de artigos de Engenharia de Software Baseada em Buscas (SBSE)⁷. Complementarmente, foram obtidos alguns artigos das seguintes conferências: *Symposium on Search-Based Software Engineering* (SSBSE) e Congresso Brasileiro de Software (CBSOft).

Dada a aparente escassez de artigos relevantes sobre o tema, a estratégia de pesquisa nestas bibliotecas digitais adotou os seguintes termos de busca:

- Next release problem AND risk
- Next release problem AND uncertainty
- Requirements engineering AND risk
- Requirements engineering AND uncertainty
- Requirements AND risk
- Requirements AND uncertainty

No caso do repositório SBSE, por se tratar de um repositório de propósito específico, foi utilizado apenas o termo de busca:

- Risk OR uncertainty

6.1.2. Critério de inclusão dos estudos primários

Para seleção dos estudos, foram concebidos os seguintes critérios de inclusão:

- Qualquer estudo que incluía uma nova função de avaliação para o problema do próximo release que incorpore métricas de risco, adicionalmente ou em substituição das funções já existentes de custo e satisfação;

³ <http://ieeexplore.ieee.org/Xplore/home.jsp>

⁴ <https://dl.acm.org>

⁵ <https://link.springer.com>

⁶ <http://www.sciencedirect.com>

⁷ http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/

- Qualquer estudo que, apesar de não incluir uma nova função de avaliação para o problema do próximo release, modifique as funções de custo ou satisfação de modo a realizar uma medição de risco para o problema;
- Qualquer estudo que realize medições de incerteza a respeito dos parâmetros de custo ou satisfação para o problema do próximo release.

6.2. Listagem de Trabalhos Relacionados

Os artigos selecionados para a revisão com base nos critérios de aceitação foram compilados na Tabela 14.

Tabela 14 - Lista de estudos primários incluídos.

Ano	Nome do artigo	Autores
2003	Quantitative Studies in Software Release Planning under Risk and Resource Constraints	(RUHE e GREER, 2003)
2009	A New Approach to the Software Release Planning	(COLARES <i>et al.</i> , 2009)
2012	A Multiobjective Optimization Approach to the Software Release Planning with Undefined Number of Releases and Interdependent Requirements	(BRASIL <i>et al.</i> , 2012)
2013	A Scenario-based Robust Model for the Next Release Problem	(PAIXÃO e SOUZA, 2013a)
2013	A Recoverable Robust Approach to the Next Release Problem	(PAIXÃO e SOUZA, 2013b)
2014	Robust Next Release Problem: Handling Uncertainty During Optimization	(LI <i>et al.</i> , 2014)
2015	Interactive Software Release Planning with Preferences Base	(DANTAS <i>et al.</i> , 2015)
2016	Risk-Aware Multi-stakeholder Next Release Planning Using Multi-objective Optimization	(PITANGUEIRA <i>et al.</i> , 2016)

2016	Uma Abordagem Multiobjetivo baseada em Otimização Interativa para o Planejamento de Releases	(SARAIVA <i>et al.</i> , 2016)
2016	The Value of Exact Analysis in Requirements Selection	(LI <i>et al.</i> , 2016)
2016	A Multi Objective Teacher-Learning-Artificial Bee Colony (MOTLABC) Optimization for Software Requirements Selection	(RANJITH e MARIMUTHU, 2016)

Conforme pode ser observado na tabela, estes onze estudos primários formam o conjunto dos principais trabalhos para o problema do próximo release que adotam de alguma maneira as métricas de risco ou incerteza. Dois trabalhos tangencialmente relacionados com o tema foram eliminados pelos critérios de seleção pois, apesar de representarem problemas de otimização combinatória dentro do contexto da Engenharia de Software, eles consistiam na busca por um subconjunto ótimo de componentes que minimizassem os riscos arquiteturais do sistema, portanto, eles não fazem parte da formulação de um problema para o próximo release (YANG, JONES e YANG, 2006) (LETIER, STEFAN e BARR, 2014).

A Figura 38 apresenta a distribuição dos estudos ao longo dos anos, o ano inicial de 2001 foi determinado por representar a data da primeira publicação a respeito do problema do próximo release. Podemos notar um primeiro esforço isolado em (RUHE e GREER, 2003) para introduzir uma medição de incerteza para a modelagem do problema. Posteriormente, outro avanço no tema somente foi observado no ano de 2009 (COLARES *et al.*, 2009). Desde então, este aspecto particular do problema permaneceu relativamente inexplorado até o ano de 2012, de onde se verifica uma maior regularidade nas publicações realizadas neste tema, culminando com 4 trabalhos em 2016.

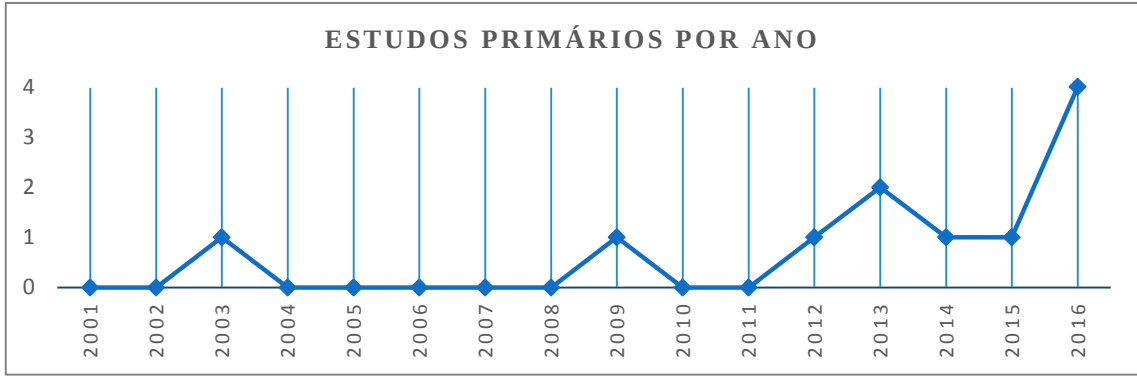


Figura 38 - Cronologia dos estudos primários selecionados.

6.3. Detalhamento de Trabalhos Relacionados

Nas seções seguintes, serão apresentados os aspectos relevantes de cada um dos onze estudos selecionados para revisão com enfoque no tema geral deste trabalho, seja uma modelagem de riscos ou alguma outra medida de incerteza para o problema.

6.3.1. Quantitative Studies in Software Release Planning under Risk and Resource Constraints

O **primeiro trabalho** relacionado ao tema (RUHE e GREER, 2003) utiliza uma estimativa de risco de forma a lidar com a incerteza intrínseca na implementação de um requisito. Este risco é dado como um valor agregado de quaisquer eventos que possam afetar negativamente o cronograma, custo ou qualidade final do projeto. Ele define um par $(r_i, R^k) \rightarrow [0,1)$, assumindo que o requisito r_i estará inserido no k -ésimo release R^k . Portanto, o problema é modelado não apenas observando um único conjunto de requisitos a serem entregues na próxima versão, mas como uma sucessão de ' k ' releases que implementarão subconjuntos distintos de requisitos. Este tipo de problema que admite mais de um release é denominado **Planejamento de Releases de Software** (*Software Release Planning*). Para cada release mapeado em incrementos disjuntos $Inc(k)$, é estipulado um valor-limite de risco máximo aceitável para aquela versão do sistema em $Risk^k$, de modo a evitar que muitos requisitos de alto risco estejam concentrados em uma única iteração, conforme definido na Equação 41.

$$\sum_{r(i) \in Inc(k)} risk(r_i, R^k) < Risk^k \quad (41)$$

Adicionalmente, a função de custo é modificada para uma função denominada “esforço”, representando uma medida mais abstrata de custo e tempo necessários para implementar os

requisitos selecionados. Finalmente, é importante notar que o trabalho adota uma modelagem mono-objetiva para o problema de planejamento de releases.

6.3.2. A New Approach to the Software Release Planning

O **segundo estudo** encontrado (COLARES *et al.*, 2009) continua a observar o problema de Planejamento de Releases de Software, mas, diferentemente do trabalho anterior, ele adota uma abordagem multi-objetiva para o problema. Porém, a função de esforço para implementar os requisitos ainda é restringida por um valor-limite de recursos disponíveis, de modo a garantir que o conjunto de requisitos não se torne caro o suficiente de modo a concentrar um conjunto de requisitos de alto custo em um único release, ou selecionar uma quantidade de requisitos muito grande, tornando-o incapaz de entregá-los dentro do cronograma estipulado para cada incremento individual.

A Equação 42 representa a modelagem da função de risco. Como pode ser observado, a equação considera que cada requisito r possui um risco associado, representado pelo termo $risk_r$. Este risco assume valores no intervalo entre zero e cinco, sendo este último o maior valor possível de risco. A variável x_i representa que o requisito r foi alocado para o i -ésimo release, portanto assumindo um valor no intervalo de 1 até $|K|$, sendo K o conjunto de releases para o problema. Esta função foi modelada de forma a alocar os requisitos de maior risco associado nos primeiros releases. Os autores justificam essa decisão pois, ao implementar um requisito de alto risco nas primeiras iterações, existe uma maior oportunidade de que este requisito seja monitorado e testado ao longo do desenvolvimento, diminuindo as possibilidades de ocorrência do risco.

$$minimizar \sum_{r \in R} (risk_r * x_i) \quad (42)$$

6.3.3. A Multiobjective Optimization Approach to the Software Release Planning with Undefined Number of Releases and Interdependent Requirements

O **terceiro estudo** encontrado (BRASIL *et al.*, 2012) também aborda uma modelagem multi-objetiva para um problema de planejamento de releases com um número indefinido de iterações. Os autores destacam a função abstrata de esforço para duas funções separadas de custo e tempo, nas quais ambas estão sujeitas a valores-limite pré-estabelecidos por iteração. A função de risco é semelhante ao trabalho anterior (COLARES *et al.*, 2009), visto na Seção 6.3.2. Mas, diferentemente, o trabalho introduz uma nova função de avaliação de prioridade. Esta nova função indica a satisfação dos clientes na implementação antecipada dos requisitos que

eles definiram como mais importantes, sendo a função maximizada conforme estes requisitos são alocados nos primeiros releases. Para impedir que todos os requisitos sejam alocados nos primeiros lançamentos do sistema, cada release está sujeito a duas funções de custo e tempo máximo permitido em cada iteração. Note que uma segunda função de satisfação ainda existe separadamente para sinalizar a importância destes requisitos independentemente de seu desenvolvimento antecipado, observando a satisfação obtida pelo projeto como um todo. É também importante destacar que as funções de satisfação, prioridade e risco estão sujeitas a duas funções de restrição de custo e tempo global do projeto, isto sinaliza que nem todos os requisitos disponíveis serão selecionados.

6.3.4. A Scenario-based Robust Model for the Next Release Problem

O **quarto estudo** (PAIXÃO e SOUZA, 2013a) admite uma implementação mono-objetiva e não adota a abordagem de iterações múltiplas. Ele modifica as funções para incorporar o conceito de incerteza nos valores estimados para os requisitos. Isto é feito a partir da simulação de diferentes cenários que representam a ocorrência de eventos distintos que podem modificar a importância dos requisitos. Assim, dado um conjunto de cenários $S = \{s_1, s_2, \dots, s_M\}$, a importância dos ' N ' requisitos são modificados para cada cenário da seguinte forma $s_s \subset S | s_s = \{v_1^s, v_2^s, \dots, v_N^s\}$, no qual v_r^s indica a importância do requisito r_i no cenário s_s . Cada cenário, por sua vez, possui uma probabilidade associada p_s , em que $\sum_{i=1}^{|S|} p_s = 1$. Sendo assim, a função objetiva de importância do requisito é modificada conforme demonstrado na Equação 43.

$$\text{maximizar} \sum_{i=1}^{|R|} \sum_{s=1}^{|S|} v_i^s p_s x_i \quad (43)$$

A função de custo também é modificada para refletir as incertezas do valor estimado de implementação para cada requisito, conforme definido na Equação 42. Adicionalmente ao valor de custo c_i associado ao requisito r_i , uma nova medida $\hat{c}_i$ é adotada para indicar a máxima variação esperada para o custo real de implementação do requisito, formando o intervalo inferior e superior tal que $c_i - \hat{c}_i \leq c_i \leq c_i + \hat{c}_i$. Adicionalmente, uma variável de controle Γ indica o número esperado de requisitos com custos incorretamente estimados, representando uma taxa de erro obtida durante a etapa de engenharia de requisitos, definindo assim o risco do modelo. Os autores detalham um exemplo para uma equipe de desenvolvimento que possui um

histórico de realizar uma estimaco incorreta de 30% dos requisitos, em um projeto que contm 50 requisitos, e, portanto, 15 destes requisitos tero valores de custo final diferente do estimado.

$$\sum_{i=1}^{|R|} c_i x_i + \max_{W \subseteq R, |W| \leq \Gamma} \sum_{i \in W} \hat{c}_i x_i \leq b \quad (44)$$

Note que a Equaco 44 adota uma abordagem pessimista ao descartar a possibilidade do custo real possuir um valor final inferior ao custo estimado. Como no existe uma maneira de saber antecipadamente a quantidade exata de requisitos que tiveram seus custos estimados de maneira incorreta, o modelo garante que, mesmo que a equipe de desenvolvimento cometa erro nos requisitos com maior variao final de custo real, a soluo ainda ser vlida, pois est variao tambm est subordinada ao oramento b . O subconjunto W agrupa os requisitos que provavelmente tero seus custos alterados, sendo subordinado ao valor mximo definido pela varivel de controle Γ .

6.3.5. A Recoverable Robust Approach to the Next Release Problem

O **quinto estudo** (PAIXO e SOUZA, 2013b) representa uma evoluo ao trabalho anterior dos mesmos autores, incorporando o conceito de um modelo recupervel para um problema do prximo release que incorpore estas mtricas de incertezas. Na prtica, este mtodo recupervel introduz um algoritmo capaz de perturbar a soluo encontrada caso ela seja considerada infactvel,. Os autores adotaram uma abordagem que, caso uma soluo no respeite as restries de custo de alocao, o algoritmo se torna capaz de modificar at k vezes a seleo de requisitos para aquela soluo de forma a torn-la factvel. Esta seleo  dada na forma de um subconjunto mnimo de requisitos que devem ser removidos da soluo. A Equaco 43, entretanto, permanece inalterada em relao ao estudo anterior. Portanto, apenas a Equaco 44  modificada conforme demonstrada na Equaco 45.

$$\max(basicCost, basicCost + uncertaintyLoss(\Gamma) - recoveryGain(k)) \leq b \quad (45)$$

As funes *basicCost*, *uncertaintyLoss* e *recoveryGain* so descritas nas Equaces 46, 47 e 48, respectivamente. Podemos observar que as duas primeiras funes compem os elementos dados na Equaco 44, enquanto que a nova funo *recoveryGain* aponta o conjunto mnimo, limitado por um valor mximo k de requisitos que podem ser removidos da soluo a fim de torn-la factvel.  interessante notar que, para um valor de k igual a zero, o problema retorna  formulao dada na Equaco 44. Sendo assim esta, nova funo da Equaco 45  considerada uma generalizao da funo anterior demonstrada na Equaco 42. De modo

similar, para um valor de Γ igual a 0%, as equações 44 e 45 também retornam para a formulação clássica de custo, dada pela Equação 44.

$$basicCost = \sum_{i=1}^{|R|} c_i x_i \quad (46)$$

$$uncertaintyLoss(\Gamma) = \max_{W \subseteq R, |W| \leq \Gamma} \sum_{i \in W} \hat{c}_i x_i \quad (47)$$

$$recoveryGain(k) = \min_{Y \subseteq R, |Y|=k} \sum_{i \in Y} c_i x_i \quad (48)$$

6.3.6. Robust Next Release Problem: Handling Uncertainty During Optimization

O **sexto estudo** (LI *et al.*, 2014) também representa uma medição de incerteza a respeito dos parâmetros de entrada para o problema do próximo release, com a diferença que adota uma abordagem multi-objetiva, diferentemente dos dois estudos anteriormente apresentados nas seções 6.3.4 e 6.3.5 (PAIXÃO e SOUZA, 2013a) (PAIXÃO e SOUZA, 2013b). Por adotar uma modelagem multi-objetiva, a função de custo não está mais subordinada ao valor-limite de orçamento. Adicionalmente, ao considerar também um intervalo de incerteza do benefício ocasionado pela seleção de um dado subconjunto de requisitos, que é semelhante ao intervalo de incerteza de custos observado no quarto estudo na Seção 6.3.4 (PAIXÃO e SOUZA, 2013a), isto contribui para que a solução admita uma região de tolerância conforme ilustrado na Figura 39.

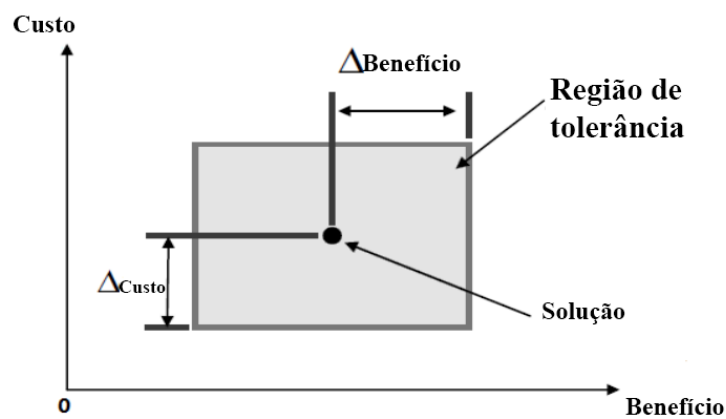


Figura 39 - Região de incerteza da solução (LI *et al.*, 2014).

Esta região de tolerância deve ser minimizada em uma função denominada *Size*, de forma a diminuir a incerteza a respeito dos parâmetros do problema. Portanto, o problema assume três funções a serem avaliadas, a função de custo esperado que deve ser minimizado, a

função de benefício a ser maximizada e a função de tamanho da região, que deve ser minimizada. Conforme apontado anteriormente, o problema não está mais restrito a um valor-limite de orçamento, porém, o problema pode ser restringido admitindo uma nova função de risco. Nesta nova função de risco, definida na Equação 49, a partir do subconjunto de requisitos selecionados $\vec{x}$, estima a probabilidade deste subconjunto estourar o orçamento esperado em θ por cento determinado pelo tomador de decisão. Este valor de porcentagem é fixado pelo tomador de decisão. Esta nova função de risco também deve ser minimizada para o problema e substitui a função *Size*. Ambas as funções são propostas no trabalho e apenas uma deve ser utilizada em detrimento da outra, sendo a primeira abordagem denominada de “Redução do Tamanho de Incerteza” e a segunda de “Redução do Risco de Falha”.

$$Risk(x) = Pro(ActualCost(\vec{x}) > \theta \cdot ExpectedCost(\vec{x})) \quad (49)$$

6.3.7. Interactive Software Release Planning with Preferences Base

Conforme pode ser observado no **sétimo estudo** (DANTAS *et al.*, 2015), a proposta retorna a uma abordagem mono-objetivo capaz de lidar com várias iterações distintas, alocando diferentes subconjuntos de requisitos em uma sucessão de releases. A modelagem do problema admite que, todo requisito r_i , além de um custo associado $cost_i$, também possui um risco associado $risk_i$. Além disso, os stakeholders interessados nos requisitos podem adotar várias condições de preferências na seleção dos requisitos que, caso não sejam atendidas, implica em uma penalidade que irá diminuir a avaliação da função de satisfação. Dentre estas preferências, um stakeholder pode determinar que um dado requisito deve ser alocado em um release específico ou que um par de requisitos deve ser alocado no mesmo release, por exemplo. Todas estas asserções de preferências são definidas em tuplas parametrizadas para o problema. A função de satisfação é descrita na Equação 50. Note que Z representa a quantidade de preferências existentes para o problema, enquanto que S representa o vetor de soluções. O valor de penalidade é calculado com base em cada uma destas preferências que não foram satisfeitas para a solução encontrada em relação a alocação dos requisitos em diferentes releases.

$$Fitness(S) = \begin{cases} score(S), & se Z = 0 \\ \frac{score(S)}{penalty(S)}, & se Z > 0 \end{cases} \quad (50)$$

A função $score(s)$ é representada na Equação 51. A variável $y_i \in \{0,1\}$ indica se o requisito foi alocado em algum release. A variável x_i indica o número do release no qual o requisito r_i foi alocado, enquanto que P é o número total de releases para o sistema. A função

$value_i$ representa a soma ponderada das preferências dos clientes para o requisito r_i . Uma informação a se destacar desta função é que o valor da função é maximizado conforme o requisito é alocado nos primeiros releases. Adicionalmente, o risco associado ao requisito também é minimizado caso o requisito seja alocado nas primeiras iterações do sistema, seguindo a mesma estratégia observada no segundo estudo da Seção 6.3.2 (COLARES *et al.*, 2009).

$$score(S) = \sum_{i=1}^{|R|} y_i \cdot (value_i \cdot (P - x_i + 1) - risk_i \cdot x_i) \quad (51)$$

6.3.8. Risk-Aware Multi-stakeholder Next Release Planning Using Multi-objective Optimization

No **oitavo estudo** (PITANGUEIRA *et al.*, 2016), os autores associam o risco à insatisfação dos clientes e realizam uma avaliação tri-objetiva, separadamente das funções de avaliação de custo e satisfação dado pela Equação 52. Conforme podemos observar na Equação 53, a medida de percepção de risco se dá por um desvio padrão das percepções de importância do requisito r_i em relação a cada cliente s_j . Estas medidas individuais de importância são comparadas em relação à percepção média de valor de todos os clientes para aquele requisito, dada pela função $avgvalue_i$, exibida em detalhes na Equação 54. Podemos notar que essa *média de valor do requisito* é semelhante ao cálculo de satisfação de requisitos em uma formulação tradicional do problema do próximo release, no qual temos o peso atribuído ao cliente em w_j multiplicado pela avaliação de importância do requisito r_i para o cliente s_j .

$$minimizar \sum_{i=1}^{|R|} risk_i \cdot x_i \quad (52)$$

$$risk_i = \sum_{j=1}^{|S|} w_j \cdot (value(r_i, s_j) - avgvalue_i)^2 / n \quad (53)$$

$$avgvalue_i = \sum_{j=1}^{|S|} w_j \cdot value(r_i, s_j) \quad (54)$$

Segundo os autores, esta medida de risco é dada desta forma pois, um requisito com uma avaliação inconsistente entre os stakeholders, no qual existe uma grande disparidade entre stakeholders que atribuem uma grande importância ao requisito, enquanto os demais atribuem uma importância muito baixa para este mesmo requisito r_i pode impactar na satisfação final do sistema. Caso este requisito seja priorizado em detrimento dos demais, um grupo significativo

de stakeholders poderiam ficar insatisfeitos e abandonar o apoio ao projeto. Em contrapartida, um requisito com uma avaliação de importância uniforme entre todos os interessados garante que, caso o requisito seja implementado, todos os stakeholders obteriam uma percepção de satisfação uniforme do sistema, minimizando o risco de insatisfação.

6.3.9. Uma Abordagem Multiobjetivo baseada em Otimização Interativa para o Planejamento de Releases

O **nono estudo** (SARAIVA *et al.*, 2016), também faz uso da função de avaliação de risco definida na Seção 6.3.2 (COLARES *et al.*, 2009), portanto, uma avaliação para múltiplos releases, além de admitir um enfoque multiobjetivo. A principal contribuição deste trabalho é o tratamento de restrições do problema, que inclui uma função de avaliação para priorizar os requisitos mais importantes para o tomador de decisão, semelhante ao trabalho descrito na Seção 6.3.7 (DANTAS *et al.*, 2015). O estudo também apresenta uma função de restrição que exige que cada release tenha pelo menos um requisito a ser implementado, além de uma função para garantir que as restrições de precedência e acoplamento entre requisitos sejam respeitadas.

6.3.10. The Value of Exact Analysis in Requirements Selection

O **décimo estudo** (LI *et al.*, 2016) representa um aperfeiçoamento das métricas apresentadas em seu estudo anterior apresentado na Seção 6.3.6 (LI *et al.*, 2014). A região de incerteza não é mais dada em razão de um intervalo de incerteza para os parâmetros de custo e satisfação, mas extrai estes valores a partir de um modelo probabilístico com base em uma distribuição de probabilidade fornecida pelos especialistas de domínio. Com base nesta distribuição, o estudo realiza um grande número de simulações baseadas em cenários em uma técnica denominada de Simulação de Monte-Carlo, de modo a obter os valores esperados para os parâmetros de custo e satisfação. A Equação 49 de risco permanece inalterada para este novo estudo, novamente representando uma estimativa da probabilidade do custo real de implementação ultrapassar o custo estimado em uma determinada porcentagem fixada pelo tomador de decisão.

6.3.11. A Multi Objective Teacher-Learning-Artificial Bee Colony (MOTLABC) Optimization for Software Requirements Selection

Finalmente, no **décimo-primeiro estudo** (RANJITH e MARIMUTHU, 2016), a proposta incorpora uma nova métrica de confiabilidade para cada requisito r_i , dada pela Equação 55. Diferentemente dos demais estudos, a medida de incerteza se dá em razão do tempo relacionados a falhas. Esta medida de confiabilidade passa a ser um novo objetivo a ser

maximizado, conforme dado na Equação 56. Os autores não esclareceram porque adotaram uma medida de maximização, sendo que, ao observarmos a Equação 55, um tempo alto de reparos para um requisito é admitido pela função avaliada como um fator positivo, sendo que é intuitivo assumirmos que um reparo ágil deveria ser considerado uma característica positiva do requisito. A diferença entre o tempo médio entre falhas (*mean time between failures* - *MTBF*) e o tempo médio para falhas (*mean time to failure* - *MTTF*) é que o primeiro se refere a itens que podem ser consertados e recolocados em operação, enquanto que o último se refere a um item que não pode ser reparado, sendo o item completamente perdido caso a falha ocorra.

$$\begin{aligned} \text{confiabilidade}_i = & \text{tempo médio entre falhas} + \text{tempo médio para falha} \\ & + \text{tempo médio para reparos} \end{aligned} \quad (55)$$

$$\text{maximizar } \sum_{i=1}^{|R|} \text{confiabilidade}_i \cdot x_i \quad (56)$$

É interessante notar que, além da confiabilidade, cada requisito r_i também possui um tempo de implementação associado, definido por tempo_i . Assim, o tempo de implementação é usado para definir uma nova função de avaliação para minimização de tempo. Portanto, neste estudo, o problema do próximo release é formulado com quatro funções de avaliação distintas.

6.4. Comparação com trabalhos relacionados

A Tabela 15 estabelece uma comparação da abordagem aqui proposta com os 11 estudos primários levantados como trabalhos relacionados ao tema. Com base nos dados coletados e representados na tabela, podemos observar um equilíbrio entre propostas para modelagem do problema do próximo release e do problema de planejamento de releases de software. Entretanto, podemos notar uma clara dominância de propostas que admitem uma formulação multi-objetiva para ambos os problemas, sendo que apenas quatro dos onze trabalhos relacionados ainda utilizam uma abordagem com um único objetivo.

tratamento de incerteza relativo ao custo do requisito. Já discutimos na seção de trabalhos relacionados as diversas maneiras que os autores destes trabalhos abordam incerteza, variando desde um parâmetro de estimativa de erro associada aos requisitos, passando por um intervalo de variação máxima de custo ou satisfação, até a forma mais recente que analisa um conjunto denso de cenários com diferentes variações para estes valores.

Dentre os parâmetros de entrada que fogem ao escopo tanto do valor de risco para requisitos quanto para os parâmetros de incerteza em relação aos valores de custo ou satisfação, podemos incluir a modelagem de preferência de alocação de requisitos, a medida de confiabilidade de requisitos e a medida de insatisfação de clientes. Apesar de não possuírem relação direta com a temática, eles influem indiretamente para o grau de sucesso do projeto de software, conforme detalhado anteriormente. É possível em alguma iteração futura da abordagem proposta a inclusão destes parâmetros, mas, visto que eles não possuem relação direta com uma modelagem de riscos para o problema do próximo release, não constituem uma prioridade para o escopo deste projeto.

Tabela 16 - Comparativo entre trabalhos relacionados com o modelo proposto em relação aos dados de entrada para o problema.

ABORDAGEM PROPOSTA	(RANJITH e MARIMUTHU, 2016)	(LI <i>et al.</i> , 2016)	(SARAIVA <i>et al.</i> , 2016)	(PITANGUEIRA <i>et al.</i> , 2016)	(DANTAS <i>et al.</i> , 2015)	(LI <i>et al.</i> , 2014)	(PAIXÃO e SOUZA, 2013b)	(PAIXÃO e SOUZA, 2013a)	(BRASIL <i>et al.</i> , 2012)	(COLARES <i>et al.</i> , 2009)	(RUHE e GREER, 2003)
Dados de entrada para o problema											
Riscos de requisitos	✓		✓		✓				✓	✓	✓
Probabilidade do risco	✓										
Severidade do risco	✓										
Técnicas de mitigação de riscos	✓										
Modelagem de incerteza no custo ou satisfação		✓				✓	✓	✓			

Cenários com diferentes variações no custo ou satisfação de requisitos				✓	✓					✓		
Variação de custo de requisito				✓	✓	✓				✓		
Variação do grau de satisfação do requisito						✓				✓		
Taxa de erro nos custos estimados dos requisitos				✓	✓							
Número máximo de requisitos que podem ser removidos do release					✓							
Probabilidade do custo real estourar o custo estimado						✓				✓		
Modelagem de preferência, prioridade de alocação ou acoplamento de requisitos			✓				✓		✓			
Medida de confiabilidade de requisitos com base na taxa de falhas e tempo de reparo											✓	
Métrica de insatisfação de clientes								✓				

A Tabela 17 dá continuidade à comparação, desta vez em relação à estratégia de resolução de problemas e as restrições utilizadas para o problema de planejamento de releases. Apesar do modelo proposto não se encaixar nesta categoria de problemas, é interessante observar as poucas semelhanças que existem nos trabalhos desta área e que cada nova contribuição ao longo dos anos concebe uma nova técnica de restrição do problema.

No tocante às estratégias de resolução do problema, dentre estes cinco trabalhos, iniciamos com a proposta (RUHE e GREER, 2003). Já observamos na seção anterior que se

trata de um projeto de planejamento de releases em que um valor máximo de risco permitido para cada release é pré-fixado. Diferentemente da abordagem proposta nesta dissertação, e, conforme podemos observar na Tabela 15, os riscos críticos não são priorizados e não impactam diretamente nas métricas de custo ou satisfação do problema. A abordagem proposta admite um impacto direto nas avaliações de custo e satisfação. No caso da função de custo, este impacto é tratado como um valor adicional acrescentado ao custo de implementação do requisito, isto é feito de modo a tornar a estimativa de custo mais realista, visto que já foi apontado que a subestimação de custos e o estouro de orçamento é um problema recorrente nos projetos de software (MASTICOLA, 2007). Em relação à avaliação de satisfação, o tratamento simultâneo de priorização de riscos críticos e requisitos de maior satisfação para os clientes reforça a idéia de que ambas devem ser levadas em igual consideração. Portanto, uma solução é considerada desejável caso os requisitos agreguem um alto valor para os clientes ao mesmo tempo que os riscos críticos sejam tratados antecipadamente. Caso o risco fosse definido como uma função separada, os tomadores de decisão poderiam simplesmente ignorar a medição de riscos que seria representada isoladamente e escolher os requisitos apenas com base nos indicadores de satisfação, o que não seria um comportamento desejável em uma abordagem baseada em riscos.

Observamos três trabalhos que utilizam uma função de risco semelhante para um problema de planejamento de releases multi-objetivo (COLARES *et al.*, 2009) (BRASIL *et al.*, 2012) (SARAIVA *et al.*, 2016). Novamente, esta função de risco é utilizada separadamente das medidas de custo ou satisfação, mas, de maneira semelhante à abordagem aqui proposta, os riscos críticos do projeto são priorizados, visto que os requisitos que possuem os riscos mais críticos possuem uma avaliação melhor, caso sejam alocados nos primeiros releases.

Finalmente, o trabalho que mais se assemelha com o modelo proposto é o estudo de Dantas *et al.* (DANTAS *et al.*, 2015), visto que a avaliação de risco está contida em uma função denominada *fitness*, que é análogo à função de satisfação do modelo. Nesta função, novamente, um requisito que possui um risco crítico melhora a função de *fitness* caso este requisito seja alocado nos primeiros releases. Portanto, neste trabalho, observamos a medida de risco com influência direta na avaliação de satisfação obtida na solução, do mesmo modo que os requisitos de maior severidade são priorizados na abordagem aqui proposta. Porém, diferentemente, neste estudo selecionado, não observamos uma relação direta entre a avaliação de custo e a formulação de riscos para o problema. Note que, na abordagem proposta, uma solução ideal admite que os riscos de maior severidade sejam priorizados, minimizando a probabilidade de ocorrência destes riscos com base em seu valor integrado à avaliação de custo. Esta

probabilidade, por sua vez, dimensiona o custo das técnicas de mitigação de riscos, em que as técnicas que são capazes de tratar uma grande gama de riscos distintos são consideradas mais desejáveis. Em contrapartida, uma técnica responsável por tratar apenas um risco representaria um tempo adicional gasto resolvendo uma parcela bastante reduzida de problemas no projeto.

Tabela 17 - Comparativo entre trabalhos relacionados com o modelo proposto em relação às estratégias de resolução do problema e as restrições impostas aos trabalhos para o problema de planejamento de releases de software.

ABORDAGEM PROPOSTA	(RANJITH e MARIMUTHU, 2016)	(LI <i>et al.</i> , 2016)	(SARAIVA <i>et al.</i> , 2016)	(PITANGUEIRA <i>et al.</i> , 2016)	(DANTAS <i>et al.</i> , 2015)	(LI <i>et al.</i> , 2014)	(PAIXÃO e SOUZA, 2013b)	(PAIXÃO e SOUZA, 2013a)	(BRASIL <i>et al.</i> , 2012)	(COLARES <i>et al.</i> , 2009)	(RUHE e GREER, 2003)
Estratégias de resolução do problema											
Abordagem prioriza riscos críticos	✓		✓		✓				✓	✓	
Riscos impactam diretamente nas métricas de satisfação	✓				✓						
Riscos impactam diretamente nas métricas de custo	✓										
Restrições para o problema de planejamento de releases											
Limite de risco máximo admitido por release											✓
Limite de recursos gastos a cada iteração										✓	
Limite de tempo por release									✓		
Limite de custo para o release		✓	✓						✓		

Cada release deve possuir pelo menos um requisito									✓			
---	--	--	--	--	--	--	--	--	---	--	--	--

Finalmente, a Tabela 18 condensa as informações a respeito das bases de dados e algoritmos utilizados nos experimentos. Os conjuntos de dados utilizados nos experimentos inicialmente utilizavam exclusivamente dados gerados artificialmente. Posteriormente, observamos uma dominância de casos de uso com base em dados de projetos reais. Ocorreram duas situações (PAIXÃO e SOUZA, 2013a) (PAIXÃO e SOUZA, 2013b) em que foram utilizados tanto dados reais quanto artificiais. Segundo os autores, os dados artificiais foram utilizados para criar um cenário com uma quantidade maior de requisitos, que não se encontrava disponível na base de dados real.

Dentre os algoritmos utilizados, notamos uma clara dominância do NSGA-II nas propostas que admitem uma formulação multi-objetiva. O MOCell aparece com o segundo maior número de utilizações. Alguns outros algoritmos de otimização multi-objetiva emergiram nesta seleção. Em outros casos, o algoritmo aleatório foi utilizado conjuntamente com as metaheurísticas multi-objetivas a fim de validar os resultados através de um “teste de sanidade” (HARMAN *et al.*, 2012). Além dos dados expostos na Tabela 6, podemos também verificar que, adicionalmente aos dados obtidos através dos algoritmos, em (COLARES *et al.*, 2009), os autores realizaram uma comparação dos resultados obtidos com um pequeno conjunto de soluções fornecidas por especialistas de domínio. Por sua vez, em (PITANGUEIRA *et al.*, 2016), os autores estabeleceram uma comparação dos resultados obtidos pelo NSGA-II com dois algoritmos de satisfatibilidade capazes de obter uma melhor aproximação do conjunto ótimo de Pareto. Apesar destes algoritmos serem capazes de retornar um resultado superior ao NSGA-II, eles exigem uma configuração a respeito do custo mínimo e máximo permitido por release, sendo reinicializados diversas vezes com diferentes parâmetros de restrição, realizando a cada iteração uma busca em um espaço de soluções reduzido. Além disso, o algoritmo pode demorar vários dias para encerrar sua execução completa, ou, caso as condições de restrição fixadas não seja satisfazíveis, o algoritmo pode incorrer em um loop infinito de execução, explorando todas as soluções possíveis, mas sem nunca encontrar uma que atenda aos parâmetros configurados.

Tabela 18 - Comparativo entre trabalhos relacionados com o modelo proposto em relação às bases de dados e algoritmos utilizados nos experimentos.

ABORDAGEM PROPOSTA	(RANJITH e MARIMUTHU, 2016)	(LI <i>et al.</i> , 2016)	(SARAIVA <i>et al.</i> , 2016)	(PITANGUEIRA <i>et al.</i> , 2016)	(DANTAS <i>et al.</i> , 2015)	(LI <i>et al.</i> , 2014)	(PAIXÃO e SOUZA, 2013b)	(PAIXÃO e SOUZA, 2013a)	(BRASIL <i>et al.</i> , 2012)	(COLARES <i>et al.</i> , 2009)	(RUHE e GREER, 2003)
Dados utilizados no experimento											
Dados Artificiais	✓						✓	✓	✓	✓	✓
Dados Reais		✓	✓	✓	✓	✓	✓	✓			
Algoritmos de busca avaliados											
Algoritmo Genético							✓	✓			✓
Algoritmo Genético Interativo					✓						
Busca Aleatória			✓							✓	✓
Busca Exaustiva											✓
MOCeII			✓						✓		✓
MOTLABC											✓
NSGA-II		✓	✓	✓		✓			✓	✓	✓
NSGDP		✓									
Simulated Annealing								✓			
SPEA2											✓

6.5. Considerações finais

Este capítulo realizou uma revisão de literatura no formato de revisão sistemática cujas fontes em sua maioria foram oriundas de bibliotecas digitais de artigos, com um conjunto reduzido de artigos coletados diretamente de eventos ou congressos realizados na área. Com base nos 11 artigos encontrados, conseguimos observar um maior crescimento em publicações a respeito de métricas de risco ou incerteza para o problema do próximo release ou planejamento de releases a partir do ano de 2012. A partir de uma discussão prolongada a respeito das 11 contribuições para este tema, os resultados foram compilados em uma série de

tabelas, que revelou uma predominância de trabalhos de próximo release multi-objetivo. Um equilíbrio de publicações para ambas as classes de problemas de próximo release ou planejamento de releases foi identificado, apontando que ambos os problemas permanecem como um tema relevante para a comunidade acadêmica. Uma grande diversidade de métricas de risco, incerteza a respeito dos parâmetros de custo ou satisfação de requisitos, além de outros trabalhos que levam outros fatores como o tempo necessário para corrigir possíveis problemas sofridos durante a implementação do subconjunto de requisitos candidatos que foram selecionados para entrega. Em relação à priorização de riscos críticos, esta estratégia se encontra bem consolidada entre as abordagens propostas e portanto, é uma característica desejável para uma proposta que incorpore riscos de software. A grande maioria das abordagens define uma função objetivo separada para a avaliação de riscos, enquanto que apenas um único trabalho incorpora esta avaliação à função de satisfação (DANTAS *et al.*, 2015). Entretanto, apenas o modelo aqui proposto incorpora a avaliação de risco em ambas as funções de custo e satisfação, além de considerar informações relativas à probabilidade e severidade de riscos, além das técnicas de mitigação de riscos. Em relação aos experimentos realizados, observamos uma predominância de uso de bases de dados declaradamente reais. Além disso, o NSGA-II é o algoritmo mais utilizado nas propostas, naturalmente por se tratar de um algoritmo bastante consolidado na literatura.

7. CONCLUSÃO

Com base em todo o trabalho apresentado até aqui, dando um destaque especial para os resultados apresentados, assim como os principais trabalhos relacionados apresentados em detalhes, podemos verificar que o trabalho satisfaz os objetivos propostos. O problema do próximo release neste trabalho foi formulado como um problema multi-objetivo, com duas funções objetivo a serem otimizadas: custo e satisfação. Estas funções já foram pré-estabelecidas na formulação original do problema, mas foram extensivamente modificadas de forma a refletir o relacionamento entre os requisitos candidatos e seus potenciais riscos associados. Os riscos, por sua vez, foram associados a uma série de técnicas de mitigação de riscos, de forma a refletir o impacto no esforço necessário em tratar tais riscos. A medida de esforço, neste caso, foi dado como um custo adicional ao custo de desenvolvimento dos requisitos. Deste modo, foi cumprido a meta de manter a avaliação de custo para os requisitos candidatos, dado que o estouro de orçamento é um ponto de falha muito recorrente em projetos de software reais (MASTICOLA, 2007). A avaliação de satisfação de clientes também foi mantida neste trabalho, portanto, o objetivo de manter o modelo consolidado de custo-importância (ou custo-satisfação) foi atendido.

Dentre os trabalhos relacionados, apesar da aparente escassez de propostas para este tema específico, conseguimos listar onze trabalhos que de alguma maneira se relacionam com a temática do trabalho. Destes onze trabalhos, cinco deles fazem alusão a alguma medição de risco incorporado ao problema de planejamento de releases. Infelizmente, nenhum trabalho para o problema do próximo release que faça uso de riscos foi encontrado. Mas, de qualquer modo, as boas práticas encontradas nestes cinco trabalhos também foram atendidas pelo presente trabalho. A principal prática se trata da priorização dos requisitos que possuem uma avaliação de risco mais crítica, em que quatro destes trabalhos criaram uma terceira função de objetivo exclusiva para tal fim. Um dos trabalhos (DANTAS *et al.*, 2015), que mais se aproximou de nossa proposta, incorporou tal avaliação dentro da função denominada “fitness”, que bastante se assemelha à função clássica de satisfação para um problema do próximo release. Porém, diferentemente da abordagem aqui proposta, estes modelos adotam um valor único de risco, sem fazer qualquer menção a sua probabilidade, severidade ou como estes riscos devem ser resolvidos por alguma técnica de mitigação de riscos. Adicionalmente, o modelo proposto não avalia apenas como o tratamento de riscos críticos melhora a satisfação obtida pelo release, mas também como estes riscos alteram a avaliação final de custo. Como resultado, temos um modelo

para o próximo release baseado em riscos que prioriza o tratamento de riscos críticos, mas ao mesmo tempo busca minimizar a probabilidade do evento de risco e o custo necessário para o tratamento destes riscos.

Dos outros seis trabalhos relacionados restantes, observamos que alguns deles lidam com a incerteza dos parâmetros de custo de implementação de requisitos. Apesar da proposta não lidar diretamente com este tipo de parâmetro, a alteração da avaliação final de custo com base no custo de tratamento de riscos atende parcialmente este tipo de incerteza intrínseca nos valores de entrada para o problema, buscando uma avaliação de custo mais realista para os subconjuntos de requisitos selecionados obtidos como solução do problema.

Finalmente, observamos a realização de dois experimentos, o primeiro, apesar de um experimento bastante reduzido e controlado, foi útil para observar o comportamento das funções de custo e satisfação e como as equações modeladas se comportam em comparação com o que foi descrito no modelo. O experimento se mostrou bem sucedido, demonstrando propriedades interessantes como a maximização da avaliação final de satisfação com base na seleção de requisitos com riscos críticos, ao mesmo tempo que a seleção de requisitos de riscos menos críticos diminui a avaliação final de satisfação em comparação com o valor de satisfação sem considerar riscos. Adicionalmente, observamos que a seleção de requisitos com uma grande quantidade de riscos compartilhados diminui a avaliação individual de custo para cada requisito. Isto se mostrou um efeito desejável, pois, um requisito com riscos exclusivos acarretaria em um tempo e esforço maior exigido para implementação satisfatória deste requisito. Este esforço exigido em tratar riscos que afetam um número reduzido de requisitos é refletido no modelo como uma avaliação individual de custo maior para estes requisitos.

O segundo experimento, por sua vez, validou o modelo do ponto de vista de um problema do próximo release multi-objetivo, utilizando três metaheurística multi-objetivo: NSGA-II, SPEA2 e MOCell. Estes algoritmos, conforme observamos nos trabalhos relacionados e em uma revisão sistemática que mapeou o histórico de contribuições para o problema do próximo release (PITANGUEIRA, MACIEL e BARROS, 2015), representam alguns dos algoritmos mais utilizados e que compõem o estado da arte dos algoritmos de busca multi-objetivo. A existência do NSGA-III é reconhecida (MKAOUER *et al.*, 2014), mas dada a ausência de trabalhos relacionados ao tema que utilizam esta metaheurística, normalmente concentrados em utilizar o NSGA-II, exigiu que a proposta também utilizasse esta versão consolidada do algoritmo. O melhor cenário de adoção do NSGA-III seria utilizá-lo conjuntamente com o já consolidado NSGA-II para permitir a comparação de resultados entre

estas metaheurísticas. Porém, em favor de uma maior diversidade de algoritmos de busca, afim de obter uma maior diferenciação de resultados, foi optado utilizar o SPEA2 e MOCell para realizarmos tal comparação de resultados.

Duas bases de dados semi-reais foram utilizadas por estes algoritmos, fornecendo soluções bastante próximas entre si e muito superiores a uma solução obtida de maneira aleatória. Estes algoritmos, no experimento menor, obtiveram resultados bastante próximos de valores ótimos utilizando apenas uma fração do tempo exigido para buscar tal solução ótima através de um algoritmo de busca exaustiva. No segundo experimento, observamos que as características do MOCell, um algoritmo de busca celular que utiliza o conceito de vizinhança de soluções, utilizou esta característica de maneira muito vantajosa ao explorar uma curva de soluções densamente agrupada, obtendo soluções melhores que o NSGA-II e SPEA2.

Portanto, com os resultados do segundo experimento utilizando estas duas bases de dados, foi possível validar a proposta do ponto de vista de um problema do próximo release multi-objetivo usando dois casos semi-reais. Seria mais desejável utilizar uma base de dados contendo todos os dados de entrada reais para o problema mas, infelizmente, uma busca realizada por trabalhos relacionados ao tema de gerenciamento de riscos não foi possível retornar uma base de dados que apresentasse tais informações disponíveis para utilização. A melhor aproximação obtida para este fim foi uma pesquisa realizada no Brasil em cinco empresas de pequeno porte que realizou um gerenciamento de riscos (NEVES *et al.*, 2014). Apesar da compilação de dados relativos a riscos, probabilidade de riscos, severidade de riscos e técnicas de mitigação de riscos, estes dados não se encontravam acessíveis para utilização, além de exibir uma desvantagem em também não apresentar o mapeamento entre requisitos e seus riscos associados, assim como também não foi encontrado um mapeamento entre riscos e suas respectivas técnicas de mitigação de riscos. Apesar destas limitações, a existência de mais de um projeto real que realiza de fato um gerenciamento de riscos em uma empresa de pequeno porte capaz de prover tais dados nos mostra indícios que tal modelagem para utilização no problema do próximo release é factível.

Adicionalmente ao uso de uma base de dados completamente real, também seria desejável obter dados com os mais distintos cenários quanto fosse possível. Dentre estes cenários, poderíamos obter quantidades bastante distintas de requisitos, riscos, técnicas e clientes. Deste modo, poderíamos observar o comportamento do modelo em, por exemplo, um cenário com poucos requisitos, mas uma grande quantidade de riscos e técnicas. Este cenário seria uma ocasião oportuna para melhor investigarmos o comportamento dos riscos

compartilhados e exclusivos entre os requisitos. Outro cenário interessante seria observarmos uma grande quantidade de stakeholders. Neste cenário, como o modelo proposto se comportaria com uma quantidade variável de riscos? Vimos que os requisitos com riscos mais críticos aumentam significativamente a avaliação final de satisfação. Este cenário continuaria coerente com uma grande quantidade de clientes a serem satisfeitos?

7.1. Trabalhos Futuros

Dentre os possíveis desdobramentos deste trabalho, podemos observar uma série de possíveis campos a serem explorados, incluindo:

- Utilização de lógica *fuzzy* para representar os valores de risco, exercendo um maior acoplamento entre os valores nominais dos graus de probabilidade e severidade de risco com sua implementação formal nas funções objetivas. Diferentemente da proposta atual, em vez de utilizar valores discretos fixados dentro de um intervalo, utilizar então uma função *fuzzy* em que um dado grau de risco pode assumir uma gama de valores distintos;
- Reconstrução das funções objetivas do ponto de vista de um problema de Planejamento de Releases, incorporando uma abordagem com múltiplas incrementos;
- Incluir restrições nos *datasets* utilizados. De maneira notável podemos citar a definição de pré-requisitos entre os requisitos candidatos do problema, conforme descrito no capítulo 2 deste trabalho;
- Realizar novos experimentos utilizando outras metaheurísticas disponíveis no jMetal, particularmente o NSGA-III apresentado anteriormente;
- Realizar novos experimentos utilizando diferentes bases de dados reais. Conforme discutido anteriormente, dada a escassez de bases de dados prontas para serem utilizadas, seria necessário aguardar alguma publicação futura que fosse capaz de fornecer tais dados ou, como trabalho futuro, realizar alguma parceria com uma ou mais empresas que permitissem fornecer tais dados para uso com base em seus projetos de software. Também é interessante notar que o jMetal foi projetado de modo que ele possa ser facilmente estendido para novas funcionalidades. Além disso, o framework está sob uma licença de uso LGPL sinalizando que, apesar que os componentes do jMetal devam continuar seguindo as diretrizes desta licença, quaisquer componentes de software adicionais que façam uso do jMetal estão livres para serem submetidos a qualquer outra

licença de uso, permitindo a criação de uma ferramenta proprietária que seja de interesse da empresa parceira em potencial. O caminho oposto também é possível, possibilitando também a criação de uma ferramenta de código livre caso seja de interesse de todas as partes envolvidas em um possível desdobramento deste trabalho;

- Incorporar medidas de incerteza a respeito da valoração de custo de requisitos e da satisfação de clientes ou, de maneira inédita, também admitir incerteza a respeito dos graus de probabilidade e severidade de riscos, assim como também a respeito de custo de utilização das técnicas de mitigação de risco. Este desdobramento teria por referência principal a proposta de Li *et al.* (LI *et al.*, 2016), que realizou medidas de incerteza a respeito dos custos e da satisfação percebida dos requisitos;
- Incorporar outras medidas que podem impactar o custo ou satisfação de requisitos além de riscos. Dentre as medidas candidatas, poderia ser observado o tempo exigido para implementação de requisitos. Deste modo, seria permitido observar como a antecipação ou atraso no cronograma do release com base na seleção de requisitos candidatos afetaria o custo e satisfação final dos requisitos entregues.

REFERÊNCIAS BIBLIOGRÁFICAS

- ALAM, I. Role of Software Metrics in identifying the risk of project. **International Journal of Advancement in Engineering Technology, Management & Applied Science**, v. 1, n. 1, p. 7, jun. 2014.
- ARNUPHAPTRAIRONG, T. **Top Ten Lists of Software Project Risks**: Evidence from the Literature Survey. International MultiConference of Engineers and Computer Scientists (IMECS 2011). Hong Kong: [s.n.]. 2011.
- BAGNALL, A. J.; RAYWARD-SMITH, V. J.; I. M. WHITTLEY. The next release problem. **Information and Software Technology**, v. 43, n. 14, p. 883-890, 2001.
- BAKER, P. et al. **Search based approaches to component selection and prioritization for the next release problem**. Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference. [S.l.]: [s.n.]. 2006.
- BANNERMAN, P. L. Risk and risk management in software projects: A reassessment. **Journal of Systems and Software**, v. 81, n. 12, p. 2118-2133, dez. 2008.
- BERANDER, P.; ANDREWS, A. Requirements Prioritization. In: AURUM, A.; WOHLIN, C. **Engineering and Managing Software Requirements**. Berlin: Springer, 2005. Cap. 4, p. 69-94.
- BOEHM, B. W. Software risk management: principles and practices. **IEEE Software**, v. 8, n. 1, p. 32-41, jan. 1991.
- BRASIL, M. M. A. et al. A Multiobjective Optimization Approach to the Software Release Planning with Undefined Number of Releases and Interdependent Requirements. In: ZHANG, R., et al. **Enterprise Information Systems: 13th International Conference, ICEIS 2011**, Beijing, China, June 8-11, 2011, Revised Selected Papers. Berlin, Heidelberg: Springer, 2012. p. 300-314.
- CAI, X.; WEI, O.; HUANG, Z. Evolutionary approaches for multi-objective next release problem. **Computing and Informatics**, v. 31, n. 4, p. 847-875, 2012.
- CAO, Y. Hypervolume Indicator. **MATLAB Central**, 2008. Disponível em: <<https://www.mathworks.com/matlabcentral/fileexchange/19651-hypervolume-indicator>>. Acesso em: 3 abr. 2017.
- CHUDASAMA, C.; SHAH, S. M.; PANCHAL, M. **Comparison of Parents Selection Methods of Genetic Algorithm for TSP**. International Journal of Computer Applications (IJCA). [S.l.]: [s.n.]. 2011. p. 3.

- CLELAND-HUANG, J. et al. **Goal-Centric Traceability for Managing Non-Functional Requirements**. Saint Louis, MO, Estados Unidos: IEEE. 2005. p. 10.
- COLANZI, T. E. et al. Search Based Software Engineering: Review and analysis of the field in Brazil. **The Journal of Systems and Software**, v. 86, n. 4, p. 970-984, ago. 2012.
- COLARES, F. et al. **A new approach to the software release planning**. Software Engineering, 2009. SBES '09. XXIII Brazilian Symposium. [S.l.]: [s.n.]. 2009. p. 207-215.
- DANTAS, A. et al. **Interactive Software Release Planning with Preferences Base**. Search-Based Software Engineering 7th International Symposium (SSBSE 2015). Bergamo, Itália: Springer. 2015. p. 341-346.
- DEB, K. **Multi-objective Optimization Using Evolutionary Algorithms**. 1. ed. [S.l.]: Wiley, 2001.
- DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. **Trans. Evol. Comp**, v. 6, p. 182-197, abr. 2002.
- DEMARCO, T.; LISTER, T. Risk Management during Requirements. **IEEE Software**, v. 20, n. 5, p. 99-101, out. 2003.
- DORRONSORO, B.; ALBA, E. **Cellular Genetic Algorithms**. [S.l.]: Springer, v. 42, 2008.
- DURILLO, J. J. et al. **A study of the multi-objective next release problem**. Proceedings of the 2009 1st International Symposium on Search Based Software Engineering, SSBSE '09. Washington, DC, USA: IEEE Computer Society. 2009. p. 49-58.
- DURILLO, J. J. et al. A study of the bi-objective next release problem. **Empirical Software Engineering**, v. 16, n. 1, p. 29-60, 2011.
- DURILLO, J. J.; NEBRO, A. J. jmetal: A java framework for multiobjective optimization. **Adv. Eng. Softw.**, v. 42, p. 760-771, out. 2011.
- EDC. GA Roulette wheel selection. **Newcastle University**, 2015. Disponível em: <<http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php>>. Acesso em: dez. 2015.
- FAY, M. P.; PROSCHAN, M. A. Wilcoxon-Mann-Whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. **Statistics Surveys**, v. 4, p. 1-39, jul. 2009. ISSN 1935-7516.
- FEATHER, M. S.; CORNFORD, S. L. Quantitative risk-based requirements reasoning. **Requir. Eng.**, v. 8, p. 248-265, nov. 2003.
- FELDERER, M.; AUER, F.; BERGSMANN, J. **Risk Management During Software Development: Results of a Survey in Software Houses from Germany, Austria and Switzerland**. International Workshop on Risk Assessment and Risk-driven Testing. Innsbruck: Springer. 2016. p. 143-155.

- FERNANDES, J. M.; MACHADO, R. J. Requirements Negotiation and Prioritisation. In: FERNANDES, J. M.; MACHADO, R. J. **Requirements in Engineering Projects**. [S.l.]: Springer, 2016. Cap. 6, p. 119-134.
- FOGELSTRÖM, N. D. et al. When Product Managers Gamble with Requirements: Attitudes to Value and Risk. In: GLINZ, M.; HEYMANS, P. **Requirements Engineering: Foundation for Software Quality**. Berlim: Springer, 2009. p. 1-15.
- HAN, W.-M. Validating differential relationships between risk categories and project performance as perceived by managers. **Empirical Software Engineering**, v. 19, n. 6, p. 1956-1966, dez. 2014.
- HARMAN, M. et al. Empirical software engineering and verification. In: _____ **Search Based Software Engineering: Techniques, Taxonomy, Tutorial**. Berlin, Heidelberg: Springer-Verlag, 2012. p. 1-59.
- HOLLAND, J. H. **Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence**. [S.l.]: MIT Press, 1992.
- HUMA HAYAT KHAN, M.; NAZ'RI BIN MAHRIN, S. B. C. Factors Generating Risks during Requirement Engineering Process in Global Software Development Environment. **International Journal of Digital Information and Wireless Communications (IJDIWC)**, v. 1, n. 4, p. 63-78, 2014.
- ISLAM, S.; HOUMB, S. H. **Integrating risk management activities into requirements engineering**. Research Challenges in Information Science (RCIS), 2010 Fourth International Conference on. Nice, France,: [s.n.]. 2010. p. 299-310.
- KAJIKO-MATTSSON, M.; LUNDHOLM, J.; NORRBY, J. **Industrial Opinion on the Effectiveness of Risk Management Methods**. Computer Software and Applications Conference (COMPSAC '09). Seattle: IEEE. 2009. p. 642-643.
- KARIM, M. R.; RUHE, G. **Bi-objective Genetic Search for Release Planning in Support of Themes**. International Symposium on Search Based Software Engineering (SSBSE 2014). Fortaleza, Brasil: [s.n.]. 2014. p. 123-137.
- KARLSSON, J.; RYAN, K. A Cost-Value Approach for Prioritizing Requirements. **IEEE Software**, v. 14, n. 5, p. 67-74, set. 1997.
- KITCHENHAM, B.; CHARTERS, S. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. University of Durham. Durham, Reino Unido, p. 65. 2007.
- LARMAN, C.; BASILI, V. R. Iterative and incremental developments. a brief history. **Computer**, v. 36, n. 6, p. 10, jun. 2003.

- LETIER, E.; STEFAN, D.; BARR, E. T. **Uncertainty, risk, and information value in software requirements and architecture**. Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). New York, NY, USA: ACM. 2014. p. 883-894.
- LI, L. et al. **Robust next release problem**: Handling uncertainty during optimization. Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. New York, NY, USA: ACM. 2014. p. 1247-1254.
- LI, L. et al. The Value of Exact Analysis in Requirements Selection. **IEEE Transactions on Software Engineering**, n. 99, p. 18, out. 2016.
- MASTICOLA, S. P. **A simple estimate of the cost of software project failures and the breakeven effectiveness of project risk management**. Proceedings of the First International Workshop on The Economics of Software and Computation, ESC '07. Washington, DC, USA: IEEE Computer Society. 2007. p. 6--.
- MKAOUER, W. et al. **High dimensional search-based software engineering**: Finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III. Genetic and Evolutionary Conference (GECCO '14). Nova York: ACM. 2014. p. 1263-1270.
- NEBRO, A. J. et al. MOCeII: A Cellular Genetic Algorithm for Multiobjective Optimization. **International Journal of Intelligent Systems**, v. 24, n. 7, p. 726-746, jul. 2009.
- NEBRO, A. J.; DURILLO, J. J. **jMetal 4.5 User Manual**. [S.l.], p. 88. 2014.
- NEVES, S. M. et al. Risk management in software projects through Knowledge Management techniques: Cases in Brazilian Incubated Technology-Based Firms. **International Journal of Project Management**, v. 32, n. 1, p. 125-138, 2014.
- NORDIN, A. et al. **Requirements elicitation and analysis**: Towards the automation of software project risk management. Software Engineering Conference (MySEC). Langkawi, Malásia: IEEE. 2014. p. 78-83.
- OBITKO, M. Selection Operators. **Introduction to Genetic Algorithms**, 1998. Disponível em: <<http://www.obitko.com/tutorials/genetic-algorithms/selection.php>>. Acesso em: dez. 2015.
- PAIXÃO, M.; SOUZA, J. **A scenario-based robust model for the next release problem**. Proceedings of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference. [S.l.]: [s.n.]. 2013a. p. 1469-1476.
- PAIXÃO, M.; SOUZA, J. **A recoverable robust approach for the next release problem**. Proceedings of the 5th SSBSE. [S.l.]: [s.n.]. 2013b.
- PITANGUEIRA, A. M. et al. **Risk-Aware Multi-stakeholder Next Release Planning Using Multi-objective Optimization**. Proceedings of the 22nd International Working Conference on

- Requirements Engineering: Foundation for Software Quality (REFSQ 2016). Gothenburg, Suécia: [s.n.]. 2016. p. 3-18.
- PITANGUEIRA, A. M.; MACIEL, R. S. P.; BARROS, M. Software requirements selection and prioritization using SBSE approaches: A systematic review and mapping of the literature. **Journal of Systems and Software**, v. 103, p. 267-280, maio 2015. ISSN 0164-1212.
- RAJAGOPAL, P. et al. **A new approach for software requirements elicitation**. 6th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. [S.l.]: [s.n.]. 2005.
- RANJITH, N.; MARIMUTHU, A. A Multi Objective Teacher-Learning-Artificial Bee Colony (MOTLABC) Optimization for Software Requirements Selection. **Indian Journal of Science and Technology**, v. 9, n. 34, p. 9, set. 2016.
- RECHENBERG, I. Evolution strategy: Nature's way of optimization. In: BERGMANN, H. W. **Optimization: Methods and applications, possibilities and limitations**. Berlim: Springer, 1989. Cap. 6, p. 106-126.
- RUHE, G.; GREER, D. **Quantitative studies in software release planning under risk and resource constraints**. Proceedings of the 2003 International Symposium on Empirical Software Engineering. Washington, DC, USA: IEEE Computer Society. 2003. p. 262-.
- RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3. ed. [S.l.]: Pearson, 2009.
- SÁNCHEZ, P. et al. Introducing Safety Requirements Traceability Support in Model-Driven Development of Robotic Applications. **IEEE Transactions on Computers**, v. 60, n. 8, p. 1059-1071, 2011.
- SARAIVA, R. et al. **Uma Abordagem Multiobjetivo baseada em Otimização Iterativa para o Planejamento de Releases**. VII Workshop de Engenharia de Software Baseada em Busca (WESB 2016). Maringá, PR, Brasil: CBSOft. 2016. p. 31-40.
- SCACCHI, W. Process Models in Software Engineering. In: _____ **Encyclopedia of Software Engineering**. [S.l.]: [s.n.], 2002. p. 24.
- SILVERMAN, B. W. **Density Estimation for Statistic and Data Analysis**. [S.l.]: CRC press, v. 26, 1986.
- SIVANANDAM, S. N.; DEEPA, S. N. **Introduction to Genetic Algorithms**. 1. ed. Berlim: Springer, 2008.
- SRINIVAS, N.; DEB, K. Multiobjective function optimization using nondominated sorting genetic algorithms. **Evol. Comput.**, v. 2, n. 3, p. 221-248, Outono 1995.

- VERNER, J.; SAMPSON, J.; CERPA, N. **What factors lead to software project failure?** Research Challenges in Information Science, 2008. RCIS 2008. Second International Conference. [S.l.]: [s.n.]. Junho 2008. p. 71-80.
- WERNER, F. **Genetic Algorithms for Shop Scheduling Problems: A Survey.** Universidade de Otto-von-Guericke. Magdeburg, p. 31. 2011.
- YANG, L.; JONES, B.; YANG, S. Genetic algorithm based software integration with minimum software risk. **Information and Software Technology**, v. 48, n. 3, p. 133-141, 2006.
- YAO, X. **Optimization by Genetic Annealing.** Second Australian Conf. on Neural Networks. Sydney: [s.n.]. 1991. p. 94-97.
- ZHANG, H. A redefinition of the project risk process: Using vulnerability to open up the event-consequence link. **International Journal of Project Management**, v. 25, n. 7, p. 694-701, out. 2007.
- ZHANG, Y.; MANSOURI, S. A.; HARMAN, M. **The multi-objective next release problem.** 9th Annual Conference on Genetic and evolutionary computation - GECCO '07. Londres: [s.n.]. 2007. p. 1129.
- ZITZLER, E.; LAUMANN, M.; THIELE, L. **Spea2: Improving the strength pareto evolutionary algorithm.** [S.l.]. 2001.
- ZITZLER, E.; THIELE, L. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. **IEEE Transactions on Evolutionary Computation**, v. 4, n. 3, p. 257-271, 1999.

A. jMETAL

Vimos no capítulo anterior uma descrição detalhada a respeito de metaheurísticas, com um enfoque maior nos algoritmos genéticos, devido sua fácil adaptação para resolver problemas com múltiplos objetivos distintos a serem otimizados. Após ter o domínio conceitual destas metaheurísticas, o próximo passo é realizar suas implementações em alguma linguagem de programação, ou, encontrar implementações pré-existentes e adaptá-las para o modelo proposto.

Caso desejássemos realizar uma nova implementação destes algoritmos, além do maior tempo necessário para a programação dos mesmos, ainda enfrentaríamos dois desafios. O primeiro, que o algoritmo deveria ser validado em relação aos seus resultados, normalmente isto é feito a partir de um “teste de sanidade” (HARMAN *et al.*, 2012), demonstrando que o algoritmo supera razoavelmente a qualidade demonstrada por um algoritmo aleatório. Normalmente isto é feito utilizando algum problema clássico de otimização, utilizado como *benchmarking* para atestar estes resultados. Portanto, enfrentaríamos uma bateria, adicionalmente aos testes considerados “úteis”, diretamente relacionados com o modelo proposto para o problema do próximo release multiobjetivo incorporando métricas de risco. Em segundo lugar, seria necessário realizar uma verificação da implementação do algoritmo, demonstrando que a modelagem conceitual dos algoritmos genéticos e seus demais procedimentos (cruzamento, mutação, seleção e substituição) foram realizados corretamente.

Portanto, a segunda alternativa consistindo em selecionar implementações pré-existentes aparenta ser mais razoável, visto que as mesmas já foram validadas por terceiros e estão prontas para uso, economizando o tempo gasto na implementação. O problema principal é que, como os algoritmos de otimização multiobjetivos foram criados por diferentes autores, existe a possibilidade de que estes algoritmos tenham sido implementados em linguagens de programação distintas, o que pode perturbar o resultado. Por exemplo, um algoritmo codificado na linguagem C é mais veloz que este mesmo algoritmo codificado em uma linguagem interpretada, como Python, por exemplo. Portanto, necessitamos de uma implementação uniforme, de modo a eliminar estas possíveis variáveis que possam perturbar o resultado final dos experimentos.

Felizmente, o framework jMetal (DURILLO e NEBRO, 2011) oferece uma solução, em que todos os algoritmos abordados neste trabalho foram pré-implementados e estão prontos para utilização, necessitando apenas de uma configuração e da implementação do problema a

ser resolvido. O jMetal (*Metaheuristics Algorithms in Java*) é um framework designado a resolver problemas de otimização para múltiplos objetivos, apesar que ele também implementa algoritmos tradicionais mono-objetivos como os algoritmos genéticos clássicos. O jMetal possui duas versões, a versão tradicional 4.5.2, disponível no link <<http://jmetal.sourceforge.net>>, e a versão mais recente 5.2, disponível por sua vez no site <<https://jmetal.github.io/jMetal/>>. A versão mais recente, além de incorporar novas metaheurísticas, oferece uma refatoração do código, tornando-o ligeiramente distinto da versão anterior. Em contrapartida, a documentação da versão mais recente se mostrou deficiente, pois possui informações faltantes, exigindo um maior esforço do usuário no aprendizado do framework. Curiosamente, a versão 4.5.2 também inclui alguns algoritmos que foram excluídos da nova versão, além de algumas medidas de desempenho exclusivas. Além disso, a versão 4.5.2 acompanha um manual oficial do usuário e será utilizado como referência principal nas seções seguintes para descrever seu funcionamento em detalhes (NEBRO e DURILLO, 2014). A semelhança entre ambas as versões não prejudica a compreensão do framework.

A Tabela 19 descreve os principais recursos oferecidos pelo jMetal. Podemos observar que, adicionalmente às metaheurísticas descritas neste trabalho, o framework também nos oferece as medidas de desempenho de Hipervolume e Espalhamento, descritas no Capítulo 2 deste documento e que são utilizadas como fator de avaliação dos experimentos realizados nesta proposta. Também podemos notar a disponibilidade de uma codificação binária, o que se ajusta perfeitamente ao problema do Próximo Release. De maneira a acompanhar esta codificação, também estão disponíveis operadores genéticos capazes de lidar com esta codificação, como por exemplo a mutação por *flipping* de bits.

Tabela 19 - Sumário de recursos disponíveis no jMetal.

Algoritmos multi-objetivos	NSGA-II, SPEA2, PAES, PESA-II, OMOPSO, MOCeII, AbYSS, MOEA/D, Denssea, CellDE, GDE3, FastPGA, IBEA, SMPSO, SMS-EMOA, dMOPSO, WASFGA, GWSFGA, MOCHC, MOMBI, MOMBI-II, NSGA-III, WASF-GA, GWASF-GA
Algoritmos mono-objetivos	Algoritmos genéticos, Estratégia de Evolução, DE, CMA-ES, PSO, Otimização de recifes de corais

Algoritmos que possuem implementações paralelas	NSGA-II, SMPSO, GDE3, SPEA2, PESA-II
Indicadores de qualidade	Distância geracional, distância geracional invertida, épsilon aditivo, espalhamento, hipervolume, R2
Codificação do problema	Binário, binário codificado como real (<i>Binary-real</i>), inteiro, inteiro codificado como real (<i>Integer-real</i>), permutação, real.
Operadores de Seleção	Torneio, seleção por melhor <i>fitness</i> , seleção por evolução diferencial, seleção aleatória, seleção por <i>ranking</i>
Operadores de Cruzamento	Cruzamento de ponto único, cruzamento de dois pontos, cruzamento de metade uniforme, cruzamento de mapeamento parcial, cruzamento de simulação binária, cruzamento evolucionário diferencial, cruzamento de mistura alfa
Operadores de Mutação	<i>Flipping</i> , mutação não-uniforme, mutação polinomial, mutação uniforme, mutação por substituição

A.1. Arquitetura do jMetal

Segundo os projetistas do jMetal, o framework foi produzido de modo a atender quatro objetivos principais. O primeiro objetivo é ser **simples e fácil de usar**, caso contrário, o framework não atingiria grande adoção por usuários. Isto é definido na organização de componentes do framework, em que cada um deve ser responsável por uma tarefa específica e não ambígua. O segundo objetivo é a flexibilidade, denotando que todos os algoritmos devem ser executados de maneira uniforme, independentemente da codificação do problema ou dos operadores genéticos escolhidos. A **portabilidade** do framework é obtida com o uso da linguagem Java. Finalmente, a **extensibilidade** define que novos operadores, metaheurísticas e problemas devem ser incorporados com facilidade ao jMetal.

A Figura 40 ilustra um diagrama UML reduzido do jMetal, demonstrando as principais classes envolvidas na execução de um algoritmo. Iniciamos definindo a representação da

solução em objetos do tipo “*Variable*”, formando um cromossomo da população administrada pelo algoritmo na classe “*SolutionSet*”. A **solução** definida em um conjunto de variáveis possui um valor de *fitness* associado, assim como um tipo de codificação que deve ser compatível com o **problema** a ser executado. O diagrama elenca alguns problemas pré-implementados no jMetal, que podem ser usados para testar o funcionamento das metaheurísticas disponíveis, incluídos como uma forma de *benchmarking*. É importante notarmos também que os **operadores** de seleção, mutação e cruzamento são definidos separadamente do algoritmo. Adicionalmente a estes três tipos de operadores, também é disponibilizado um operador de busca local, podendo ser combinado em auxílio com o algoritmo principal.

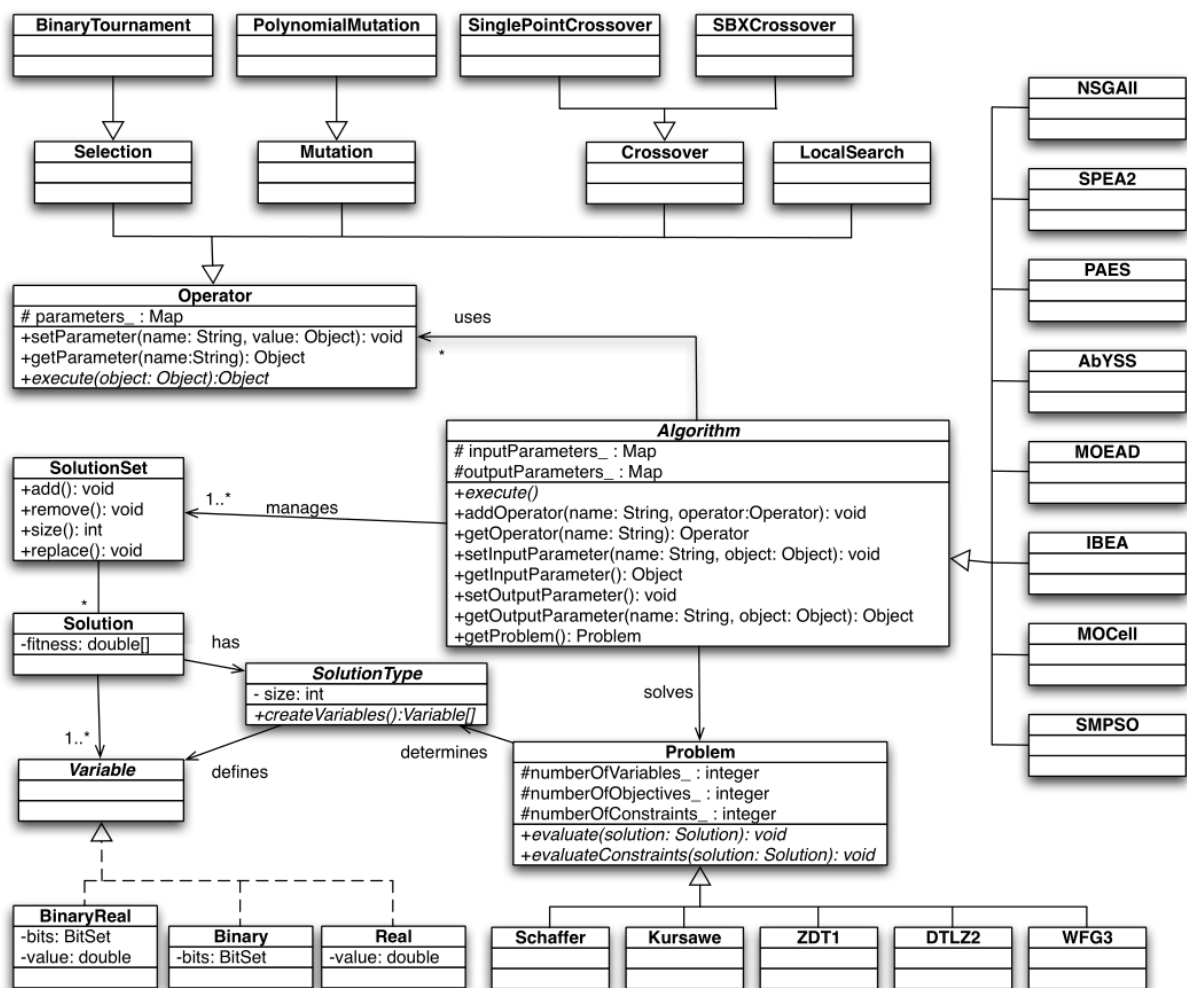


Figura 40 - Diagrama de classes do jMetal (NEBRO e DURILLO, 2014).

Note que a classe “*Problem*” admite três critérios principais: o tamanho do cromossomo, definido pelo atributo “*numberOfVariables*”; a quantidade de objetivos a serem otimizados, definido pelo atributo “*numberOfObjectives*”; e a definição de restrições, se existirem, definidas pelo atributo “*numberOfConstraints*”.

Inicialmente a configuração do algoritmo, conforme visualizado na classe “*Algorithm*”, aparenta ser uma tarefa penosa, em que o algoritmo deve ser modificado manualmente para definir qual o problema a ser otimizado, utilizando qual tipo de codificação e um conjunto de operadores. No entanto, felizmente, conforme podemos ver no conjunto de pacotes ilustrado na Figura 41, o pacote “*experiments*” é responsável por oferecer classes que irão auxiliar nesse procedimento, sem exigir modificações diretas no código do algoritmo.

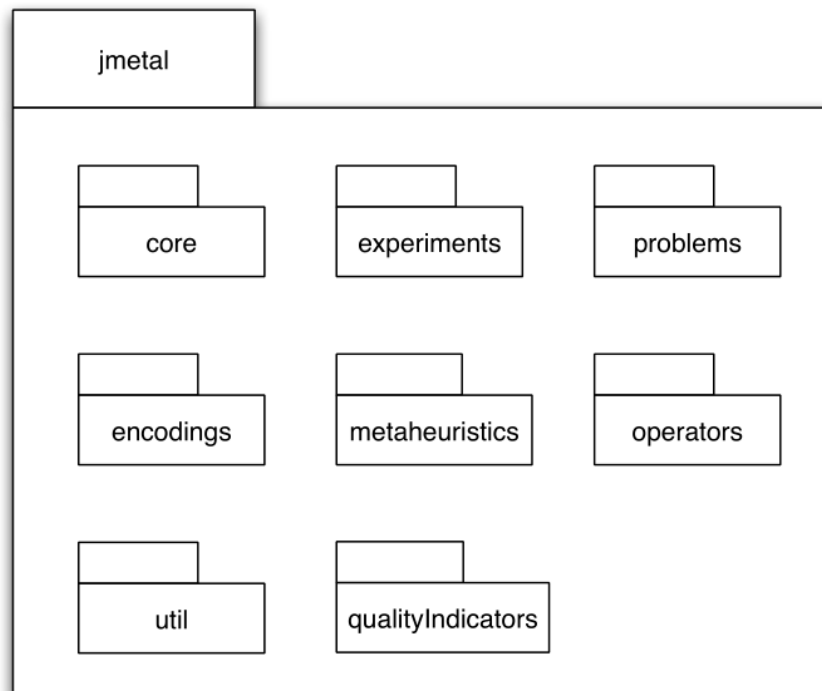


Figura 41 - Pacotes disponíveis no jMetal.

O pacote “*experiments*” é responsável por realizar a ligação entre os algoritmos utilizados no pacote “*metaheuristics*”. Estes algoritmos, por sua vez, serão configurados a partir dos operadores de seleção, mutação e cruzamento disponíveis no pacote “*operators*”. Para avaliar a solução obtida pelas metaheurísticas, o pacote “*qualityIndicators*” é encarregado de fornecer as medidas de desempenho necessárias para esta tarefa, tais como o espalhamento e hipervolume. Finalmente, o problema a ser otimizado é inserido no pacote “*problems*”. A seção seguinte irá detalhar o procedimento necessário.

A.2. Configurando um experimento com o jMetal

Iniciamos a configuração do jMetal focado no problema a ser resolvido. Para isto, observamos as características desejáveis para um problema do próximo release multiobjetivo. Inicialmente, definimos que, por ser um problema combinatorial simples, uma codificação **binária** é suficiente.

A documentação oficial do jMetal recomenda utilizar a codificação “*BinaryReal*”, ou “binário codificado como número real” para representar valores binários. Porém, após testes iniciais, esta codificação apresentou problemas. Os valores das variáveis, que deveriam assumir valores zero, indicando que um dado requisito não foi selecionado, ou valores um, sinalizando a seleção do requisito, produziram valores inexatos, tais como “0,001” para se aproximar do valor nulo ou “0,999” para se aproximar do valor um. Isto pode adulterar ligeiramente o resultado esperado para os valores de *fitness*, portanto, decidimos utilizar a codificação “*Binary*” para o problema, que não apresentou nenhum prejuízo para a execução do algoritmo e produziu valores exatos para os cromossomos.

Com base na codificação, precisamos selecionar operadores compatíveis com valores binários. O operador de seleção escolhido foi a técnica de torneio, por ser o operador de seleção padrão do NSGA-II, SPEA2 e MOCell. O operador de *flipping* para a mutação e o operador de cruzamento de metade uniforme foram selecionado segundo as justificativas já apresentadas no segundo experimento da proposta na Seção 5.2.

A.2.1. Modelagem do problema

Para a modelagem do problema do próximo release, foi necessário criar uma nova classe de problemas, inserida em “**jmetal.problems.NRP**”, seguindo o padrão fornecido pelo jMetal. A classe “NRP1.java” foi definida, sendo necessário sinalizarmos o tamanho da string do cromossomo, a quantidade de objetivos a serem otimizados, o número de restrições do problema e o tipo de codificação utilizado. Estes parâmetros são definidos no construtor da classe. As funções objetivo, por sua vez, devem ser definidas em métodos próprios da classe. Estes objetivos são chamados por um outro método da classe denominado “*Evaluate*”. Este método é responsável por receber como entrada a string de um cromossomo, computar seu valor de *fitness* em relação com a cadeia de bits recebidos e retornar os valores de *fitness* relativos a cada objetivo do problema. É importante frisar que, como os algoritmos assumem que os objetivos sempre devem ser minimizados, caso algum objetivo esteja sendo maximizado, seu valor final de *fitness* deve ser multiplicado por “-1”.

Finalmente, codificado o problema, ele deve ser cadastrado na classe “**jmetal.problems.ProblemFactory**”. Isto se trata de um passo simples em definir um valor de caracteres que será responsável por chamar a classe do problema dentro de um experimento. O padrão a ser seguido pelo jMetal é definir esta string com o mesmo nome do pacote do problema.

A.2.2. Definição do experimento

Para a criação de um experimento, devemos criar uma nova classe no pacote “**jmetal.experiments.studies**”. Tomemos como exemplo a classe “StandardStudy.java”. Esta classe exige a definição do nome do experimento a ser realizado, os algoritmos utilizados, o(s) problema(s) a serem otimizados por estes algoritmos selecionados, e, finalmente, os indicadores de qualidade utilizados. O Algoritmo 7 ilustra estes parâmetros. Omitimos os parâmetros adicionais de lista de fronteiras de Pareto e do diretório do experimento por uma questão de simplificação. O parâmetro “*independentRuns*” sinaliza a quantidade de vezes que um experimento deve ser executado. É possível sinalizar a quantidade de *threads* a serem utilizadas pelo jMetal. Por exemplo, caso o experimento seja realizado em um computador com processador de oito núcleos, o usuário pode ajustar o parâmetro em 8 *threads*, de forma a serem alocadas em cada um dos núcleos disponíveis no processador.

Algoritmo 7 - Parâmetros de entrada de “StandardStudy”.

```

01 exp.experimentName_ = "StandardStudy";
02 exp.algorithmNameList_ = new String[]{
    "NSGAI", "SPEA2", "MOCe", "SMPSO", "GDE3"};
03 exp.problemList_ = new String[]{"ZDT1", "ZDT2", "ZDT3", "ZDT4", "ZDT6",
    "WFG1", "WFG2", "WFG3", "WFG4", "WFG5", "WFG6", "WFG7", "WFG8", "WFG9",
    "DTLZ1", "DTLZ2", "DTLZ3", "DTLZ4", "DTLZ5", "DTLZ6", "DTLZ7"};
04 exp.indicatorList_ = new String[]{"HV", "SPREAD", "EPSILON"};
05 exp.independentRuns_ = 100;

```

Adicionalmente, cada algoritmo selecionado deve possuir uma classe de configuração correspondente no pacote “**jmetal.experiments.settings**”. Por exemplo, o NSGA-II possui uma classe de configuração denominada “NSGAIISettings.java”. Nesta classe, definimos o tamanho da população, o número máximo de iterações, a taxa de mutação, a taxa de crossover e quaisquer outros parâmetros adicionais necessários para a execução do algoritmo. O SPEA2 e o MOCe possuem um parâmetro adicional para definir o tamanho do arquivo, o MOCe exclusivamente possui um valor de *feedback* a ser configurado. É nesta classe de configuração que anexamos os operadores de cruzamento, mutação e seleção que serão utilizados pelo algoritmo neste experimento.

Para um problema de codificação binária, o arquivo de configuração deve ser configurado apropriadamente, conforme observado no Algoritmo 8.

Algoritmo 8 - Representação binária para o algoritmo.

```
01 object [] problemParams = {"Binary"};
```

Além disso, precisamos definir os parâmetros utilizados pelo NSGA-II, conforme visto no Algoritmo 9.

Algoritmo 9 - Parâmetros de configuração do NSGA-II.

```
01 populationSize_ = 1000;
02 maxEvaluations_ = 2500;
03 mutationProbability_ = 0.1;
04 crossoverProbability_ = 0.9 ;
```

O método “**configure()**” dentro da classe “NSGAIISettings.java” adiciona estes parâmetros ao algoritmo e seleciona os operadores genéticos apropriados, conforme observado no Algoritmo 10.

Algoritmo 10 - Configuração do NSGA-II a partir do método "configure()".

```
01 //Tamanho da população e número máximo de iterações

02 populationSize_ =
    Integer.parseInt(configuration.getProperty("populationSize",
    String.valueOf(populationSize_)));

03 maxEvaluations_ =
    Integer.parseInt(configuration.getProperty("maxEvaluations",
    String.valueOf(maxEvaluations_)));

04 //Inserindo estes parâmetros no algoritmo

05 algorithm.setInputParameter("populationSize",populationSize_);
06 algorithm.setInputParameter("maxEvaluations",maxEvaluations_);

07 //selecionando o operador de cruzamento de metade uniforme e
    atribuindo sua taxa de cruzamento

08 crossoverProbability_ =
    Double.parseDouble(configuration.getProperty(
    "crossoverProbability",String.valueOf(crossoverProbability_)));

09 parameters = new HashMap() ;

10 parameters.put("probability", crossoverProbability_) ;

11 crossover = CrossoverFactory.getCrossoverOperator("HUXCrossover",
    parameters);
```

```

12 //Selecionando o operador de mutação flipping e atribuindo sua taxa
    de mutação definida
13 mutationProbability_ = Double.parseDouble(configuration.getProperty(
    "mutationProbability",String.valueOf(mutationProbability_)));
14 parameters = new HashMap() ;
15 parameters.put("probability", mutationProbability_) ;
16 mutation = MutationFactory.getMutationOperator("BitFlipMutation",
    parameters);
17 //Definição da seleção por torneio, que não possui parâmetros de
    configuração
18 parameters = null ;
19 selection = SelectionFactory.getSelectionOperator("BinaryTournament",
    parameters) ;
20 //adicionando os operadores genéticos ao algoritmo
21 algorithm.addOperator("crossover",crossover);
22 algorithm.addOperator("mutation",mutation);
23 algorithm.addOperator("selection",selection);

```

De posse das classes de configuração de cada algoritmo, o experimento é iniciado a partir do método “**exp.initExperiment();**”. Em sequência, geramos os indicadores de desempenho com o método “**exp.generateQualityIndicators();**”.

Opcionalmente, podemos criar um arquivo com um front de referência a ser utilizado por estes indicadores de qualidade. Dentro do diretório que foi definido para o experimento, criamos uma nova pasta denominada “referenceFronts”. O arquivo dentro desta pasta receberá o nome do experimento com uma extensão “.rf”. Este arquivo-texto deve conter os valores de *fitness* para cada ponto do espaço de busca que seja de interesse. A criação deste arquivo pode ser interessante nas situações em que é necessário comparar execuções por algoritmos distintos, ou quando realizamos uma nova modelagem para o mesmo problema, observando o quanto a nova modelagem é capaz de melhorar estes valores de referência. Caso o arquivo não seja criado manualmente pelo usuário, o jMetal criará um front de referência com valores obtidos pelo problema a partir da execução de alguma metaheurística, armazenando os melhores pontos encontrados com seus respectivos valores em um arquivo com extensão “.rf” automaticamente dentro da pasta “referenceFronts”.