UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

RODRIGO BASNIAK

SELEÇÃO DE REQUISITOS DE SOFTWARE UTILIZANDO MÉTODOS META-HEURÍSTICOS

DISSERTAÇÃO

CURITIBA

2019

RODRIGO BASNIAK

SELEÇÃO DE REQUISITOS DE SOFTWARE UTILIZANDO MÉTODOS META-HEURÍSTICOS

Dissertação apresentada ao Programa de Pós-graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de ”Mestre em Computação Aplicada” – Área de Concentração: Engenharia de Sistemas Computacionais.

Orientador: Prof.ª Dra. Maria Cláudia Figueiredo Pereira Emer

Co-orientador: Prof. Dr. Laudelino Cordeiro Bastos

CURITIBA

2019

Dados Internacionais de Catalogação na Publicação

Basniak, Rodrigo
Seleção de requisitos de software utilizando métodos metaheurísticos [recurso eletrônico] / Rodrigo Basniak.-- 2019.
1 arquivo eletrônico (117 f.) : PDF ; 1,76 MB.

Modo de acesso: World Wide Web.
Texto em português com resumo em inglês.
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada. Área de Concentração: Engenharia de Sistemas Computacionais. Linha de Pesquisa: Engenharia de Software, Curitiba, 2019.
Bibliografia: f. 94-99.

1. Computação - Dissertações. 2. Metaheurísticas. 3. Software - Desenvolvimento. 4. Processo decisório. 5. Modelos matemáticos. 6. Inteligência coletiva. 7. Algoritmos genéticos. 8. Computação evolutiva. 9. Algoritmos de formiga. 10. Otimização matemática. 11. Otimização combinatória. 12. Solução de problemas. 13. Métodos de simulação. I. Emer, Maria Cláudia Figueiredo Pereira, orient. II. Bastos, Laudelino Cordeiro, coorient. III. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada. IV. Título.

CDD: Ed. 23 -- 621.39

Biblioteca Central do Câmpus Curitiba - UTFPR
Bibliotecária: Luiza Aquemi Matsumoto CRB-9/794

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Pesquisa e Pós-Graduação

ATA DA DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 75

DISSERTAÇÃO PARA OBTENÇÃO DO TÍTULO DE MESTRE EM COMPUTAÇÃO APLICADA
PROGRAMA DE PÓS-GRADUAÇÃO EM: COMPUTAÇÃO APLICADA
ÁREA DE CONCENTRAÇÃO: ENGENHARIA DE SISTEMAS COMPUTACIONAIS
LINHA DE PESQUISA: ENGENHARIA DE SOFTWARE

No dia 27 de junho de 2019 às 9h30 reuniu-se na Sala B-204 da Sede Centro a banca examinadora composta pelos pesquisadores indicados a seguir, para examinar a dissertação de mestrado do candidato Rodrigo Basniak**,** intitulada: Seleção de requisitos de software utilizando métodos meta-heurísticos.

Orientadora: Profa. Dra. Maria Claudia Figueiredo Pereira Emer

Coorientador: Prof. Dr. Laudelino Cordeiro Bastos

Após a apresentação, o candidato foi arguido pelos examinadores que, em seguida à manifestação dos presentes, consideraram o trabalho de pesquisa: ( ) Aprovado. ( ) Aprovado com restrições. Revisor indicado para verificação: _____. ( ) Reprovado.

Observações:
_____
_____
_____

Nada mais havendo a tratar, a sessão foi encerrada às __h__, dela sendo lavrado a presente ata, que segue assinada pela Banca Examinadora e pelo Candidato.

O candidato está ciente que a concessão do referido título está condicionada à: (a) satisfação dos requisitos solicitados pela Banca Examinadora; (b) entrega da dissertação em conformidade com as normas exigidas pela UTFPR; (c) atendimento ao requisito de publicação estabelecido nas normas do Programa; e (d) entrega da documentação necessária para elaboração do Diploma. A Banca Examinadora determina um **prazo máximo de _____ dias**, considerando os prazos máximos definidos no Regulamento Geral do Programa, para o cumprimento dos requisitos (desconsiderar caso reprovado), sob pena de, não o fazendo, ser desvinculado do Programa sem o Título de Mestre.

Prof. Dr. Laudelino Cordeiro Bastos – Presidente – UTFPR _____

Profª. Drª. Silvia Regina Vergilio – UFPR _____

Prof. Dr. Heitor Silverio Lopes – UTFPR _____

Prof. Dr. Adolfo Gustavo Serra Seca Neto – UTFPR _____

Assinatura do Candidato: _____

Reservado à Coordenação

DECLARAÇÃO PARA A OBTENÇÃO DO TÍTULO DE MESTRE

A Coordenação do Programa declara que foram cumpridos todos os requisitos exigidos pelo Programa de Pós-Graduação para a obtenção do título de Mestre.

Curitiba, _____de _____de 20____.

Carimbo e Assinatura do(a) Coordenador(a) do Programa

Com muito carinho e afeto dedico este meu trabalho à minha esposa e meu filho.

AGRADECIMENTOS

Agradeço ao meus orientadores, Professora Maria Claudia Emer e Professor Dr. Laudelino Bastos, por compartilhar seus conhecimentos e por me orientar e motivar nesta caminhada do conhecimento.

Agradeço a minha família, em especial à minha esposa Bruna e meu filho Rafael, pelo apoio e compreensão nos vários momentos que precisei estar ausente durante esta jornada.

Por fim, agradeço a todos os professores do PPGCA da Universidade Tecnológica Federal do Paraná, por todo o conhecimento e experiência que pude absorver ao longo das aulas.

RESUMO

BASNIAK, Rodrigo. SELEÇÃO DE REQUISITOS DE SOFTWARE UTILIZANDO MÉTODOS META-HEURÍSTICOS . 118 f. DISSERTAÇÃO – PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA , Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

A seleção de requisitos de software é muito importante para qualquer projeto de software e decisões ruins nesta etapa podem atrapalhar ou até mesmo impedir seu sucesso. Isto faz com que estudos sobre a aplicação de técnicas para auxiliar na tomada de decisão na seleção de requisitos também sejam importantes. Uma das abordagens utilizadas na literatura é modelar a seleção de requisitos como um problema de otimização e utilizar métodos meta-heurísticos para resolvê-lo. Um mapeamento sistemático de literatura foi realizado com o objetivo de identificar os problemas existentes nas soluções já propostas e as lacunas desta área de pesquisa. Este mapeamento constatou que as soluções propostas tratam o problema de forma binária e simplificado, ignorando as interdependências entre os requisitos e utilizando dados de problemas gerados aleatoriamente. Poucos trabalhos consideram todos os tipos de interdependências existentes entre os requisitos e quando o fazem, a solução é utilizar operadores de reparação. Com o intuito de propor uma solução mais robusta, o problema que é inerentemente binário, é tratado como um problema de variáveis contínuas e com interdependências entre os requisitos. Técnicas meta-heurísticas são utilizadas para a encontrar a solução e com auxílio de uma codificação desenvolvida especificamente para o problema de seleção de requisitos com interdependências. Desta forma, a superfície do espaço de busca fica mais suave e as meta-heurísticas conseguem chegar nas melhores soluções mais facilmente. Três algoritmos foram selecionados para o estudo: otimização por enxame de partículas, evolução diferencial e algoritmos genéticos. Experimentos foram realizados com a codificação proposta e com uma codificação binária, que é mais comum na literatura. Os resultados mostram que a abordagem proposta é superior à tradicional quando são consideradas as interdependências entre os requisitos, pois em projetos com mais 32 requisitos a abordagem tradicional não consegue resolver os problemas utilizados neste trabalho. Os dados dos problemas foram gerados a partir de projetos reais de desenvolvimento de software e foram disponibilizados publicamente, juntamente com uma ferramenta protótipo para execução dos algoritmos. Com a codificação proposta, os três algoritmos tiveram uma performance parecida em relação à qualidade dos resultados, mas o algoritmo de evolução diferencial se mostrou superior em relação ao número de avaliações da função de *fitness*.

Palavras-chave: Requisito de Software, Next Release Problem, Evolução Diferencial, Otimização por Enxame de Partículas, Algoritmos Genéticos

ABSTRACT

BASNIAK, Rodrigo. Software Requirements Selection Using Meta-Heuristic Methods. 118 f. DISSERTAÇÃO – PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA , Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

The software requirements selection is an important step in any software development software and bad decisions can cause the project to fail. This makes reasearches on techniques to aid the professionals on the decision making process also be important. One existing approach is to model the problem as an optimization problem an solve it using meta-heristic thecniques. A systematic literature review was done to help identify the problems existent in this research field. The review found that the proposed solutions interpret the problem as a binary problem and simplifies it removing all interdependencies that exists between the requirements and use randomly generated datasets. Just a few researches take the full set of interdependencies in account and when they do they use repair operators to fix invalid solutions. With a goal of proposing a more robust solution, the problem that is inherently binary, is treated as a continuous space problem and with all interdependencies between the requirements. Meta-heuristic techniques are then used to solve the requirements selection problem with the help of a new solution encoding, developed to solve exactly this type of problem. With this approach the search space surface gets smoother and the meta-heuristic techniques can find better solutions. Three algorithms were selected to solve the problem: differential evolution, particle swarm optimization and genetic algorithms. Results obtained with experiments using the proposed encoding were compared to the results obtained using the traditional binary encoding. The proposed technique is superior to the traditional one when the interdependencies between the requirements are considered, since in projects with more than 32 requirements the traditional approach can't even find any valid solutions. The projects used in this paper were generated based on three real world software development projects. All these data were made publicly available together with a prototype software tool to run the algorithms developed in this paper. Using the proposed encoding the three algorithms performed similarly when comparing the quality of theirs results but, the differential evolution algorithmwas superior when observing the number of fitness evaluations needed to find the best solutions.

Keywords: Software Requirement, Next Release Problem, Diferential Evolution, Particle Swarm Optimization, Genetic Algorithm

LISTA DE FIGURAS

FIGURA 1 – Etapas do trabalho 18

FIGURA 2 – Etapas de um GA 32
FIGURA 3 – Exemplo de *crossover* de um-ponto 33
FIGURA 4 – Atualização da posição de uma partícula 41

FIGURA 5 – Superfície do espaço de busca de um problema contínuo 52
FIGURA 6 – Superfície do espaço de busca de um problema binário 52
FIGURA 7 – Vetor solução da codificação binária com penalização 54
FIGURA 8 – Vetor solução da codificação contínua 60
FIGURA 9 – Tela principal do NRP Solver 72
FIGURA 10 – Área de carregamento/salvamento de resultados 73
FIGURA 11 – Área de configuração do *dataset* 74
FIGURA 12 – Área de configuração do algoritmo DE 75
FIGURA 13 – Área de configuração do algoritmo PSO 76
FIGURA 14 – Área de configuração do algoritmo GA 77
FIGURA 15 – Área de configuração do número de execuções independentes 77
FIGURA 16 – Área de exibição de resultados 79

FIGURA 17 – Evolução do tempo de processamento com o aumento dos requisitos 81
FIGURA 18 – Curva de *fitness* do projeto CBS 85
FIGURA 19 – Curva de *fitness* do projeto FPG 85
FIGURA 20 – Curva de *fitness* do projeto SRT 86

LISTA DE TABELAS

TABELA 1 – Lista de interdependências entre requisitos. Fonte: Adaptado de Carlshamre et al. (2001) 23

TABELA 2 – *Datasets* utilizados no trabalho. Fonte: Autoria Própria 50
TABELA 3 – Quantidade de cada interdependência para cada projeto. Fonte: Autoria Própria 51
TABELA 4 – Valores de importância para cada *stakeholder* dos projetos. Fonte: Autoria Própria 51

TABELA 5 – Resultados de *baseline*. Fonte: Autoria Própria 80
TABELA 6 – Resultados obtidos pelos alunos. Fonte: Autoria Própria 82
TABELA 7 – Resultados obtidos utilizando a codificação binária. Fonte: Autoria Própria 83
TABELA 8 – Resultados obtidos utilizando a codificação contínua e DE. Fonte: Autoria Própria 86
TABELA 9 – Resultados obtidos utilizando a codificação contínua e PSO. Fonte: Autoria Própria 87
TABELA 10 – Resultados obtidos utilizando a codificação contínua e GA. Fonte: Autoria Própria 88
TABELA 11 – Compilação de resultados obtidos por todos os métodos. Fonte: Autoria Própria 89

LISTA DE SIGLAS

IoT	*Internet of Things*
KP	*Knapsack Problem*
AHP	Analytical Hierarchy Process
QFD	*Quality Function Deployment*
UTFPR	Universidade Tecnológica Federal do Paraná
MDS	*Market Driven Software*
PO	*Product Owner*
KP	*Knapsack Problem*
NRP	*Next Release Problem*
GA	*Genetic Algorithm*
PSO	*Particle Swarm Optimization*
ACO	*Ant Colony Optimization*
GPSO	*Greedy Particle Swarm Optimization*
SCPSO	*Socio-Cognitive Particle Swarm Optimization*
BPSO	*Binary Particle Swarm Optimization*
MBPSO	*Multi Mutation Binary Particle Swarm Optimization*
NHBPSO	*New Hybrid Binary Particle Swarm Optimization*
FA	*Firefly Algorithm*
QEA	*Quantum Evolutionary Algorithm*
QSEA	*Quantum Swarm Evolutionary Algorithm*
GAF	*Genetic Algorithm Framework*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 MOTIVAÇÃO E JUSTIFICATIVA 16
1.2 OBJETIVOS 17
1.2.1 Objetivo Geral 17
1.2.2 Objetivos específicos 17
1.3 METODOLOGIA 17
1.4 ESTRUTURA DO TRABALHO 19
2 FUNDAMENTAÇÃO TEÓRICA 21
2.1 REQUISITO DE SOFTWARE 21
2.2 INTERDEPENDÊNCIAS ENTRE REQUISITOS 23
2.3 MÉTODOS ANALÍTICOS PARA SELEÇÃO DE REQUISITOS 23
2.4 PROBLEMA DA MOCHILA 0-1 25
2.4.1 Definição Intuitiva 25
2.4.2 Definição Formal 26
2.4.3 Complexidade 26
2.5 *NEXT RELEASE PROBLEM* 26
2.5.1 Modelo de Satisfação Total 27
2.5.2 Modelo de Satisfação Parcial 28
2.6 META-HEURÍSTICAS E OTIMIZAÇÃO COMBINATÓRIA 29
2.6.1 Algoritmos Genéticos 30
2.6.1.1 Seleção 32
2.6.1.2 Cruzamento 33
2.6.1.3 Mutação 33
2.6.2 Evolução Diferencial 34
2.6.2.1 Mutação 36
2.6.2.2 Cruzamento 37
2.6.2.3 Seleção 37
2.6.2.4 Parâmetros 37
2.6.3 Otimização por Enxame de Partículas 38
2.7 CONSIDERAÇÕES SOBRE ESTE CAPÍTULO 41
3 TRABALHOS RELACIONADOS 42
3.1 MODELAGEM 42
3.2 MÉTODOS 43
3.2.1 Evolução Diferencial 43
3.2.2 Otimização por Colônia de Formigas 44
3.2.3 Otimização por Enxame de Partículas 44
3.2.4 Outros 45
3.3 LACUNAS 46
3.4 CONSIDERAÇÕES SOBRE ESTE CAPÍTULO 47
4 ABORDAGENS ADOTADAS PARA SELEÇÃO DE REQUISITOS 48
4.1 ABORDAGEM PROPOSTA 48

4.2 PROJETOS 49
4.3 CODIFICAÇÃO 51
4.3.1 Codificação binária com penalização progressiva 53
4.3.1.1 Função de *fitness* 55
4.3.1.2 Decodificação 57
4.3.2 Codificação contínua 59
4.3.2.1 Função de *fitness* 60
4.3.2.2 Decodificação 62
4.4 META-HEURÍSTICAS 66
4.4.1 Algoritmos Genéticos 66
4.4.2 Evolução Diferencial 66
4.4.3 Otimização por Enxame de Partículas 68
4.5 MÉTODO EXATO 70
4.6 ESTUDO DE CASO COM ALUNOS 71
4.7 FERRAMENTA NRP SOLVER 71
4.8 CONSIDERAÇÕES SOBRE ESTE CAPÍTULO 78
5 RESULTADOS E DISCUSSÃO 80
5.1 BASELINE 80
5.2 SOLUÇÃO EXATA 81
5.3 ESTUDO DE CASO COM ALUNOS 82
5.4 CODIFICAÇÃO BINÁRIA COM PENALIZAÇÃO PROGRESSIVA 83
5.5 CODIFICAÇÃO CONTÍNUA 84
5.5.1 Evolução Diferencial 84
5.5.2 Otimização por enxame de partículas 87
5.5.3 Algoritmos Genéticos 88
5.6 COMPILAÇÃO DE RESULTADOS 89
5.7 AMEAÇAS À VALIDADE 90
5.8 CONSIDERAÇÕES SOBRE ESTE CAPÍTULO 90
6 CONCLUSÃO 92
6.1 CONTRIBUIÇÕES 93
6.2 TRABALHOS FUTUROS 94
REFERÊNCIAS 95

1 INTRODUÇÃO

O ciclo de vida de um software é composto por diversas fases, sendo que a primeira delas é a coleta e análise dos requisitos que o sistema deve atender. Esta é uma área tão ampla e importante, que se transformou em um campo de estudo próprio, denominado de Engenharia de Requisitos.

Para muitos autores, o sucesso de um projeto de desenvolvimento de software começa nessa etapa. Quando um problema que nasceu na fase de requisitos é encontrado em fases avançadas de desenvolvimento, mais recursos deverão serem gastos para corrigir este problema. Recursos esses, que deixarão de ser consumidos para o desenvolvimento de outros requisitos que já haviam sido planejados. Numa situação como esta, o melhor que se pode esperar é que os requisitos que não poderão mais ser desenvolvidos sejam de funcionalidades não essenciais para o software (SAWYER et al., 1999).

Antes que os requisitos possam ser analisados, modelados e detalhados, eles precisam ser levantados e priorizados por meio de um processo de elicitação de requisitos. Reuniões e entrevistas são realizados com os envolvidos no projeto e possíveis usuários do sistema, também conhecidos como *stakeholders*. Indiferente dos métodos utilizados nesta fase, o resultado final é sempre o mesmo: uma lista com vários desejos e expectativas por parte dos *stakeholders* de como o software deve funcionar e o que ele deve realizar (JANTUNEN et al., 2011). Cada item desta lista é chamado de Requisito de Software, e é a partir deles que a equipe de desenvolvimento traduz os desejos dos stakeholders em um produto de software. Se todos esses requisitos forem corretamente implementados no sistema, em teoria, todos os *stakeholders* estariam completamente satisfeitos com o produto final, pois ele atenderia a todas as expectativas.

Em projetos reais é comum os *stakeholders* terem um orçamento máximo e previamente definido, e este orçamento raramente comporta a implementação de todos os requisitos desejados (JANTUNEN et al., 2011). Para ajudar no processo decisório de quais requisitos serão ou não implementados, nasceu uma área de pesquisa que tem por objetivo determinar meios e métodos para seleção de requisitos. A decisão final deve atender da melhor forma possível os

stakeholders e com isso garantir o maior nível de satisfação deles para com o projeto.

Com o desenvolvimento da Internet das Coisas (IoT), mais e mais produtos conectados à internet estão surgindo, mesmo em aparelhos que há algumas décadas jamais se imaginaria que poderiam ter um software rodando. Devido a estes aparelhos, cada vez mais softwares inundam o mercado e, portanto, boas soluções para o problema de seleção de requisitos são muito importantes. Problemas durante a fase de seleção de requisitos podem acarretar em um custo mais elevado do produto final, e até por em risco vidas de pessoas como é o caso de softwares que rodam em carros autônomos (XIE et al., 2017). Desta forma, novas soluções são importantes não apenas para os projetos de software em si, como também para a sociedade em geral.

No relatório anual do *Standish Group* [1] é possível observar que, nos Estados Unidos, a cada ano são gastos mais de 250 bilhões em desenvolvimento de aplicações. De um total de 8380 projetos analisados, 31% são cancelados antes de estarem completos, 52% dos projetos custam 189% a mais do que o orçamento planejado originalmente e apenas 16% são finalizados no prazo e dentro do orçamento inicial. Ainda de acordo com esse relatório, os requisitos de software são um dos principais responsáveis por essas falhas, seja na fase de levantamento, priorização ou elicitação.

Se a fase de seleção de requisitos for negligenciada, as consequências para o projeto ou até mesmo a empresa podem ser catastróficas (RUHE, 2001). Mesmo assim, estudos revelaram que as práticas utilizadas na indústria são baseadas principalmente em procedimentos informais e, a maioria deles não se baseia no envolvimento abrangente dos *stakeholders* ou dá atenção suficiente aos recursos e restrições que podem afetar as decisões (RUHE; SALIU, 2005).

Para atender os *stakeholders* da melhor forma possível é vital que os requisitos de software sejam levantados, analisados e selecionados corretamente (PERINI et al., 2007). Este processo de seleção consiste em identificar requisitos chave dentro de um conjunto maior de requisitos. Assim, os requisitos menos importantes são deixados para o final e caso haja uma interrupção no projeto ou corte no orçamento as funcionalidades mais importantes do sistema já foram implementadas (RUHE; SALIU, 2005). Outra vantagem dessa abordagem fica clara quando o projeto é desenvolvido utilizando metodologias ágeis de desenvolvimento, em que versões parciais do sistema são entregues em intervalos regulares de tempo: logo no início do projeto as primeiras versões do sistema já possuem funcionalidades importantes e são mais funcionais, gerando um melhor engajamento com os usuários, maior aceitação do sistema e

[1] The Standish Group Report. https://www.projectsmart.co.uk/white-papers/chaos-report.pdf. Último acesso: 18/11/2017.

uma maior propensão a dar um parecer para a equipe de desenvolvimento.

Com base no cenário exposto fica clara a necessidade e a importância de métodos para auxiliar na priorização de requisitos. Uma das vertentes que estudam soluções para esse problema o tratam como um problema de otimização que deve então ser resolvido computacionalmente (BAGNALL et al., 2001). Essa também será a abordagem utilizada neste trabalho.

A seleção de requisitos é um problema NP-difícil modelado com base no Problema de Mochila (KP) (GAREY, 1979). O objetivo a ser otimizado é a satisfação global dos *stakeholders* sem exceder o orçamento disponível. Na literatura já foram propostos métodos para seleção de requisitos como o Processo Analítico Hierárquico (AHP) e a Desdobramento da Função Qualidade (QFD), ambos propostos por Karlsson (KARLSSON; AHLSTRöM, 1996). No QFD os requisitos são priorizados em uma escala ordinal e no AHP os requisitos são classificados por pares. Entretanto, nenhum destes métodos suportam as interdependências que existem entre os requisitos, que são necessidades reais em projetos de software, e eles precisam realizar um grande número de comparações entre os requisitos quando o tamanho do projeto cresce.

O problema de seleção de requisitos foi formulado como um problema de otimização de objetivo simples pela primeira vez em 2001 e tem sido resolvido na literatura utilizando diferentes meta-heurísticas. Já em 2007 o problema foi formulado como um problema de otimização multi-objetivo, no qual outros fatores que podem afetar o processe de seleção também são levados em conta e, ao final várias soluções são propostas para que o profissional decida qual será adotada (ZHANG et al., 2007). Entretanto, a maior parte dos trabalhos publicados não leva em conta a interdependência entre os requisitos nem foi testada com dados de projetos reais.

No trabalho (FINKELSTEIN et al., 2009), os autores utilizaram uma abordagem multi-objetivo para uma análise entre múltiplos clientes com desejos conflitantes, mas as interdependências não foram consideradas. Ao longo dos anos vários algoritmos foram criados para resolver o problema, desde os mais tradicionais como algoritmos genéticos (DURILLO et al., 2011) até algoritmos inspirados em comportamentos quânticos (KUMARI et al., 2013).

Baseado na literatura estudada, apenas os trabalhos (SAGRADO et al., 2011), (SAGRADO et al., 2015), (CHAVES-GONZáLEZ; PÉREZ-TOLEDANO, 2015) e (SOUZA et al., 2011) levaram em consideração todas as interdependências entre os requisitos, porém apenas os dados dos trabalhos de Sagrado foram disponibilizados publicamente. Apesar disso, estes dados não são de projetos reais, mas sim gerados aleatoriamente e com apenas 20 requisitos. Outro fator importante nesses trabalhos é que para tratar as interdependências corretamente nas soluções foram utilizados operadores de reparação nos algoritmos, ou seja, quando eles geram

soluções inválidas elas são tratadas posteriormente de acordo com critérios pré-estabelecidos.

Com o objetivo de gerar uma maior contribuição para este campo de pesquisa, este trabalho propões novas bases de dados de requisitos geradas a partir de projetos reais de software e uma nova forma de codificar o problema de modo a eliminar a necessidade de operadores de reparação ou outros meios para tratar soluções inválidas. Isso é feito tratando o problema como um problema de variáveis contínuas ao invés de binárias, como é feito em toda a literatura estudada. As bases de dados possuem vários *stakeholders* com pesos diferentes e diversas interdependências entre os requisitos, pois do contrário tais bases de dados não refletem a real situação de projetos de desenvolvimento de software (LIM; FINKELSTEIN, 2012).

Tais contribuições são importantes pois possibilitam que outros autores possam realizar pesquisas com dados de projetos reais, visto que as bases de dados foram disponibilizadas publicamente na página do *Github* do autor. Além disso, a nova codificação proposta elimina completamente soluções inválidas durante a execução do algoritmo, garantindo que apenas soluções válidas sejam geradas e contribuindo assim para uma melhor convergência do algoritmo.

1.1 MOTIVAÇÃO E JUSTIFICATIVA

Muitas vezes quem toma a decisão de quais funcionalidades serão ou não implementadas em um software não possui uma visão ampla do projeto ou das necessidades de todos os envolvidos, resultando em decisões ruins e *releases* que deixam a desejar do ponto de vista dos usuários e *stakeholders* (RUHE, 2001).

Mesmo quando há uma maior participação dos *stakeholders* na fase de planejamento, outro problema emerge, que é o conflito de interesses no qual não existem recursos para incluir duas ou mais funcionalidades desejadas por diferentes *stakeholders*. Para um software de grande porte, a quantidade de requisitos e *stakeholders* é tão grande que pode gerar um grande stress na equipe durante o processo de seleção de requisitos a ponto de comprometer a qualidade das decisões (QASIM; ISMAIL, 2017).

Com o surgimento de novas técnicas para solução de problemas de busca e otimização, bem como do aumento na capacidade de processamento dos computadores, os problemas podem ser resolvidos muito mais rapidamente ou executar muito mais avaliações em cima dos resultados de modo a encontrar soluções melhores. O estudo destas técnicas em problemas do mundo real é muito importante para a computação em geral, pois estes algoritmos atuam com o conceito de “caixa preta”, ou seja, eles recebem uma entrada, processam e devolvem uma saída, sem ter absolutamente nenhum conhecimento do problema em questão.

Como a seleção de requisitos é uma etapa que existe em qualquer tipo de projeto e não apenas nos projetos de software, saber quais métodos aplicar sobre esse problema de modo a obter melhores resultados possui uma grande importância para muitas áreas de conhecimento, e não apenas para a computação (ZHANG et al., 2008).

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

O objetivo geral deste trabalho é avaliar o comportamento de métodos meta-heurísticos em problemas reais de seleção de requisitos de software, propondo um novo tipo de codificação para este problema e comparando os resultados obtidos manualmente com os resultados gerados pelos algoritmos. A fim de garantir que este objetivo seja alcançado e bem avaliado, alguns objetivos específicos foram levantados e serão apresentados na próxima seção.

1.2.2 OBJETIVOS ESPECÍFICOS

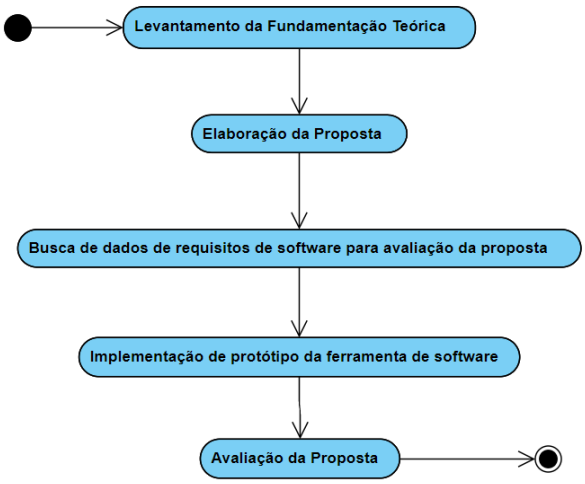
- Determinar o estado da arte desta área de pesquisa, por meio de uma revisão sistemática de literatura, que servirá como embasamento para seleção das meta-heurísticas a serem utilizadas;
- Levantar junto de empresas de desenvolvimento de software dados de requisitos de projetos reais com interdependências entre os requisitos;
- Avaliar se é possível obter resultados superiores aos utilizados durante o desenvolvimento dos projetos com o emprego de meta-heurísticas na seleção de requisitos de software.

1.3 METODOLOGIA

Para atingir os objetivos do trabalho as etapas apresentadas na Figura 1 e serão descritas na sequência.

- **Levantamento da Fundamentação Teórica:** dois mapeamentos sistemáticos de literatura fora realizados: um sobre seleção de requisitos utilizando meta-heurísticas e outro sobre o problema da mochila. O mapeamento sistemático de literatura contribui para identificar novas oportunidades de pesquisa e prover uma visão geral destes tópicos. Isso é feito por meio de métodos explícitos e rigorosos para identificar, avaliar criticamente e

Figura 1: Etapas do trabalho



Fonte: Autoria Própria

sintetizar estudos relevantes em uma área particular de pesquisa. A metodologia utilizada para realização dos mapeamentos é a proposta no trabalho de Petersen (PETERSEN et al., 2008). O motivo de serem escolhidos dois temas para mapeamento é que durante os estudos preliminares sobre seleção de requisitos utilizando métodos de otimização não foi encontrado nenhum estudo que relacionasse os avanços dessa área com os avanços da área de pesquisa sobre o problema de mochila. Avaliou-se que seria muito importante conhecer os avanços e o estado da arte das duas áreas porque o problema de seleção de requisitos é na verdade uma aplicação prática do problema de mochila. A descrição detalhada do processo de mapeamento e os resultados encontrados estão disponíveis no Apêndice A.

- **Elaboração da Proposta:** com base nos resultados obtidos com o mapeamento sistemático da literatura, foram escolhidas as lacunas que o trabalho se propõe a solucionar e quais algoritmos seriam utilizados para testar a solução proposta. Três algoritmos metaheurísticos foram selecionados para resolver as instâncias de problemas utilizados neste trabalho: algoritmos genéticos, otimização por enxame de partículas e evolução diferencial. As instâncias escolhidas foram geradas a partir de dados de projetos reais e uma nova codificação real foi desenvolvida com o objetivo de evitar as deficiências encontradas nos métodos propostos na literatura.

- **Busca de Dados de Requisitos de Software para Avaliação da Proposta:** algumas empresas com as quais o autor possui contato foram contactadas com o objetivo de obter dados reais de projetos que pudessem ser utilizados para aplicar a solução proposta neste trabalho. Duas empresas concordaram em fornecer os dados de forma anônima,

ou seja, sem identificar os projetos, as empresas e o conteúdo textual dos requisitos. Ao todo foram disponibilizados dados de três projetos, dois de médio porte (com 143 e 246 requisitos) e um de pequeno porte (com 32 requisitos).

- **Implementação de Protótipo da Ferramenta de Software:** para viabilizar os testes dos algoritmos e conjuntos de dados utilizados neste trabalho, um protótipo de software foi desenvolvido. Este protótipo teve como objetivo a automatização dos testes que necessitavam ser realizados com cada configuração. As técnicas meta-heurísticas são dependentes dos parâmetros de configuração de cada algoritmo e estes devem ser executados com diferentes combinações destes valores até encontrar os melhores valores. Além dos resultados numéricos, é importante observar as curvas de convergência dos resultados, o protótipo foi desenvolvido levando este requisito em consideração.

- **Avaliação da Proposta:** Para avaliar se a abordagem proposta neste trabalho é capaz de obter resultados superiores aos obtidos manualmente, três estudos de caso foram definidos. O primeiro consistiu em tentar encontrar a solução exata para as bases de dados fornecidas pelas empresas. O segundo consistiu em realizar um experimento com alunos de graduação e mestrado da disciplina de Engenharia de Requisitos da Universidade Tecnológica Federal do Paraná (UTFPR) para analisar a qualidade dos resultados obtidos manualmente por eles. Por fim, os três algoritmos selecionados foram desenvolvidos em duas versões: uma binária com penalização e uma real com a nova codificação proposta. Todas as versões de cada algoritmo foram executadas para resolver cada uma das bases de dados obtidas e os resultados foram comparados entre si. Como as bases de dados são inéditas na literatura, não possuem um resultado ótimo conhecido, e a utilização de 3 algoritmos diferentes, cada um em outras duas variantes, contribui para a avaliação quantitativa da solução proposta.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está dividido em seis capítulos, organizados da seguinte forma: Introdução, neste Capítulo 1, com uma breve contextualização do tema, a motivação, justificativa, objetivos e forma de avaliação.

A fundamentação teórica (Capítulo 2) apresentamos as principais definições utilizadas ao longo do trabalho, como requisitos de software, interdependências entre requisitos, problema da mochila e meta-heurísticas para otimização. Além das definições também serão explicados os conceitos e o funcionamento de meta-heurísticas como Algoritmos Genéticos, Evolução Di-

ferencial e Otimização por Enxame de Partículas.

No Capítulo 3 apresentamos os principais trabalhos relacionados com esta área de pesquisa bem como as lacunas que foram encontradas nestes trabalhos.

O Capítulo 4 contém os detalhes da metodologia que foi aplicada para resolver o problema de seleção de requisitos, os algoritmos desenvolvidos, *datasets* propostos e a ferramenta de software desenvolvida para aplicação dos algoritmos e meta-heurísticas.

Os resultados gerados pelos métodos propostos e desenvolvidos no Capítulo 4 serão apresentados no Capítulo 5.

Por fim, o Capítulo 6 conclui o documento apresentando as conclusões gerais deste trabalho e propõe possibilidades de diferentes abordagens para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Apresentamos os principais conceitos necessários, além de uma contextualização para melhor compreensão deste trabalho. Neste capitulo são abordados os conceitos de requisitos de software, interdependências entre requisitos e as meta-heurísticas utilizadas neste trabalhos.

Por fim, são descritos o problema de mochila e como ele já foi adaptado na literatura para ser utilizado no problema de seleção de requisitos.

2.1 REQUISITO DE SOFTWARE

Apesar de ser um conceito bem básico, é importante definir requisito de software nesse trabalho, pois é o ponto central do problema sendo estudado. De acordo com (ISO/IEC/IEEE. . . , 2010) a definição dada é a seguinte:

> 1. Uma condição ou função desejada por um usuário para resolver um problema ou atingir um objetivo. 2. uma condição ou função que precisa ser cumprida ou possuída por um sistema, componente de sistema, produto, ou serviço para satisfazer um acordo, padrão, especificação, ou outro formalmente descrito em um documento. 3. uma representação documentada de uma condição ou capacidade como em (1) ou (2). 4. uma condição ou função que precisa ser cumprida ou possuída por um sistema, produto, serviço, resultado, ou componente para cumprir um contrato, padrão, especificação, ou outro formalmente imposto por um documento. Requisitos incluem as necessidades quantificadas e documentadas, desejos, e expectativas de um patrocinador, cliente, e outros *stakeholders*.

De uma forma geral, podemos dizer que um requisito de software é uma funcionalidade que deve ser implementada em um produto de software. Porém, está é uma definição simples, pois produtos de software não representam somente funções, mas também objetivos, propriedades e restrições que o sistema deve possuir para satisfazer as necessidades do usuário, como dito em (ISO/IEC/IEEE. . . , 2010).

Os requisitos podem ainda ser classificados da seguinte maneira:

- **Requisitos Funcionais:** são os requisitos que descrevem as funções que o sistema deve realizar. Cada requisito funcional deixa sua contribuição para a construção de uma parte do sistema.
- **Requisitos Não-Funcionais:** são requisitos que não dizem respeito diretamente à alguma funcionalidade do sistema, mas expressam propriedades ou restrições do sistema.
- **Requisitos de Domínio:** são requisitos derivados do domínio da aplicação, e não são obtidos diretamente da necessidade dos usuários, apesar disso, podem se transformar em requisitos funcionais do sistema.
- **Requisitos de Usuário:** são requisitos funcionais ou não-funcionais, mas do ponto de vista do usuário.
- **Requisitos de UX:** são requisitos de experiência de usuário, relacionados à experiência que sistema deve passar a seus usuários.

Apesar de todas essas classificações, o processo de seleção de requisitos é independente delas, pois todos os requisitos são tratados genericamente apenas como requisitos de software. Por esse motivo não há necessidade de diferenciar os requisitos neste trabalho, de modo que todos serão chamados apenas de requisitos.

Existe todo um processo que vai desde o levantamento e especificação dos requisitos até sua seleção e implementação. Este trabalho se propõe a auxiliar no processo decisório presente na fase de seleção de requisitos, sendo assim, assume que os requisitos já estão detalhados e especificados corretamente de acordo com a metodologia de desenvolvimento adotada pelos desenvolvedores.

A fase de seleção de requisitos está intimamente ligada ao planejamento de *release*, que consiste em determinar quais requisitos farão parte de um determinado produto de software, sem deixar de atender às restrições de tempo, orçamento e outros recursos (RUHE, 2001). Esta fase ocorre tanto em ambientes de desenvolvimento que utilizam metodologias ágeis de desenvolvimento de software, quanto em ambientes que utilizam metodologias tradicionais. Porém, ela se faz muito mais presente em ambientes ágeis, pois a cada iteração é lançada uma nova versão do produto com novos recursos e funcionalidades. Estas metodologias geralmente utilizam ciclos curtos de desenvolvimento, de 2 a 4 semanas, e portanto a seleção de requisitos acaba sendo feita com certa frequência (ALLIANCE, 2017).

2.2 INTERDEPENDÊNCIAS ENTRE REQUISITOS

Em um projeto real, apenas uma parcela dos requisitos é completamente independente dos outros. Dada uma lista de requisitos, existem interações entre eles, as quais neste trabalho serão denominadas de interdependências, seguindo a nomenclatura proposta em (CARLSHAMRE et al., 2001).

Carlshamre et al. (2001) realizaram uma pesquisa em cima de projetos reais a fim de levantar os tipos de interdependências que podem existir entre os requisitos. Ele estudou 5 projetos em uma ampla variedade, sendo o mais simples um projeto de uma pequena empresa que possui apenas esse produto; e o maior um projeto de grande porte de uma empresa de telecomunicações. Foram levantados 6 tipos de interdependências, que foram classificadas de acordo com a Tabela 1. Ainda na Tabela 1, a linha referente à interdependência OR representa um "ou exclusivo", ou seja, se R_1 for implementado no projeto, R_2 não pode ser implementado ou vice-versa.

Tabela 1: Lista de interdependências entre requisitos. Fonte: Adaptado de Carlshamre et al. (2001)

Tipo	Descrição
R_1 AND R_2	R_1 necessita de R_2 para funcionar, e R_2 necessita de R_1 para funcionar
R_1 REQUIRES R_2	R_1 necessita de R_2 para funcionar, mas não o contrário
R_1 TEMPORAL R_2	R_1 precisa ser implementado antes de R_2
R_1 CVALUE R_2	R_1 afeta o valor de R_2 para um stakeholder
R_1 ICOST R_2	R_1 altera o custo de implementação de R_2
R_1 OR R_2	Somente R_1 ou R_2 pode ser implementado, nunca os dois

Se todas essas relações forem levadas em consideração, o problema original fica mais complexo e difícil de resolver. A maioria dos estudos realizados sobre o tema considera apenas a relação de precedência, e somente poucos trabalhos consideram outros tipos mais complexos de interdependências (PITANGUEIRA et al., 2014).

2.3 MÉTODOS ANALÍTICOS PARA SELEÇÃO DE REQUISITOS

Segundo o trabalho (LEFFINGWELL; WIDRIG, 2000) a seleção de requisitos é um problema de decisão complexo, e por isso, várias técnicas analíticas já foram propostas na literatura, como o Processo Analítico Hierárquico (AHP) (KARLSSON; RYAN, 1997), Método de Soma Ponderada (DANESH et al., 2009) e Análise de Custo-Valor (KUKREJA et al., 2013).

A aplicação do AHP contempla as seguintes fases: estruturação dos critérios e alternativas; coleta de julgamentos; cálculo de prioridades; verificação da consistência do julgamento;

e, por último, o cálculo das prioridades globais das alternativas. A estruturação dos critérios consiste em modelar o problema de decisão numa estrutura hierárquica, a qual, partindo do objetivo principal, decompõe-se em vários critérios necessários ao alcance do objetivo, formando uma camada de critérios (KARLSSON; RYAN, 1997). O julgamento é realizado por meio da comparação pareada entre dois elementos de um mesmo nível. Os elementos são comparados a partir de uma matriz quadrada, cuja ordem é igual ao número de elementos subordinados ao nó imediatamente superior. Os elementos subordinados são dispostos na mesma ordem, formando as linhas e as colunas da matriz. O principal problema do AHP é o número de comparações necessárias para chegar a uma solução, se tornando impraticável para projetos com mais de 20 requisitos (ACHIMUGU et al., 2014). O número total de comparações necessárias é de $(n-1)/2$. Para um projeto de 100 requisitos, que ainda não é considerado um projeto de grande porte, seriam necessárias 4950 comparações (LIM; FINKELSTEIN, 2012).

Em uma situação ideal, na qual não existem dependências entre os requisitos, o planejamento pode ser facilmente realizado de acordo com as prioridades geradas por meio destas técnicas, apenas considerando os recursos e prazos disponíveis (SHAO et al., 2016). Entretanto, uma pesquisa realizada em 5 projetos reais mostrou que, em média, apenas 20% dos requisitos são independentes, o que está bem longe da situação ideal (CARLSHAMRE et al., 2001).

As revisões sistemáticas de literatura realizadas em (ACHIMUGU et al., 2014), (RIEGEL; DOERR, 2015) e (PITANGUEIRA et al., 2014) apresentaram o estado da arte das técnicas de priorização de requisitos e foram unânimes ao identificar as duas maiores deficiências das técnicas pesquisadas: escalabilidade e dependências entre requisitos.

Os métodos citados, também não levam em conta as dependências entre os requisitos, mesmo estes sendo um fator importante no resultado final e na ordem em que os requisitos serão implementados. Isso faz com que os resultados gerados por essas técnicas raramente possam ser utilizados diretamente no projeto, sem um pós processamento manual dos resultados (SHAO et al., 2016).

No geral, essas técnicas funcionam muito bem para projetos pequenos, com 10-30 requisitos e 5-6 *stakeholders* (FEHLMANN, 2008), e mesmo assim, apesar de existirem várias técnicas disponíveis, apenas algumas poucas delas são capazes de obter níveis aceitáveis de qualidade e eficiência em certas situações (SHER et al., 2014).

Não suficiente, tais técnicas também possuem dificuldade em tratar corretamente os aspectos associados ao Desenvolvimento Dirigido pelo Mercado (MDS), com o grande número de *stakeholders* suas importâncias individuais para o projeto e distribuição geográfica (SANTOS et al., 2016).

O cenário se agrava em ambientes ágeis de desenvolvimento de software, nos quais o processo de priorização de requisitos precisa ser refeito todas as vezes entre os ciclos de desenvolvimento (POPLI et al., 2014). Na prática, essa etapa é ignorada e quem decide quais requisitos serão implementados no próximo ciclo é o *Product Owner* (PO) ou o gerente do projeto, muitas vezes sem ter a visão global sobre os interesses de todos os *stakeholders* (DANEVA et al., 2013). Tal afirmação vem de um estudo de caso realizado em (DANEVA et al., 2013), com três projetos de uma madura e conhecida empresa, globalmente reconhecida pela sua excelência no ramo de desenvolvimento de software. O estudo consistiu em um questionário para identificar os responsáveis pelas tomadas de decisão em diversas fases do desenvolvimento de um software.

Além das dificuldades e limitações dos métodos em si conforme o número de requisitos aumenta esse processo pode ser demorado, cansativo e gerar stress entre 19 os *stakeholders*, comprometendo o processo (YOUSUF et al., 2016).

2.4 PROBLEMA DA MOCHILA 0-1

O Problema da Mochila é um problema de otimização combinatória de especial importância na Ciência da Computação, pois pertence à classe de problemas NP-Completos (GAREY, 1979). É também, um dos principais problemas da área de Pesquisa Operacional, visto que muitas situações nas quais se busca maximizar o lucro podem ser modeladas e resolvidas utilizando este problema. Sua resolução demanda um grande esforço computacional, tornando-se especialmente interessante para a área de Programação Paralela. O Problema de Mochila 0-1 é uma aplicação do KP original no qual só existe um item de cada tipo, e a resolução consiste em decidir se cada item será ou não incluído na solução.

2.4.1 DEFINIÇÃO INTUITIVA

O Problema da Mochila, ou *Knapsack Problem* (KP), pode ser definido intuitivamente por meio de um cenário. Suponha-se um ladrão que deseja roubar uma residência. Tudo de que o ladrão dispõe para carregar o produto de seu furto é uma mochila escolar, de porte médio. Existem objetos das mais variadas dimensões, com os mais variados valores (e.g., joias, roupas, eletrônicos, etc.). Caberá ao ladrão, portanto, escolher os itens a carregar na sua mochila de modo que o valor a ser roubado seja máximo e que ele possa carregar.

O Problema da Mochila, portanto, pode ser visto como o problema de preencher uma mochila de capacidade limitada com itens que possuem um valor associado de modo a não

exceder esta capacidade e maximizar o valor resultante.

2.4.2 DEFINIÇÃO FORMAL

De acordo com a definição formal do KP existem n tipos de itens. Cada $x_i \in [0,1]$ representa a quantidade de itens do tipo i, onde cada tipo tem associado um valor v_i e um peso p_i sendo que $i \in 1,2,...,n$. O máximo peso que a mochila pode carregar é C. Matematicamente o problema é definido pela Equação (1) (GAREY, 1979).

$$\begin{array}{lll} \text{maximizar} & \sum_{i=1}^{n} x_i v_i & \Big| \quad \sum_{i=1}^{n} x_i p_i \leq C \\ & \text{onde} \quad x_i = 0 \quad ou \quad x_i = 1 & \end{array} \tag{1}$$

2.4.3 COMPLEXIDADE

O KP é um problema NP-Completo, portanto, o é resolvível em tempo polinomial por um algoritmo não-determinístico. De fato, não há prova de que existe algoritmo que resolva o KP em tempo polinomial em uma máquina determinística (pertence à classe NP) e qualquer outro problema NP pode ser reduzido polinomialmente a um KP, sendo, assim, o problema NP-Completo (GAREY, 1979). Devido à sua complexidade exponencial, uma solução eficiente para o problema deve ser aquela que consegue reduzir, ao máximo, os termos da expressão exponencial.

2.5 *NEXT RELEASE PROBLEM*

Um problema presente em qualquer empresa de desenvolvimento e manutenção de software que é vendido para uma grande variedade de clientes é a seleção de quais requisitos deveriam ser implementados no próximo *release* (BAGNALL et al., 2001). De uma forma geral, a empresa se depara com:

- demandas de seus clientes para um grande número de melhorias no software;
- situações em que algumas melhorias necessitam de outras melhorias como pre-requisitos;
- clientes que possuem diferentes pesos e importâncias para a empresa;
- requisitos que podem demandar diferentes quantidades de tempo e esforço para serem implementados.

O desafio é selecionar um conjunto de requisitos que seja factível dentro de seu orçamento e que atenda às demandas de seus clientes mais importantes. Uma decisão errada nesta etapa pode gerar consequências graves para a empresa, como a perda de clientes e, até mesmo, a falência da empresa (BAGNALL et al., 2001).

O problema de seleção de requisitos foi modelado como um problema de otimização pela primeira vez em 2001 e batizado de *Next Release Problem* (NRP) (BAGNALL et al., 2001). Ao longo dos anos este modelo foi sofrendo modificações e adaptações, mas duas versões são as mais utilizadas na literatura e foram chamadas neste trabalho de "Modelo de Satisfação Total"e "Modelo de Satisfação Parcial".

2.5.1 MODELO DE SATISFAÇÃO TOTAL

Este é o modelo original proposto em 2001 por Bagnal e nele existe um conjunto de requisitos e um orçamento limitado, que não é suficiente para implementar todos os requisitos. Também existem *stakeholders* que possuem diferentes pesos para a empresa e cada um deseja um pequeno conjunto de funcionalidades. Caso todas as funcionalidades desejadas por um *stakeholder* sejam implementadas ele irá gerar um valor para a empresa. Caso uma das funcionalidades não seja implementada, aquele *stakeholder* não irá gerar valor algum (BAGNALL et al., 2001).

De acordo com a definição matemática existe um conjunto $R = [r_1, r_2, ..., n]$ de todos os requisitos de software desejados por todos os *stakeholders* $S = [s_1, s_2, ..., m]$. Cada *stakeholder*, j, tem um subconjunto de requisitos R_j desejados por ele, e um peso, w_j, que representa a importância que ele tem para a empresa. Associado a cada requisito r_i existe um custo de desenvolvimento $cost(r_i)$.

A tarefa é encontrar um subconjunto de *stakeholders*, $S' \in S$ que tenham todos seus desejos implementados. Através do conjunto S' é então possível encontrar o conjunto R' com todos os requisitos que serão implementados. Dessa forma, a empresa deseja maximizar a satisfação total de acordo com a Equação (2).

$$\text{maximizar} \quad \sum_{j=1...m}^{n} w_j x_j \quad \Bigg| \quad \sum_{j=1...m}^{n} cost(R_j) x_i \leq B \qquad \text{onde} \quad x_j \in [0,1] \tag{2}$$

Este modelo foi utilizado na literatura por vários anos, mas atualmente não reflete a

realidade de grande parte das empresas de desenvolvimento de software e foi aos poucos sendo trocado pelo "Modelo de Satisfação Parcial".

2.5.2 MODELO DE SATISFAÇÃO PARCIAL

A principal diferença que este modelo possui em relação ao Modelo de Satisfação Total é que não existe a necessidade de que todos os requisitos desejados por um *stakeholder* sejam implementados para que eles gerem um valor ao software.

Dado um projeto de software, existe um conjunto de *stakeholders* $S = [s_1, s_2, ..., s_m]$, e cada *stakeholder* s_j possui uma importância w_j que representa a relevância dele para a empresa.

Também existe um conjunto de requisitos $R = [r_1, r_2, ..., r_n]$ que devem ser implementados. Para implementar cada requisito, uma certa quantidade de recursos deve ser gasta, ou seja, um custo, dado por $cost_i$.

Vários *stakeholders* podem desejar o mesmo requisito, e nem todos requisitos têm a mesma importância para todos os *stakeholders*. Nesse caso, cada *stakeholder* dá um valor para cada requisito e esse valor é denominado por $value_{\mathrm{ij}}$, que representa o quanto os *stakeholders* desejam aquele requisito. Esse é o valor dado por cada *stakeholder*, mas para chegar no valor final de satisfação que esse requisito irá gerar caso seja implementado, é necessário considerar os pesos dos *stakeholders* para a empresa, e isso é feito através da Equação (3).

$$score_{\mathrm{i}} = \sum_{j=1}^{m} w_j value_{\mathrm{ij}} \tag{3}$$

Onde:

$score_{\mathrm{i}}$ valor de satisfação total que o requisito irá gerar caso seja implementado

m total de *stakeholders* associados ao projeto

w_j valor de importância do j-ésimo *stakeholder*

$value_{ij}$ valor dado ao i-ésimo requisito pelo j-ésimo *stakeholder*

O vetor decisão $x = [x_1, x_2, ..., x_n] \in [0, 1]$ indica quais requisitos serão incluídos no software. No caso de $x_i = 1$, o requisito será implementado, e no caso de $x_i = 0$, o requisito não será desenvolvido.

O orçamento do projeto é fixo e representado por B. A Equação (4) define esta versão do modelo matemático.

$$\text{maximizar} \quad \sum_{i=1}^{n} score_i x_i \quad \Bigg| \quad \sum_{i=1}^{n} cost_i x_i \leq B \qquad \text{onde} \quad x_i \in [0,1] \tag{4}$$

Onde:

$score_{\mathrm{i}}$ valor de satisfação total que o requisito irá gerar caso seja implementado

n total de requisitos do projeto

m total de *stakeholders* associados ao projeto

$cost_i$ custo de desenvolvimento do i-ésimo requisito

B orçamento disponível para o projeto

x_i variável binária que indica se o requisito será desenvolvido ou não

Este é o modelo que será utilizado neste trabalho, devido a melhor representar o atual cenário de da indústria desenvolvimento de software.

2.6 META-HEURÍSTICAS E OTIMIZAÇÃO COMBINATÓRIA

Problemas de otimização consistem em achar a melhor combinação dentre um conjunto de variáveis para maximizar ou minimizar uma função, geralmente chamada de função objetivo ou função custo. Esses problemas podem ser divididos em três categorias: aqueles cujas variáveis assumem valores reais (ou contínuos), aqueles cujas variáveis assumem valores discretos (ou inteiros) e aqueles em que há variáveis inteiras e contínuas, classificados, respectivamente, como problemas de Otimização Contínua, Otimização Combinatória ou Discreta, e Otimização Mista.

Dada a natureza do problema, a solução mais simples seria empregar uma busca exaustiva e encontrar a melhor solução. Porém, geralmente são problemas de alta complexidade, nos quais tal solução poderia levar dias, ou até meses de tempo de processamento (QASIM; ISMAIL, 2017). Na literatura muitos métodos foram sendo propostos para resolver tais problemas, porém não há garantia de que a solução encontrada seja a melhor. As soluções encontradas

por eles são muito próximas da solução ótima mas elas são encontradas em apenas uma pequena fração do tempo que os métodos exatos levariam. Cabe ressaltar, que nem todos os tipos de problema de otimização possuem um método exato para solucioná-lo que não seja uma busca exaustiva.

Dentre as técnicas aproximadas de otimização já desenvolvidas existe um grupo chamado de meta-heurísticas, que são aplicadas a problemas onde não se conhece um algoritmos eficiente para resolvê-lo. Elas utilizam combinações de soluções aleatórias e conhecimento histórico dos resultados anteriores já encontrados pelo algoritmo para se guiarem e realizarem suas buscas pelas melhores soluções. Meta-heurísticas são algoritmos aproximados que incorporam mecanismos para evitar confinamento em mínimos ou máximos locais. Conhecimentos específicos do problema podem ser utilizados na forma de heurística para auxiliar no processo de busca (por exemplo, na busca de um possível bom vizinho de um determinado ponto). Resumindo, pode-se dizer que meta-heurísticas são mecanismos de alto nível para explorar espaços de busca, cada uma usando um determinado tipo de estratégia (BECCENERI et al., 2008).

O balanço dinâmico entre diversificação e intensificação é de grande importância na aplicação de uma meta-heurística, fazendo uma distinção entre os termos ingleses *exploration* e *exploitation*. O primeiro pode-se traduzir por diversificação, exploração diversificada, busca em largura ou simplesmente exploração; o segundo por exploração focada, busca em profundidade ou intensificação. Um dos desafios na aplicação de uma meta-heurística é encontrar o equilíbrio ideal entre diversificação e intensificação. A estratégia de busca de uma meta-heurística depende da filosofia empregada por ela, e seu objetivo é escapar dos mínimos locais a fim de proceder a exploração do espaço de busca por soluções ainda melhores (BECCENERI et al., 2008).

A seguir serão detalhados as meta-heurísticas que serão aplicadas neste trabalho para a seleção de requisitos.

2.6.1 ALGORITMOS GENÉTICOS

Algoritmos Genéticos, ou *Genetic Algorithms* (GA) são uma família de métodos computacionais inspirados no conceito de evolução das espécies. Tipicamente, os GAs são usados na otimização de funções, codificando uma população de soluções candidatas (ou indivíduos), de um determinado problema em estruturas de dados organizadas de forma semelhante aos cromossomos. Após a escolha da população inicial, gerada aleatoriamente, ou escolhida arbitrariamente pelo analista em função de conhecimento *a priori* sobre o problema estudado, são então aplicados iterativamente procedimentos de seleção, recombinação e mutação a estas es-

truturas, criando assim uma geração subsequente que, em média, apresentará um desempenho melhor que a geração precedente (quando avaliados por uma função de *fitness*) que teve parte de suas características mais desejáveis preservadas e aperfeiçoadas por procedimentos específicos (HOLLAND, 1975).

A formulação básica do processo envolve dois estágios: (i) Seleção da população que irá procriar, formando uma população intermediária e (ii) aplicação dos operadores de reprodução (*crossover*) e mutação (*mutation*) a esta população intermediária. As escolhas dos esquemas de cruzamento e mutação, da forma de codificação dos indivíduos, da definição da função *fitness*, e a calibragem dos parâmetros dos operadores são os aspectos mais importantes de qualquer GA, pois são fortemente dependentes do problema analisado (LEE, 2008).

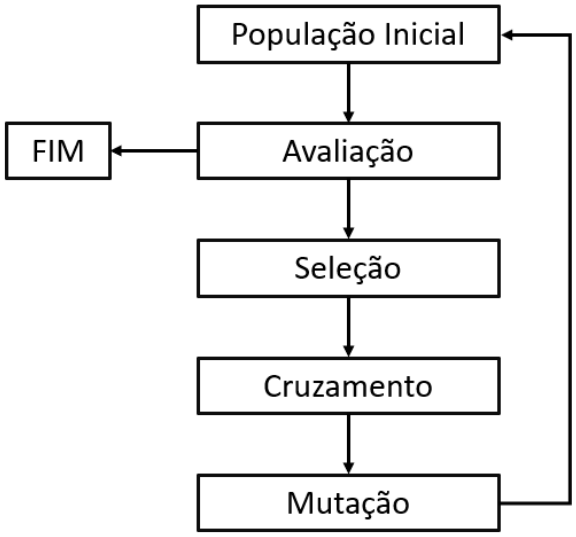
O primeiro passo no uso de GAs é escolher como representar cada indivíduo desta população, ou seja, como codificá-los, transformando cada parâmetro do problema por meio de um conjunto finito de símbolos. Uma escolha muito comum é a codificação binária, sendo que neste caso os indivíduos são representados como uma *string* de bits de comprimento mantido constante ao longo do processo de otimização. Esta solução, embora simples, apresenta dois problemas principais (LEE, 2008):

- Se o número de valores discretos válidos (permitidos) não for uma potência de dois, uma grande parcela das combinações possíveis, geradas no processo de criação de novos indivíduos, corresponderão a genes que não têm expressão em nenhum fenótipo, tornando muitas destes cromossomos redundantes, obrigando o analista a buscar métodos para o tratamento destas anomalias, convertendo códigos não-válidos em válidos.
- A distância entre duas soluções com representação binária, medida como o número de bits diferentes nos cromossomos, pode não corresponder à distância verdadeira entre as soluções. Por exemplo, dois números inteiros na base 10 podem ser vizinhos, mas terem vários bits diferentes na representação binária.

A avaliação da qualidade de cada indivíduo de uma dada geração – e, consequentemente, da probabilidade deste transmitir suas características à geração seguinte - é realizada por meio da função de *fitness*. Esta função distingue-se conceitualmente da função objetivo do problema de otimização, pois enquanto a primeira é utilizada para se avaliar a performance de uma determinada solução, a segunda traduz esta medida de performance como a alocação das oportunidades reprodutivas de determinado indivíduo. Contudo, em diversas aplicações, a própria função de *fitness* pode se prestar a esse fim (LEE, 2008).

A Figura 2 mostra as etapas de execução do GA, e as etapas principais serão detalhada em seguida.

Figura 2: Etapas de um GA



Fonte: Autoria Própria

2.6.1.1 SELEÇÃO

Na execução do GA, a etapa de seleção é responsável pela formação das populações intermediárias, ou seja, são selecionados os indivíduos mais aptos conforme a avaliação da função de *fitness*. Portanto, os indivíduos mais aptos têm uma maior probabilidade de contribuir com seu material genético na formação da próxima geração de soluções candidatas, enquanto que os menos adequados tenderão a desaparecer das gerações futuras. Ressalta-se que esta etapa não é capaz de introduzir novos indivíduos à população, ao contrário das etapas de cruzamento e mutação, que efetivamente são os responsáveis pela exploração do espaço solução. Não obstante, esta etapa é a principal responsável pelas características de convergência do GA pelo fato de determinar a pressão evolutiva sobre a população, calibrando a forma como os melhores indivíduos serão favorecidos (GOLDBERG; DEB, 1990).

Uma consequência direta do critério de seleção escolhido é a velocidade de convergência, onde métodos que implicam pouca pressão evolutiva tenderão a possuir convergência lenta, já que indivíduos menos aptos terão ainda uma chance elevada de contribuir para a geração subsequente, ao passo que uma pressão elevada provavelmente será seguida de uma velocidade de convergência muito elevada, que pode conduzir o GA a uma solução de baixa qualidade, devido à convergência prematura em um ótimo local ou estagnação da população. Sendo assim,

o esquema de seleção adotado deve procurar preservar a diversidade da população (LEE, 2008).

Uma opção muito utilizada na etapa de seleção, é o uso de Elitismo, ou seja, os *n* melhores indivíduos de uma geração são replicados na geração seguinte, onde *n* é um parâmetro livre. Contudo, se *n* for muito grande em relação à população, o processo de solução estará sujeito à convergência prematura (GOLDBERG; DEB, 1990).

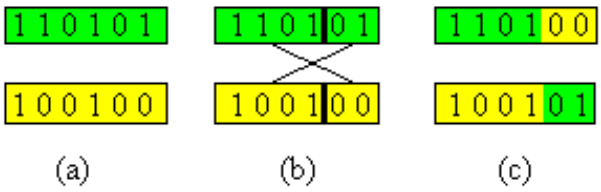
2.6.1.2 CRUZAMENTO

A etapa de cruzamento (também conhecida como *crossover*) é responsável pela criação de novos indivíduos a partir dos indivíduos selecionados para contribuírem com a próxima geração. Os genes, elementos utilizados na codificação de cada indivíduo, são fornecidos pelos pais para formar o filho, sendo necessário se estabelecer um critério de como será feita a contribuição de cada pai.

As estratégias mais utilizadas nesta etapa são:

- **um-ponto:** um ponto de cruzamento é escolhido e a partir deste ponto as informações genéticas dos pais serão trocadas. As informações anteriores a este ponto em um dos pais são ligadas às informações posteriores à este ponto no outro pai, como é mostrado no exemplo da Figura 3.
- **multi-pontos:** é uma generalização da ideia de troca de material genético através de pontos, onde muitos pontos de cruzamento podem ser utilizados.
- **uniforme:** não utiliza pontos de cruzamento, mas determina, através de um parâmetro global, qual a probabilidade de cada variável ser trocada entre os pais.

Figura 3: Exemplo de *crossover* de um-ponto



Fonte: Autoria Própria

2.6.1.3 MUTAÇÃO

A etapa de mutação é responsável pelo aumento da diversidade da população através da alteração aleatória de um ou mais genes de um indivíduo com uma probabilidade denomi-

nada taxa de mutação. Assim, há a introdução de um novo indivíduo na população que possui características bastante diversas de seus pais (GOLDBERG; DEB, 1990).

A importância deste operador está na prevenção da convergência prematura do GA ao mesmo tempo em que permite a exploração do espaço de busca em regiões afastadas da região onde a maioria da população se encontra, o que pode ser interessante quando não se conhece exatamente a topologia da função de *fitness* e há a possibilidade de existirem mínimos locais próximos ao mínimo global. Isto torna este operador mais importante quando o GA está próximo das gerações finais, quando quase todos os indivíduos apresentam qualidade similar (GOLDBERG; DEB, 1990).

No caso de codificação binária, a mutação é realizada simplesmente se invertendo um dos bits do cromossomo aleatoriamente. Para codificação com números reais, o valor do gene afetado é substituído por outro, gerado aleatoriamente ou por perturbação do valor original, ao qual é somado um ruído gerado aleatoriamente, condicionado a alguma distribuição (GOLDBERG; DEB, 1990).

Na calibração do GA deve-se coordenar o peso relativo entre as estratégias de cruzamento e mutação, que têm influências opostas no processo de solução. Enquanto um GA dominado pelo operador cruzamento tende a convergir prematuramente, devido à falta de diversidade da população, um GA dominado pelo operador mutação implicaria na perda de informação importante da geração anterior sobre a aptidão dos indivíduos (GOLDBERG; DEB, 1990).

2.6.2 EVOLUÇÃO DIFERENCIAL

A Evolução Diferencial tem se apresentado como um simples, mas poderoso algoritmo de otimização numérica para busca da solução ótima global, sendo aplicado com sucesso na solução de vários problemas de difícil otimização (CHENG; HWANG, 2001). Foi proposta como uma alternativa dos algoritmos genéticos para uma mais rápida otimização, a Evolução Diferencial tem gradualmente recebido crescente interesse e se tornado um algoritmo cada vez mais popular nos últimos anos, sendo utilizado em muitos casos práticos, principalmente por apresentar boas propriedades de convergência e de fácil entendimento.

A escolha pelo algoritmo de Evolução Diferencial é baseada em algumas de suas características úteis, como: não requerer informações da derivada da função objetivo, que nem sempre é possível obter uma expressão analítica para derivada desta; apresentar menor probabilidade de se tornar preso a um ótimo local, pois busca a solução ótima global através da

manipulação de uma população de soluções candidatas, ou, equivalentemente, busca simultaneamente a solução em diferentes regiões no espaço de busca e por não ser necessário trabalhar com grandes tamanhos de população (CHENG; HWANG, 2001).

A Evolução Diferencial é uma técnica estocástica de busca direta, baseada nos mecanismos da evolução natural das espécies, utilizando de procedimentos de seleção de modo organizado baseados na aptidão dos indivíduos e operadores de mutação e cruzamento. O método utiliza vetores de n dimensões como uma população para cada geração e consiste em gerar aleatoriamente uma população de indivíduos onde cada indivíduo representa um ponto de busca no espaço de soluções (STORN; PRICE, 1997).

Dada uma população inicial escolhida aleatoriamente, cobrindo todo o intervalo de busca, esta sofre modificações gerando novos indivíduos pela adição da diferença ponderada entre dois indivíduos aleatórios da população original a um terceiro indivíduo escolhido também aleatoriamente da população original. Esta operação é chamada de mutação. Os componentes do vetor da mutação são então combinados com os componentes de um outro vetor escolhido aleatoriamente, denominado de vetor alvo, para produzir o vetor chamado de vetor teste. Este processo de combinação é muitas vezes referido como cruzamento (*crossover*). Se o vetor teste gera um valor melhor da função de *fitness* que o vetor alvo, o vetor teste substitui o vetor alvo na geração seguinte. Esta última operação é chamada seleção (STORN; PRICE, 1997).

O processo de busca da melhor solução consiste na evolução da população ao longo das gerações. O objetivo deste procedimento é gerar uma população melhorada com indivíduos mais aptos. Acredita-se que o indivíduo mais evoluído represente a solução ótima do problema, sendo sua aptidão avaliada pelos valores da função de *fitness*. O procedimento de otimização termina quando um numero máximo de gerações é alcançado ou até que algum critério de parada seja cumprido.

O algoritmo é iniciado gerando uma população de N indivíduos representados como um vetor das variáveis de decisão. A população inicial é então gerada aleatoriamente dentro dos limites definidos pelo usuário, seguindo uma distribuição uniforme, conforme a Equação (5).

$$x_i^0 = Lim_{\inf}(x_i) + rand_i(Lim_{\sup}(x_i) - Lim_{\inf}(x_i)) \tag{5}$$

Onde:

x_i^0 é o i-ésimo indivíduo da população inicial, sendo que $i = 1, 2, ..., N$

$Lim_{\inf}(x_i)$ são os valores dos limites inferiores de cada variável de decisão

$Lim_{\sup}(x_i)$ são os valores dos limites superiores de cada variável de decisão

$rand_i$ é um número aleatório entre 0 e 1

Uma vez gerada a população inicial, a função objetivo de cada indivíduo da população inicial é calculada e armazenada para comparação futura. Na evolução do algoritmo, cada indivíduo pode sofrer alterações de uma geração para outra através da aplicação dos operadores: de mutação, cruzamento e de seleção.

2.6.2.1 MUTAÇÃO

Nesta operação são escolhidos aleatoriamente três vetores distintos entre todos os N indivíduos da população corrente. Um vetor escolhido aleatoriamente X_{r1}^{k} sofre uma perturbação resultante da diferença vetorial entre dois outros vetores diferentes X_{r2}^{k} e X_{r3}^{k}, escolhidos aleatoriamente e multiplicados pelo fator de perturbação F (STORN; PRICE, 1997). Para cada indivíduo da população corrente, um vetor de mutação $V^{\mathrm{k+1}}$ é gerado de acordo com a Equação (6).

$$v_{\mathrm{i,j}}^{\mathrm{k+1}} = x_{\mathrm{r1,\,j}}^{k} + F(x_{\mathrm{r2,\,j}}^{k} - x_{\mathrm{r3,\,j}}^{k}) \tag{6}$$

Onde:

k é o número da geração

i é o número do índice na população, deve ser um número entre $1...N$, sendo N o número total de indivíduos na população

j é o número do índice da variáveis de decisão, que fica no intervalo de $1...D$, sendo D o número total de variáveis de decisão

$v_{\mathrm{i,j}}^{\mathrm{k+1}}$ j-ésima variável de decisão do i-ésimo indivíduo da k-ésima geração

$x_{\mathrm{r1,\,j}}^{k}, x_{\mathrm{r2,\,j}}^{k} e x_{\mathrm{r3,\,j}}^{k}$ são as j-ésimas variáveis de decisão de três membros da população

F é um fator de perturbação que define a mutação e controla a amplitude da diferença vetorial.

Os elementos $x_{\mathrm{r1,\,j}}$, $x_{\mathrm{r1,\,j}}$ e $x_{\mathrm{r1,\,j}}$ são três vetores distintos escolhidos aleatoriamente entre todos os N indivíduos da geração k.

2.6.2.2 CRUZAMENTO

Com o objetivo de aumentar a diversidade da população, Storn e Price introduziram o operador de cruzamento. A operação *crossover* é usada para gerar um vetor de teste $U^{\mathrm{k+1}}$, onde esse é obtido substituindo componentes do vetor alvo X_t^k, escolhido aleatoriamente e diferente de $x_{\mathrm{r1,\,j}}$, $x_{\mathrm{r1,\,j}}$ e $x_{\mathrm{r1,\,j}}$, por componentes correspondentes do vetor gerado aleatoriamente na mutação $v^{\mathrm{k+1}}$. A taxa de *crossover* C_r determina quando um parâmetro deve ser substituído (STORN; PRICE, 1997). Neste trabalho o tipo de cruzamento utilizado foi o denominado de cruzamento binomial, e as componentes que formam o vetor teste $u_{\mathrm{i,j}}^{\mathrm{k+1}}$ são escolhidas da seguinte forma:

$$u_{\mathrm{i,j}}^{\mathrm{k+1}} = \begin{cases} v_{\mathrm{i,j}}^{\mathrm{k+1}} \text{ se } r \leq C_r \\ x_{\mathrm{i,j}}^{\mathrm{k}} \text{ se } r > C_r \end{cases} \tag{7}$$

Sendo:

r número aleatório com distribuição uniforme entre 0 e 1

C_r fator que define o cruzamento (*crossover*), fornecido pelo usuário

$v_{\mathrm{i,j}}^{\mathrm{k+1}}$ componentes do vetor de mutação

$x_{\mathrm{i,j}}^{\mathrm{k}}$ componentes do vetor alvo

2.6.2.3 SELEÇÃO

Na seleção a função objetivo do vetor teste é calculada e comparada com a do vetor alvo correspondente. Se o valor da função objetivo do vetor alvo da geração anterior $f(X_t^k)$ for melhor que a $f(U^{\mathrm{k+1}})$ do vetor teste, o vetor alvo antigo é permitido avançar para a próxima geração G+1, caso contrário o valor alvo é substituído pelo vetor teste na próxima geração (STORN; PRICE, 1997).

2.6.2.4 PARÂMETROS

Em contraste com a maioria dos algoritmos evolutivos, nos quais muitos parâmetros precisam ser ajustados, a evolução diferencial apresenta apenas três parâmetros a serem ajustados, são eles: tamanho da população, fator de perturbação e probabilidade de cruzamento. Desde que o método foi proposto muitas pesquisas no meio científico vem sendo realizadas com o objetivo de analisar a melhor região de escolha destes parâmetros, dentro da pré-estabelecida

na literatura. Na maioria das vezes a probabilidade de cruzamento C_r deve ser considerada menor que um, entretanto se a convergência não for alcançada utilizar um fator entre 0,8 e 1 pode ser conveniente. Para muitas aplicações um tamanho de população N de $10 * D$ é uma boa escolha, sendo D igual a dimensão ou número de variáveis. Já o fator de cruzamento é usualmente escolhido no intervalo [0.5 ,1]. Quanto maior for o número da população escolhida menor deve ser o fator de perturbação (CHENG; HWANG, 2001).

Nas resoluções de aplicações práticas é importante que se adquira maior conhecimento sobre como escolher as variáveis de ajuste para um determinado tipo de problema, ou seja, uma estratégia pode funcionar adequadamente para um tipo de problema, mas não tão bem para outro (STORN; PRICE, 1997).

2.6.3 OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS

A expressão Enxame de Partículas ou *Particle Swarm Optimization* (PSO) refere-se a um método de otimização desenvolvido na década de 1990 que, à semelhança do GA apresentado na seção anterior, também trabalha com um conjunto de soluções iniciais candidatas, geradas aleatoriamente, assumidas como sendo próximas ao ponto ótimo. Ao longo de um processo iterativo de solução, estas soluções candidatas aproximam-se idealmente da solução ótima. Ressalta-se que este método, assim como o GA, também não necessita de informações externas ao próprio método para a sua aplicação, tais como derivadas da função objetivo ou condições de continuidade ou convexidade, sendo a função objetivo utilizada para se avaliar a qualidade da solução encontrada por cada partícula (KENNEDY; EBERHART, 1995).

Destaca-se também que muitas vezes, a PSO é um método alternativo ao GA, utilizado com sucesso em aplicações nas quais se deseja velocidade de convergência, com a vantagem potencial de que este método possui um número menor de parâmetros de ajuste quando comparado ao GA (LEE, 2008).

A origem do método PSO remonta às pesquisas em vida artificial, que consistiam no estudo do comportamento do tipo enxame observado em algumas espécies e na modelagem computacional deste comportamento. (REYNOLDS, 1987) desenvolveu o sistema *boid* demonstrando como um conjunto simples de regras poderia ser usado para gerar um padrão bastante complexo de comportamento de enxame. Baseados nestes trabalhos anteriores, bem como na simulação do comportamento grupal de aves, Kennedy e Eberhart propuseram uma metodologia de otimização similar ao procedimento adotado pelas mesmas para a localização de alimento (KENNEDY; EBERHART, 1995). Conceitualmente, pode-se classificar PSO como pertencendo à família Inteligência de Enxame (*swarm intelligence*) (BONABEAU et al., 2001).

No PSO cada solução candidata voa por meio do espaço de busca modelada por vetores posição e velocidade, sendo que os primeiros vetores representam as soluções candidatas em si, enquanto que os segundos são empregados na regra de modificação destas soluções, que é por sua vez orientada pela combinação do comportamento individual e social dos grupos de aves anteriormente citados, otimizando assim uma determinada função objetivo (KENNEDY; EBERHART, 1995).

Em termos qualitativos, pode-se descrever a metodologia PSO da seguinte forma:

- Cada partícula é avaliada por meio da função objetivo;
- São definidos vetores de correção da velocidade e direção de cada partícula em função da melhor solução individualmente encontrada e da melhor solução encontrada por qualquer partícula vizinha;
- Atualiza-se a posição de cada partícula da população.

Logo, a partícula que atingiu a melhor solução em uma dada iteração atrai as demais, de sua vizinhança, na sua direção (socialização), contudo, a individualidade de cada uma é representada pela memória (coordenadas e valor da função objetivo) da melhor solução atingida até aquele momento (KENNEDY; EBERHART, 1995).

No PSO, N é o número de partículas da população e cada partícula possui os seguintes atributos:

X_i : coordenadas da posição da partícula

V_i : velocidade da partícula

Y_i : melhor posição já encontrada pela partícula

Logo a população P é representada pela Equação (8):

$$P(k) = [X_i(k), X_2(k), ..., X_n(k)] \tag{8}$$

Onde cada partícula é um vetor de dimensão m, sendo que m é o número de variáveis da função objetivo, assim em cada iteração k, tem-se a posição de cada partícula i dada pela Equação (9).

$$X_i(k) = [X_i1(k), X_i2(k), ..., X_im(k)] \tag{9}$$

A velocidade de cada partícula, em cada iteração i é representada pelo vetor $V(t)$ da Equação (10):

$$V_i(k) = [V_i1(k), V_i2(k), ..., V_im(k)] \tag{10}$$

Pode-se então apresentar a formulação básica do PSO, formalizada pela Equação (11).

$$V_i^{\mathrm{k+1}} = wV_i^k + c_1 rand_1(pbest - s_i^k) + c_2.rand_2(gbest - s_i^k) \tag{11}$$

Onde:

$V_i^{\mathrm{k+1}}$ é a velocidade da partícula i na iteração k

w é uma função de ponderação

$c_1 e c_2$ são coeficientes de ponderação

$rand$ é um número aleatório de 0 a 1

s_i^k é a posição da partícula i na iteração k

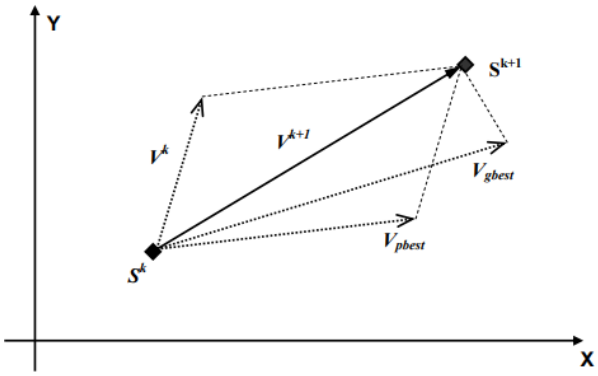
$pbest$ representa as coordenadas do melhor resultado já encontrado pela partícula i

$gbest$ representa as coordenadas do melhor resultado encontrado por qualquer partícula

A Figura 4 mostra graficamente como as componentes atuam para determinar a nova posição que a partícula assumirá na próxima iteração.

Analisando-se ainda a Equação (11) junto da Figura 4, se observa que a parcela wV_i^k corresponde ao vetor velocidade atual da partícula, atuando no sentido de permitir que esta explore novas áreas do espaço solução, portanto, esta parcela representa o grau de diversificação do procedimento de busca e como este varia ao longo do processo iterativo, tendo uma grande influência nas iterações iniciais e diminuindo conforme o número de máximo de iterações é atingido. Ou seja, ao final do processo, as partículas terão suas trajetórias determinadas quase exclusivamente pelos melhores resultados individuais e globais encontrados. Ressalta-se ainda que as outras duas parcelas $c_1 rand_1(pbest - s_i^k)$ e $c_2.rand_2(gbest - s_i^k)$ também conhecidas respectivamente como V_pbest e V_gbest.

Figura 4: Atualização da posição de uma partícula



Fonte: Autoria Própria

2.7 CONSIDERAÇÕES SOBRE ESTE CAPÍTULO

Ao longo desta fundamentação, é possível observar que os métodos analíticos existentes na literatura para seleção de requisitos são complexos de se utilizar em projetos de grande porte, gerando a necessidade de outros métodos para auxiliar no processo decisório. Por ser um processo complexo uma alternativa é a utilização de um sistema computacional para resolver o problema, e para isso é necessário modelá-lo de acordo.

Foram descritos o problema de mochila, as técnicas de otimização utilizadas neste trabalho, o funcionamento dos algoritmos e suas particularidades.

Por fim foram apresentados os modelos matemáticos mais utilizados na literatura para modelar o problema de seleção de requisitos como um problema de otimização: o modelo de satisfação total e o de satisfação parcial. No modelo de satisfação total, cada *stakeholder* possui um conjunto de requisitos que deseja implementar no software e caso todos sejam implementados eles geram um valor de satisfação para com o software. A satisfação de todos os *stakeholders* atendidos é somada ao final para determinar qual foi o valor total de satisfação obtido pelo software. Já no modelo de satisfação parcial, cada requisito gera uma pequena parcela de satisfação de acordo com o valor dado por cada *stakeholder*. Neste caso não há necessidade de implementar todos os requisitos desejados pelo *stakeholder* porém, quanto maior o número de requisitos desejados por ele forem implementados, maior será o valor de satisfação gerado.

O capítulo que se segue mostrará trabalhos relacionados sobre seleção de requisitos e quais as técnicas utilizadas na solução do problema.

3 TRABALHOS RELACIONADOS

Está seção trata sobre os principais trabalhos relacionados à seleção de requisitos como um problema de otimização. O objetivo é mostrar os trabalhos mais relevantes e também como esta área de pesquisa evoluiu desde que o problema foi modelado pela primeira vez como um problema de otimização.

De uma forma geral é possível observar que a modelagem do problema com uma visão multi-objetivo tem crescido com relação à visão de objetivo simples e que as meta-heurísticas são as técnicas mais eficientes. Entretanto não existe um consenso sobre qual é a melhor meta-heurística para resolver o problema, pois diferentes técnicas já foram aplicadas com sucesso na literatura (PITANGUEIRA et al., 2014), como será explicado a seguir.

3.1 MODELAGEM

Bagnal, Reyward-Smith e Whittley foram os primeiros a modelar o problema de seleção de requisitos como um problema de otimização. Desde então o problema tem sido explorado e estudado, seja na sua forma original ou modificada, por diversos pesquisadores (BAGNALL et al., 2001).

Em sua formulação original, ele foi modelado como sendo um problema de objetivo simples, visando maximizar o lucro obtido de *stakeholders* que solicitaram determinados requisitos. Nesse modelo o lucro a ser gerado por um *stakeholder* só é obtido caso todos os requisitos solicitados por ele sejam implementados. É um modelo que pode ser observado em empresas que desenvolvem grandes softwares corporativos, nas quais a implementação dos requisitos solicitados por um *stakeholder* irá resultar na aquisição de novas licenças do produto sendo desenvolvido.

Já o modelo mais utilizado na literatura recente, proposto por Greer e Ruhe, considera que nem todas as solicitações de um *stakeholder* precisam ser atendidas e, o que deve ser maximizado é o grau de satisfação a ser obtido com a implementação dos requisitos. Este modelo

reflete um cenário mais comum de desenvolvimento de software e poderia ser usado para modelar de uma forma mais realista o envolvimento dos futuros usuários do produto. Eles podem não gerar um lucro direto, mas são parte vital do processo de aceitação e do sucesso do novo produto (GREER; RUHE, 2004).

3.2 MÉTODOS

O método mais popular para resolver esses problemas são os algoritmos genéticos com codificação discreta. Isto funciona bem para seleção de requisitos independentes, mas para casos em que interdependência entre os requisitos precisa ser levada em conta, as soluções geradas devem ser reparadas e uma abordagem comum é criar critérios para isso, como por exemplo, remover ou adicionar itens da esquerda para a direita até a solução se tornar válida. Tal abordagem pode gerar soluções tendenciosas e fazer o algoritmo convergir para um máximo local.

A codificação binária é mais óbvia para este problema, pois no problema de mochila é um bit para cada item que decide se o item será ou não incluso. Apesar da utilização de números reais aumentar o tamanho dos cromossomos e a dificuldade de os algoritmos tratarem problemas deste tipo aumentar, trabalhos passados mostraram que o uso de codificação real em problemas binários pode levar à obtenção de melhores soluções (KRAUSE; LOPES, 2013).

3.2.1 EVOLUÇÃO DIFERENCIAL

O algoritmo de Evolução Diferencial, foi utilizado com sucesso no problema de seleção de requisitos, mas em poucos trabalhos, sempre em formulações multi-objetivo (CHAVES-GONZáLEZ; PÉREZ-TOLEDANO, 2015) (KUMARI et al., 2013). Em um deles o algoritmo foi adaptado de modo a incorporar aspectos da computação quântica para melhorar a diversidade do fronte de Pareto (KUMARI et al., 2013).

De acordo com a revisão sistemática apresentada no Apêndice A, o algoritmo DE também é um dos mais utilizados para a resolução do KP. A maioria dos estudos levantados no Apêndice A, utilizando DE para resolver o KP, utilizaram versões binárias do algoritmo original (DENG et al., 2009) (PENG et al., 2008) (JUN; JIAN, 2009) (HOTA; PAT, 2011) (GUO et al., 2014) (ZOUACHE; MOUSSAOUI, 2015) (LIBAO et al., 2016a) (PENG et al., 2016), fazendo uso de uma função sigmoide para converter os valores contínuos gerados pelo algoritmo em valores binários.

Como o algoritmo de evolução diferencial foi concebido inicialmente para ser utili-

zado exclusivamente em problemas com variáveis contínuas (STORN; PRICE, 1997), ele é uma boa opção para ser utilizado no problema de seleção de requisitos, desde que o problema seja codificado de forma a fazer uso de variáveis contínuas. Uma de suas desvantagens é que dependendo do tipo de problema ele pode convergir muito lentamente ou levar à máximos locais. Sua versão binária já foi utilizada para seleção de requisitos por, mas utilizando critérios de reparo das soluções que funcionou muito bem no problema pequeno estudado, mas que podem levar à resultados tendenciosos em problemas reais e maiores (CHAVES-GONZáLEZ; PÉREZ-TOLEDANO, 2015).

Já no problema da mochila, a evolução diferencial foi relativamente bem explorada e para trabalhar as deficiências inerentes do algoritmo já foram propostas variações utilizando novas estratégias de *crossover* em (WAGDY; ALMAZYAD, 2017), múltiplas estratégias de mutação em (XIANG et al., 2015) ou mutações híbridas em (LIBAO et al., 2016b).

3.2.2 OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS

Os algoritmos baseados em Colônias de Formigas(ACO) também foram bastante explorados em ambos os problemas, principalmente na seleção de requisitos, no qual é o segundo algoritmo bio-inspirado mais utilizado. Ele é inspirado no comportamento natural das formigas na busca por boas fontes de alimento.

A comunicação entre formigas é indireta, feita a partir de trilhas de feromônio. Quando uma formiga encontra uma fonte de comida, deixa uma trilha de feromônio para servir de guia para as demais formigas. Quanto melhor for a trilha, maior será a quantidade de feromônio e melhor é a fonte de alimento.

O ACO é bastante utilizado na sua formulação original ou em adaptação com outros métodos, como o Ant-Q que é um algoritmo desenvolvido a partir do algoritmo original e da técnica de aprendizado por reforço Q-Learning. Também já foi proposto como uma boa técnica de seleção de requisitos a ser explorada em ambientes iterativos de desenvolvimento de software (SAGRADO et al., 2015).

3.2.3 OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS

Existem vários tipos de algoritmos de inteligência por enxame, sendo o mais comum o algoritmo de Otimização por Enxame de Partículas, tanto que é o segundo método mais utilizado no problema de mochila. Tais algoritmos foram pouco ou nunca explorados no NRP e essa seria mais uma lacuna para pesquisas. O PSO em especial poderia ser mais explorado no NRP,

dado o alto volume de trabalhos publicados no problema de mochila com ele, e visto que, na seleção de requisitos, ele só foi utilizado uma vez, e com bons resultados (CHAVES-GONZáLEZ; PÉREZ-TOLEDANO, 2015).

O *Greedy Particle Swarm Optimization* (GPSO), proposto em (HE, 2010), combina um tipo de transformação baseada em algoritmos gulosos com uma versão binária do PSO para obter o melhor resultado (até a data de sua publicação) de uma famosa instância do KP.

Outra variação, batizada de *Socio-Cognitive Particle Swarm Optimization* (SCPSO) e proposta em (DEEP; BANSAL, 2008), introduzindo o conceito de distância entre a melhor solução global e a melhor solução de cada indivíduo. Com isso, uma nova equação de atualização da velocidade foi proposta, implementando pesos de inércia e um parâmetro de escala. Os resultados obtidos foram melhores que os do *Binary Particle Swarm Optimization* (BPSO) original.

Em uma nova versão do PSO, denominada *Multi Mutation Binary Particle Swarm Optimization* (MBPSO), dois novos operadores de mutação foram introduzidos no PSO: um operador de mutação microscópica, que altera o valor de apenas um dos bits da partícula e um operador de mutação macroscópica, que altera todos os bits de uma partícula. Entretanto, o método utilizado permite a geração de soluções inválidas no grupo e estas são tratadas utilizando um sistema de penalizações (LI; LI, 2009). Outro trabalho utilizou uma estratégia muito parecida, porém quando o operador de mutação macroscópica reinicializa a partícula, ele o faz com uma certa probabilidade de dissipação, atuando ao mesmo tempo como mutação e um sistema de busca local (WANG et al., 2009).

Outras versões do PSO ainda podem ser encontradas na literatura, como o *New Hybrid Binary Particle Swarm Optimization* (NHBPSO), que adiciona o operador de *crossover* dos algoritmos genéticos ao PSO (GHERBOUDJ; SAID, 2012). Uma modificação em que quando uma partícula atinge a mesma posição do máximo global, ela é totalmente reinicializada para outro valor só se mostrou eficiente para problemas com poucos itens (OUYANG; WANG, 2012). Por fim o PSO foi combinado com o algoritmo *Firefly Algorithm* (FA) para formar um novo algoritmo cooperativo (ZOUACHE et al., 2017).

3.2.4 OUTROS

Em 2002, foi proposto um algoritmo evolucionário baseado nos conceitos da computação quântica, chamado de *Quantum Evolutionary Algorithm* (QEA), introduzindo um operador de variação para promover a otimização dos indivíduos (HAN; KIM, 2003). Esse algoritmo ser-

viu de inspiração para a criação de uma versão quântica do PSO, batizada de *Quantum Swarm Evolutionary Algorithm* (QSEA), que se mostrou superior ao QEA tanto em velocidade quanto em qualidade das soluções (WANG et al., 2005).

Outra versão inspirada no comportamento quântico em que uma função variável foi introduzida no lugar da função tradicional de atualização, com o objetivo de evitar que o algoritmo fique estagnado em um máximo local (LI, 2013).

Uma terceira versão quântica teve como principal melhoria o sistema de busca local, no qual no momento que uma nova posição ótima é atingida, cada item do conjunto é adicionado ou removido individualmente de modo encontrar uma solução melhor (HADDAR et al., 2016).

Outros algoritmos menos convencionais já foram utilizados com sucesso no problema de mochila, como o algoritmo baseado no comportamento dos vagalumes. É um algoritmo de inteligência de enxames inspirado pelo brilho dos vagalumes na natureza. O brilho de um vagalume tem relação direta com a qualidade da solução que ele representa e quanto mais brilhante o vagalume, mais ele atrai outros vagalumes, ou seja, melhora outras soluções existentes na população (BHATTACHARJEE; SARMAH, 2015).

3.3 LACUNAS

Uma grande lacuna nessa área de pesquisa é a carência de bases de dados de projetos reais. Apenas duas bases públicas de projetos reais foram encontradas na literatura. Um projeto real foi disponibilizado em (BAKER et al., 2006), de um projeto da Motorola que possui 35 requisitos, mas todos os *stakeholders* tinham o mesmo peso para a empresa e portando não existe priorização dos mesmos. O outro foi disponibilizado em (LIM; FINKELSTEIN, 2012) e diz respeito a um projeto realizado na universidade de Londres (*University College London*), com 142 requisitos, 77 *stakeholders*, e interdependências do tipo mutualidade e exclusão apenas. Todos os outros *datasets* utilizados nos trabalhos foram gerados aleatoriamente, ou minerados a partir de repositórios de *bugs* de projetos *open source*.

Os *datasets* minerados foram propostos (DUAN et al., 2009) e resolvidos como um problema de otimização de seleção de requisitos pela primeira vez por (XUAN et al., 2012). Neles, cada *bug* foi considerado um requisito, e cada usuário que fez algum comentário no mesmo foi considerado um *stakeholder* que deseja esse requisito. O maior problema desse tipo de base de dados é que elas não possuem nenhum tipo de priorização entre os *stakeholders* nem interdependências entre os requisitos, ou seja, não condizem bem com um projeto real de software.

A inexistência de uma codificação robusta para problemas com interdependências e que sempre gere soluções válidas na população é também uma lacuna nesta área de pesquisa.

Na literatura estudada, nenhum trabalho explorou o NRP utilizando algoritmos concebidos para problemas de espaço contínuo sem adaptar os algoritmos para usar codificação binária, explorar outros métodos mais robustos com o uso de codificação real é uma lacuna de estudo presente na literatura.

3.4 CONSIDERAÇÕES SOBRE ESTE CAPÍTULO

Segundo abordado anteriormente o NRP pode ser interpretado como um problema de otimização e resolvido utilizando técnicas diferentes técnicas de otimização. Dentre estas técnicas, as mais utilizadas foram GA, DE, PSO e ACO, seja em suas formulações originais ou adaptados de acordo com novas formulações propostas. Apesar do NRP ser uma aplicação do KP, observou-se que algumas técnicas que foram aplicadas nele acabaram sendo pouco e nunca testadas com o NRP, deixando uma brecha a ser explorada na literatura, juntamente de outras que foram explicadas ao longo do capítulo.

Nos estudos mencionados é possível observar uma das maiores lacunas presentes na literatura são a falta de bases de dados públicas de requisitos de software baseadas em projetos reais e modelagens que não consideram as interdependências entre os requisitos. As bases de dados públicas que existem são geradas aleatoriamente, e apenas algumas possuem o conjunto completo de interdependências entre os requisitos.

O capítulo que segue irá mostrar como foi o desenvolvimento do projeto de pesquisa, atuando sobre as duas principais lacunas encontradas e como consequência disso a necessidade de propor um novo modelo de codificação.

4 ABORDAGENS ADOTADAS PARA SELEÇÃO DE REQUISITOS

Este capítulo tem como objetivo elucidar a maneira como a pesquisa foi realizada em todas as suas etapas. Desta forma, ele foi divido em partes para que em cada uma seja apresentada os métodos e técnicas utilizadas. A primeira parte trata de descrever a abordagem proposta, explicando quais lacunas a pesquisa se propõe a resolver. A segunda descreve a codificação proposta para o problema e a codificação que será utilizada para comparar os resultados. Em seguida, serão detalhados os *datasets* propostos. Na sequencia serão descritos os algoritmos desenvolvidos para aplicação das meta-heurísticas escolhidas e, por fim, a última parte descreve a ferramenta que foi desenvolvida para resolver o NRP.

4.1 ABORDAGEM PROPOSTA

As maiores deficiências encontradas na área de estudos de seleção de requisitos como um problema de otimização foram:

- Ausência de bases de dados de requisitos de projetos reais e que levem em conta os vários tipos existentes de interdependências. Por esse motivo, para a elaboração deste trabalho, decidiu-se levantar bases de dados que atendam a tais requisitos e que seus dados possam ser disponibilizados publicamente. Foram obtidos dados de três projetos reais, de duas empresas diferentes, um de pequeno porte e dois de médio porte para gerar as bases de dados utilizadas nesta pesquisa.
- Não foi encontrada uma codificação robusta, na literatura pesquisada, que não: gere resultados tendenciosos, precise de penalizações ou ainda, precise de reparações para tratar as soluções inválidas na população. Para tratar esta lacuna, será proposta uma nova codificação contínua, que sempre gera soluções válidas na população e por isso elimina a necessidade de penalizações ou reparações.

As bases de dados que estão sendo propostas neste trabalho são inéditas, ainda não possuem soluções ótimas conhecidas. Por esse motivo, ao invés de explorar apenas um método

de resolução, foram selecionados três métodos diferentes. Assim é possível comparar os resultados entre eles para poder melhor validar os resultados obtidos, são eles: Evolução Diferencial, Otimização por Enxame de Partículas e Algoritmos Genéticos. Para alcançar o objetivo do trabalho e determinar se os métodos podem obter resultados superiores aos obtidos manualmente, os resultados devem ser comparados com os requisitos que foram implementados no projeto real.

Como este trabalho propõe o uso de bases de dados reais ele deve considerar as interdependências entre os requisitos. Por isso a codificação proposta deve ser robusta o suficiente para comportar um grande número de interdependências e, para isso foi proposto o uso de uma codificação real ao invés de binária. Esta solução codifica a prioridade de cada requisito, ao invés da variável de decisão binária que determina se o requisito vai ou não ser implementado. Com isso foi possível eliminar completamente a necessidade de se utilizar operações de reparação ou penalizações, pois toda e qualquer solução gerada pelo algoritmo é sempre válida para o problema em questão.

O modelo matemático utilizado será o de Satisfação Parcial, pois é o modelo utilizado pelas empresas que desenvolveram os projetos que foram utilizados para gerar os *datasets*.

Os algoritmos foram escolhidos baseado nos mapeamentos sistemáticos do Apêndice A. Nestes mapeamentos, observou-se que GA e PSO são as técnicas mais utilizadas na literatura para resolver o problema de mochila, por isso ambas serão utilizadas no trabalho. Além destas técnicas, o algoritmo de DE já havia sido utilizado, no decorrer desta pesquisa, em experimentos com seleção de requisitos e apresentado resultados promissores, porém utilizando o Modelo de Satisfação Parcial. Por esse motivo, o algoritmo de DE também foi escolhido para ser utilizado no trabalho.

4.2 PROJETOS

Para sanar o problema de não existirem *datasets* baseados em projetos reais com interdependências, o autor entrou em contato com empresas que trabalha para conseguir dados de projetos reais e que pudessem ser disponibilizados publicamente. Ao todo foram fornecidos dados numéricos de 3 projetos diferentes. Esses dados incluem:

- quantidade de *stakeholders*;
- peso de cada *stakeholder* (variando de 1 a 5);
- quantidade de requisitos;

- custo de desenvolvimento de cada requisito (variando de 1 a 5);
- risco de mudança de cada requisito (variando de 1 a 5);
- grau de importância dado a cada requisito por cada *stakeholder* (variando de 1 a 5);
- valor total de satisfação obtido durante a implementação do projeto.

A condição para fornecimentos dos dados é que tanto as empresas quanto os projetos se mantivessem anônimos. Para tanto, dados que pudessem identificar os projetos como nome e descrição textual dos requisitos não foram disponibilizados. Cada projeto recebeu um código para identificá-los neste trabalho: CBS, FPG e SRT.

O projeto CBS é de um pequeno aplicativo móvel para rastreamento de ônibus de transporte coletivo privado, desenvolvido ao longo de 3 meses por uma pequena empresa de desenvolvimento de software. O orçamento disponível era de apenas 57% do necessário para implementar todos os requisitos solicitados pelos 5 *stakeholders* envolvidos.

O projeto FPG é de um sistema de auditoria de projetos de engenharia, de uma empresa do ramo de energia. Foi desenvolvido ao longo de 2 anos por um instituto de pesquisa de grande porte. Com o orçamento disponível foi possível implementar 77% das necessidades dos 3 *stakeholders* do projeto.

O projeto SRT é de um sistema de treinamentos profissionais para o setor de varejo, composto por um cliente móvel e três interfaces web. O desenvolvimento durou 12 meses e foi desenvolvido pela mesma empresa do projeto CBS. O orçamento disponível era de 58% do necessário para implementar todos os requisitos solicitados pelos 5 *stakeholders* envolvidos.

A Tabela 2 apresenta alguns dados relativos a cada um dos três projetos que serão utilizados no estudo de caso.

Tabela 2: *Datasets* utilizados no trabalho. Fonte: Autoria Própria

Instância	Requisitos	Stakeholders	Interdependências	Custo Total	Orçamento
CBS	32	5	23	419	239
FPG	143	3	178	1718	1319
SRT	246	5	209	2418	1390

A Tabela 3 apresenta alguns dados mais detalhados com relação ao número de cada tipo de interdependências contidas em cada projeto.

Tabela 3: Quantidade de cada interdependência para cada projeto. Fonte: Autoria Própria

Instância	AND	REQUIRES	TEMPORAL	CVALUE	ICOST	OR
CBS	1	10	0	0	2	4
FPG	4	151	5	0	14	0
SRT	9	96	19	0	44	16

A Tabela 4 apresenta o peso que cada um dos *stakeholder* tem em seus respectivos projetos.

Tabela 4: Valores de importância para cada *stakeholder* dos projetos. Fonte: Autoria Própria

Instância	S_1	S_2	S_3	S_4	S_5
CBS	4	1	3	5	5
FPG	2	5	3	-	-
SRT	4	4	3	2	5

4.3 CODIFICAÇÃO

Apesar do problema de seleção de requisitos ser um problema binário, quando as interdependências entre os requisitos são levadas em conta e consideradas como restrições no problema, o espaço de busca é alterado e o número de soluções inválidas para o problema aumenta muito. Isso acontece devido ao elevado número de restrições que passam a existir e como as populações iniciais das meta-heurísticas são geradas aleatoriamente, é impossível que a geração inicial contenha apenas soluções válidas.

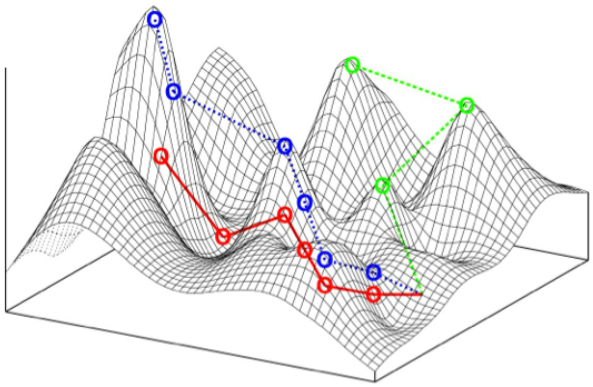
Quando o NRP é resolvido como um problema binário, duas coisas precisam ser feitas:

(a) Se a meta-heurística selecionada for para variáveis de espaço contínuo, ela deve ser adaptada para o uso de variáveis binárias.

(b) Se as interdependências devem ser levadas em conta, é necessário desenvolver um sistema para tratar as soluções inválidas que irão surgir na população.

Mudar uma meta-heurística que foi desenvolvida para variáveis de espaço contínuo para variáveis binárias pode levar a uma degradação dos resultados. Isso acontece porque em um problema de estaço contínuo a superfície do espaço de busca é contínua e suave, e isso facilita com que as soluções evoluam e subam no espaço de busca até atingirem a solução ótima (KRAUSE; LOPES, 2013). A Figura 5 mostra um exemplo da superfície do espaço de busca de um problema com variáveis no espaço contínuo.

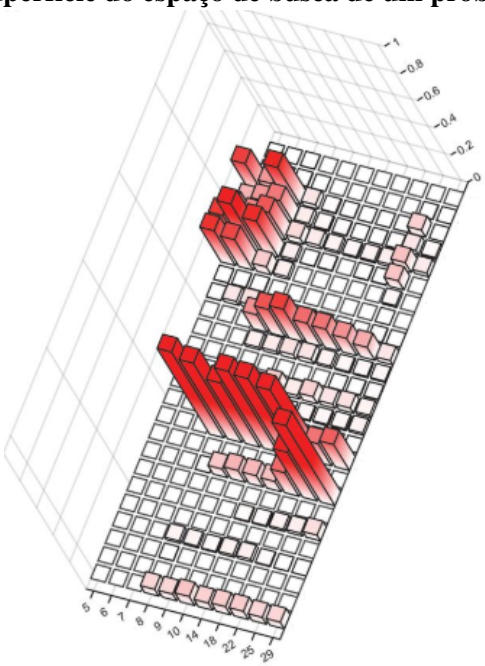
Figura 5: Superfície do espaço de busca de um problema contínuo



Fonte: Google Image Search

Quando a meta-heurística é adaptada para variáveis binárias a superfície do espaço de busca muda completamente. Ao invés de uma superfície suave e contínua ela fica com saltos, resultando em uma maior dificuldade das soluções caminharem pela superfície (KRAUSE; LOPES, 2013). A Figura 6 mostra um exemplo de uma superfície do espaço de busca de um problema com variáveis binárias.

Figura 6: Superfície do espaço de busca de um problema binário



Fonte: Google Image Search

A ideia para este trabalho foi adaptar o problema à meta-heurística, e não o contrário como foi feito em todos os trabalhos estudados na revisão sistemática. Para atingir esse objetivo uma nova codificação contínua está sendo proposta, na qual ao invés de codificar uma variável

binária (que identifica se o requisito irá ou não ser adicionado no produto) será codificada a prioridade dos requisitos dentro do projeto.

Ainda na codificação binária, é necessário o desenvolvimento de um operador de penalização ou de reparação das soluções inválidas. A penalização consistem em reduzir o valor do *fitness* de soluções inválidas de modo a promover as soluções válidas para as próximas gerações. Já a utilização de um operador de reparação tem um inconveniente quando existem muitas interdependências. Em tais situações é comum elas conflitarem entre si e o operador de reparação deve priorizar uma ou outra. Na prática, todas possuem o mesmo peso, e por isso as soluções geradas dessa forma são tendenciosas.

O uso da codificação contínua proposta neste trabalho elimina completamente as duas desvantagens acima citadas e pode ser escalada para qualquer número e tipo de interdependências diferentes presentes nos projetos.

De uma forma bem sucinta, no vetor solução foram codificadas as prioridades dos requisitos disponíveis no projeto, e não a variável binária, que indica se um requisito será implementado ou não. Com essa abordagem, todas as restrições são tratadas no processo de codificação/decodificação e qualquer solução gerada é sempre válida e imparcial.

A penalização as soluções inválidas é uma prática comum e muito utilizada desde que foi inicialmente proposta (STORN; PRICE, 1997). Então para poder validar a codificação proposta todos os *datasets* serão resolvidos utilizando as meta-heurísticas com codificação binária e penalização. Os resultados das duas versões serão então comparados entre si e com as soluções encontradas durante o desenvolvimento dos projetos para avaliar qual se sai melhor.

4.3.1 CODIFICAÇÃO BINÁRIA COM PENALIZAÇÃO PROGRESSIVA

Esta codificação consiste em armazenar no vetor quais requisitos serão utilizados na solução, porém, a grande quantidade de interdependências transforma o problema em algo extremamente restrito. Com isso, apenas algumas soluções geradas serão válidas e haverá uma grande dificuldade de alguns algoritmos propagarem estas soluções para as gerações seguintes, principalmente o DE, que não possui nenhum operador parecido com o elitismo, presente no GA.

Existe uma grande chance de a população inicial não possuir nenhuma solução válida nos projetos maiores (SRT e FPG). Isso testará os algoritmos ao extremo, pois será bem difícil gerar soluções válidas apenas a partir de soluções inválidas.

Para eliminar as soluções inválidas e priorizar as válidas, será implementado um sis-

tema de penalização, que será explicado em maiores detalhes a seguir.

Durante a decodificação cada requisito será analisado para todas as interdependências que afetam ele. Para cada restrição que seja infringida, será subtraído um determinado valor do *fitness* final. Ou seja, quanto mais restrições forem infringidas, mais o *fitness* vai diminuir.

Para tentar melhorar a diversidade das soluções nas gerações iniciais, o valor diminuído do *fitness* vai ser menor no começo da execução e vai sendo aumentado conforme as gerações forem passando, ou seja, será um sistema de penalização progressivo.

A parte superior da Figura 7 mostra como ficou estruturado o vetor solução para esta codificação. De P_1 a P_n são armazenadas as variáveis de decisão x_i que representam se o requisito r_i será ou não desenvolvido. Ao final do vetor foram adicionadas 4 informações extras que são carregadas com o vetor solução e tem como objetivo simplificar o código das metaheurísticas e a exibição dos resultados no software que foi desenvolvido para resolver os *datasets* propostos. Estas informações extras são:

- $P_{\mathbf{n+1}}$**:** número total de restrições que não foram atendidas;
- $P_{\mathbf{n+2}}$**:** custo total para implementar a solução;
- $P_{\mathbf{n+3}}$**:** valor total de satisfação obtido pela solução;
- $P_{\mathbf{n+4}}$**:** valor do *fitness* da solução;

Figura 7: Vetor solução da codificação binária com penalização

P_1	P_2	...	P_n	P_{n+1}	P_{n+2}	P_{n+3}	P_{n+4}
x_1	x_2	...	x_n	Infringements	total cost	total value	fitness
0	1	...	0	2	256	1266	0,65789

Fonte: Autoria Própria

Na parte inferior da Figura 7 é possível observar o exemplo de um vetor solução com os valores preenchidos. Neste exemplo os requisitos r_1 e r_n não seriam implementados pois o valor do elemento que os representa no vetor é 0, ao contrário do requisito r_2 que possui o valor 1 no vetor solução. Entretanto essa seria uma solução inválida pois não respeitou duas restrições do problema, como pode ser visto na posição do elemento $infringements$. O custo total da solução é 256, o valor total de satisfação 1266 e o *fitness* de 0.65789.

4.3.1.1 FUNÇÃO DE *FITNESS*

A função de *fitness* calcula o valor total de satisfação dos *stakeholders* com a versão final do software. Isso é feito por meio da soma dos valores de satisfação de cada requisito selecionado, ou $score_i$, conforme a Equação (3).

Para cada requisito incluído na solução, é necessário verificar suas interdependências e, se ela for do tipo ICOST, ela altera o custo de desenvolvimento de outro requisito. Portanto é necessário fazer essa compensação no valor total da soma.

Além disso, deve ser levado em conta a geração atual e o número máximo de gerações para determinar o valor que será subtraído do *fitness* para compensar as restrições que não foram atendidas.

O fator penalização escolhido foi de 0.05 para cada interdependência não respeitada, porém esse valor é progressivo, ele começa com zero na primeira iteração e vai aumentando até 0.05 na última iteração. Tal estratégia permite que no início do algoritmo as soluções sejam mais livres, indiferente de serem válidas ou não, permitindo uma maior diversidade de soluções. Conforme as gerações vão evoluindo as soluções inválidas devem ir sendo descartadas aos poucos para que nas gerações finais fiquem apenas as válidas, por isso elas são mais penalizadas perto do fim da execução.

O Algoritmo 1 apresenta o pseudocódigo utilizado para o cálculo da função de *fitness* desta codificação.

O algoritmo recebe 3 entradas:

- **solution:** vetor contendo os requisitos selecionados para o projeto;
- **generation:** número da geração atual;
- **maxGenerations:** número máximo de gerações.

As variáveis $totalCost$ e $totalValue$ representam as somas de todos os custos e valores de satisfação dos requisitos selecionados, e são inicializadas com zero. Em seguida é feito um *loop* através de todos os requisitos no qual o custo e valor de cada um é adicionado às suas respectivas variáveis. O custo individual de cada requisito e seu valor de satisfação vêm do *dataset* e foram chamados aqui de $requirementCost$ e $requirementValue$.

Na linha 7 outro *loop* através dos requisitos é iniciado, agora para analisar as interdependências associadas a cada requisito. Apenas dois tipos são relevantes para o cálculo do

fitness: ICOST e IVALUE. Essas interdependências alteram o custo ou valor do requisito associado a elas, então caso elas existam nas interdependências do requisito, diminuímos o valor que foi adicionado às variáveis de soma (linhas 4 e 5), e somamos o novo valor, chamado de $alternativeRequirementCost$ ou $alternativeRequirementValue$.

Por fim, um valor intermediário do *fitness* é calculado (linha 23), em que o valor total de satisfação da solução é dividido pelo máximo teórico (valor que seria obtido caso todos os requisitos fossem implementados). Em seguida é aplicada a penalização progressiva (linha 24). Caso o *fitness* seja menor que zero, seu valor é definido para zero. A variável $numberOfInfringements$ vem armazenada dentro do vetor solução e foi calculada pela função de decodificação, que sera apresentada no Algoritmo 2.

Algorithm 1: calculateBinaryFitness

```
Input: solution, generation, maxGenerations
totalCost = 0;
totalValue = 0;
foreach requirement in solution do
    totalCost = totalCost + requirementCost;
    totalValue = totalValue + requirementValue;
end
foreach requirement in solution do
    foreach dependency in requirementDependencies do
        if typeOfDependency == ICOST then
            if solution contains affected requirement then
                totalCost = totalCost - requirementCost;
                totalCost = totalCost + alternativeRequirementCost;
            end
        end
        if typeOfDependency == IVALUE then
            if solution contains affected requirement then
                totalValue = totalValue - requirementValue;
                totalValue = totalValue + alternativeRequirementValue;
            end
        end
    end
end
fitness = totalValue / maxTheoreticalValue;
fitness = fitness - (numberOfInfringements * 0.05 * generation / maxGenerations);
if fitness < 0 then
    fitness = 0;
end
return fitness
```

4.3.1.2 DECODIFICAÇÃO

O processo de decodificação recebe um vetor com os requisitos que foram escolhidos para a solução e retorna o próprio vetor solução adicionando o número total de restrições que

não foram respeitadas. O Algoritmo 2 apresenta o processo de decodificação da solução.

Algorithm 2: decodeBinarySolution

```
Input: solution
effort = 0;
infringements = 0;
foreach requirement in solution do
    if requirement is selected then
        effort = effort + requirementEffort;
        if requirement has 'REQUIRES' or 'TEMPORAL' then
            if required requirement is no in solution then
                infringements = infringements + 1;
            end
        end
        if requirement has 'AND' then
            if associated required is no in solution then
                infringements = infringements + 1;
            end
        end
        if requirement has 'OR' then
            if associated required is in solution then
                infringements = infringements + 1;
            end
        end
    end
    set infringements total in solution vector;
    return solution
end
```

As variáveis $effort$ e $infringements$ são inicializadas com zero e representam o custo total de desenvolvimento e o total de restrições que não foram respeitadas pela solução respectivamente.

O processo em si se inicia com um *loop* pelo vetor solução verificando se o requisito foi selecionado ou não para o projeto. Em caso positivo o esforço de desenvolvimento do requisito é somado à variável $effort$ e em seguida são realizadas as validações das restrições.

A primeira validação é referente às interdependências REQUIRES e TEMPORAL.

Caso elas estejam presentes no requisito em questão é verificado se o requisito associado a elas não está presente na solução e, caso não esteja $infringements$ é incrementada em 1. A próxima validação é referente à interdependência AND e caso ela exista e o requisito associado não esteja presente no projeto, $infringements$ é incrementada em 1. Para a interdependência OR é verificado se o requisito associado foi incluído na solução e em caso positivo $infringements$ é incrementada em 1.

Finalmente, após todas as validações o número total de restrições não atendidas é salvo no vetor solução e este é retornado pela função.

4.3.2 CODIFICAÇÃO CONTÍNUA

A ideia ao propor esta codificação foi de criar um sistema de codificação mais robusta. Esta codificação sempre gera soluções válidas na população porque ao invés de armazenar quais requisitos serão utilizados, ela armazena a prioridade de cada requisito. Com isso, o processo de decodificação consiste em ir adicionando os requisitos em um vetor de requisitos selecionados e durante este processo ir tratando qualquer interdependência que possa influenciar os requisitos sendo adicionados.

A parte superior da Figura 8 mostra como ficou estruturado o vetor solução para esta codificação. De P_1 a P_n são armazenadas as prioridades de cada requisito dentro do projeto. Ao final do vetor foram adicionadas 3 informações extras que são carregadas com o vetor solução e tem como objetivo simplificar o código das meta-heurísticas e a exibição dos resultados no software que foi desenvolvido para resolver os *datasets* propostos. Estas informações extras são:

- $P_{\mathbf{n+1}}$**:** custo total para implementar a solução;
- $P_{\mathbf{n+2}}$**:** valor total de satisfação obtido pela solução;
- $P_{\mathbf{n+3}}$**:** valor do *fitness* da solução;

Na parte inferior da Figura 8 é possível observar o exemplo de um vetor solução com os valores preenchidos. Neste exemplo a ordem de prioridade dos requisitos seria r_2, r_n e r_1. O custo total da solução é 273, o valor total de satisfação 1399 e o *fitness* de 0.75691.

Figura 8: Vetor solução da codificação contínua

P_1	P_2	...	P_n	P_{n+1}	P_{n+2}	P_{n+3}
$Priority_1$	$Priority_2$	...	$Priority_n$	total cost	total value	fitness
0,364	0,987	...	0,644	273	1399	0,75691

Fonte: Autoria Própria

4.3.2.1 FUNÇÃO DE *FITNESS*

A função para cálculo do *fitness* para esta codificação é muito parecida com a do Algoritmo 1, porém sem a penalização de *fitness* já que para este método nunca haverá uma solução inválida. O Algoritmo 3 apresenta o pseudocódigo desta função.

Algorithm 3: calculateContinuousFitness

```
   Input: solution
1  totalCost = 0;
2  totalValue = 0;
3  foreach requirement in solution do
4      totalCost = totalCost + requirementCost;
5      totalValue = totalValue + requirementValue;
6  end
7  foreach requirement in solution do
8      foreach dependency in requirementDependencies do
9          if typeOfDependency == ICOST then
10             if solution contains affected requirement then
11                 totalCost = totalCost - requirementCost;
12                 totalCost = totalCost + alternativeRequirementCost;
13             end
14         end
15         if typeOfDependency == IVALUE then
16             if solution contains affected requirement then
17                 totalValue = totalValue - requirementValue;
18                 totalValue = totalValue + alternativeRequirementValue;
19             end
20         end
21     end
22 end
23 fitness = totalValue / maxTheoreticalValue;
24 return fitness
```

As variáveis $totalCost$ e $totalValue$ representam as somas de todos os custos e valores de satisfação dos requisitos selecionados, e são inicializadas com zero. Em seguida é feito um *loop* através de todos os requisitos no qual o custo e valor de cada um é adicionado às suas respectivas variáveis. O custo individual de cada requisito e seu valor de satisfação vêm do *dataset* e foram chamados aqui de *requirementCost* e *requirementValue*.

Na linha 7 outro *loop* através dos requisitos é iniciado, agora para analisar as interdependências associadas a cada requisito. Apenas dois tipos são relevantes para o cálculo do *fitness*: ICOST e IVALUE. Essas interdependências alteram o custo ou valor do requisito as-

sociado a elas, então caso elas existam nas interdependências do requisito, diminuímos o valor que foi adicionado às variáveis de soma (linhas 4 e 5), e somamos o novo valor, chamado de $alternativeRequirementCost$ ou $alternativeRequirementValue$.

Por fim, o valor do *fitness* é calculado (linha 23), em que o valor total de satisfação da solução é dividido pelo máximo teórico (valor que seria obtido caso todos os requisitos fossem implementados).

4.3.2.2 DECODIFICAÇÃO

Nesta codificação o que está sendo armazenado é a prioridade de cada requisito e eles são analisados em ordem decrescente de prioridade, pois quanto maior o valor maior a prioridade do requisito. O valor de prioridade foi definido no intervalo [0,1]. O Algoritmo 4 mostra o pseudo-código do processo de decodificação.

Inicialmente algumas variáveis são inicializadas:

- **decodedSolution:** inicializada com um vetor vazio, ela armazena os requisitos que serão utilizados pela solução;
- **orderedSolutionVector:** contém uma lista de requisitos ordenados por prioridade e recebe a saída de uma função que ordena os requisitos ($orderSolutionVector$);
- **remaining:** definida inicialmente como o orçamento total do projeto, pois nesse ponto ainda não existem requisitos adicionados na solução;
- **effort:** variável para armazenar o custo total de adicionar um requisito e todas as suas dependências.

Algorithm 4: decodeContinuousSolution

```
   Input: solution
1  decodedSolution = new empty vector;
2  orderedSolutionVector = orderSolutionVector(solution);
3  remainingBudget = projectBudget;
4  effort = 0;
5  foreach currentRequirement in orderedSolutionVector do
6      if requirementEffort > remainingBudget then
7          continue;
8      end
9      if decodedSolution contains currentRequirement then
10         continue;
11     end
12     proposal = new empty vector;
13     call addDependencies(requirement, out areDependenciesValid, out
         tempRequirements);
14     if areDependenciesValid then
15         add each requirement in tempRequirements to proposal;
16         add currentRequirement to proposal;
17     else
18         continue;
19     end
20     effort = calculateEffort(proposal);
21     if effort > remainingBudget then
22         continue;
23     end
24     add proposal to decodedSolution;
25     remainingBudget = projectBudget - effort;
26 end
27 return decodedSolution;
```

Na linha 5 o *loop* principal tem inicio e analisa cada um dos requisitos ordenados para ver se é possível adicioná-los à solução ou não. As verificações necessárias são as seguintes:

- Na linha 6, caso o escorço de implementação do requisito ($requirementEffort$) seja maior que o orçamento ainda disponível em *remainingBudget* não é possível adicioná-lo

à solução e a análise deste requisito é interrompida e a do próximo começa.

- Na linha 9, caso o requisito em questão já esteja na solução atual (*decodedSolution*) a análise deste requisito é interrompida e a do próximo começa. Isso pode acontecer caso ele tenha sido adicionado como dependência de outro requisito analisado anteriormente.

- Na linha 12, uma nova variável chamada *proposal* é inicializada como um vetor vazio. O objetivo desta variável é armazenar uma "proposta"de requisitos, que pode ou não ser adicionada à solução. Esta proposta é formada por um determinado requisito mais todos os requisitos que devem ser implementados com ele devido a alguma interdependência.

- Na linha 13, a função *addDependencies* é chamada e ela gera uma lista de todos os requisitos dos quais o requisito sendo analisado depende. A saída dela é realizada através de variáveis passadas por referência, e são: a lista de requisitos relacionados ao requisito analisado (*tempRequirements*) e uma variável booleana que indica se esse conjunto de requisitos dependentes respeita todas as restrições (*areDependenciesValid*).

- Na linha 14, caso adicionar os requisitos dependentes não deixe de respeitar nenhuma restrição, cada um destes requisitos é então adicionado à "proposta"juntamente do requisito sendo analisado (linhas 15 e 16). Ainda neste *if*, caso os requisitos dependentes violem alguma restrição o processo é interrompido e a análise do próximo requisito tem início.

- Na linha 20, a função *calculateEffort* é chamada e tem por objetivo somar os esforço de desenvolvimento de todos os requisitos contidos na "proposta". Está função já considera a interdependência do tipo ICOST e faz os devidos ajustes no custo dos requisitos passados para ela.

- Na linha 21, caso o esforço para desenvolver todos os requisitos contidos na "proposta"seja maior que o orçamento ainda disponível em *remainingBudget*, ela é completamente descartada e não é possível adicionar o requisito sendo analisado na solução.

- Na linha 24, caso a proposta não viole nenhuma restrição nem exceda o orçamento, cada requisito contido nela é adicionado à solução.

- Na linha 25, o custo total de desenvolvimento da "proposta"é subtraído do orçamento ainda disponível na solução (*remainingBudget*).

Depois que todos os requisitos são analisados a função retorna um vetor contendo apenas os requisitos que foram selecionados pela solução, que estão armazenados na variável *decodedSolution*.

A função *addDependencies* utilizada no Algoritmo 4 está detalhada no Algoritmo 5. Esta função recebe um requisito e navega por toda a árvore de dependências, chamando ela mesma recursivamente até analisar todos os requisitos necessários para adicionar o requisito passado inicialmente para a função. Ao final, ela retorna um valor booleano indicando se encontrou algum conflito nas dependências que precisam ser atendidas e um conjunto com todos os requisitos que precisariam ser adicionados junto do requisito inicialmente analisado.

Algorithm 5: addDependencies

```
   Input: requirement
   Output: areDependenciesValid, proposal
1  areDependenciesValid = true;
2  foreach dependency in requirementDependencies do
3      if type of dependency == (REQUIRES or TEMPORAL or AND) then
4          add afftectedDependencyRequirement to proposal;
5          call addDependencies(afftectedDependencyRequirement, out
             areDependenciesValid, out proposal);
6          if !areDependenciesValid then
7              return;
8          end
9      end
10 end
11 foreach requirement in proposal do
12     if type of dependency == OR then
13         if affected requirement is in solution then
14             areDependenciesValid = false ;
15         end
16     end
17 end
18 return
```

Na linha 2, o processo para determinar quais requisitos estão relacionados ao requisito passado para a função *addDependencies* tem início e ela precisa analisar apenas as interdependências do tipo REQUIRES, TEMPORAL e AND. Porém, para validar se a proposta é válida é preciso avaliar se nenhuma interdependência do tipo OU foi violada, o que é feito no *loop* da linha 11.

4.4 META-HEURÍSTICAS

Esta seção explica como foi realizada a implementação e apresenta os algoritmos desenvolvidos para cada meta-heurística.

4.4.1 ALGORITMOS GENÉTICOS

O GA é uma das meta-heurística mais difundidos para uso em otimização devido à sua simplicidade de uso. Apesar do algoritmo em si não ser complexo ele é extenso e trabalhoso de implementar. Por esse motivo, optamos por utilizar uma biblioteca de GA já existente e de código aberto chamada GAF [1].

A biblioteca precisa que o vetor solução seja passado para ela como um vetor binário, e para conseguir utilizá-la com a codificação contínua a prioridade de cada requisito foi codificada em genes de 10 *bits*, que pode representar 1024 possíveis valores diferentes. Este valor é suficiente para os *datasets* propostos, pois o maior deles possui 246 requisitos.

Como o algoritmo de GA não foi desenvolvido pelo autor, não será apresentado nenhum algoritmo nesta seção, mas o código completo utilizando a biblioteca GAF pode ser visto no código-fonte do software desenvolvido para este trabalho.

4.4.2 EVOLUÇÃO DIFERENCIAL

Os detalhes do algoritmo de DE que foi utilizado neste trabalho são apresentados no Algoritmo 6 e os detalhes do código são apresentados a seguir.

O algoritmo se inicia com a geração da população inicial, na linha 1, com um *loop* do tamanho da variável $PopulationSize$. A cada iteração um indivíduo é inicializado com valores aleatórios, tem seu valor de *fitness* salvo em seu vetor solução (ver Figuras 7 e 8) e é adicionado à população inicial (variável $Population$).

Na linha 6 é iniciado o *loop* principal do algoritmo e ele é executado até que a condição de parada seja atingida. Dentro desse *loop* é iniciado um outro *loop* que percorre cada indivíduo da população (linha 7) e realiza as seguintes operações:

[1] GAF Source Code. https://bitbucket.org/johnnewcombe/gaf/wiki/Home. Último acesso: 12/03/2018.

Algorithm 6: Pseudo-código do algoritmo de DE

```
1  for i = 1 to PopulationSize do
2      individual = generateRandomIndividual() ;
3      EvaluateFitness(individual);
4      Population[i] = individual;
5  end
6  while StopConditionSatisfied = false do
7      for for i = 1 to PopulationSize do
8          r1, r2, r3 = selectDifferentRandomIndices;
9          jrand = random(0...NumberOfRequirements);
10         for j = 0 to NumberOfRequirements do
11             if ValueHasToChangeDueCrossoverRate or j = jrand then
12                 TrialIndividual[j] = Population[r3][j] + F * (Population[r1][j] –
                    Population[r2][j]);
13             else
14                 TrialIndividual[j] = Population[i][j];
15             end
16         end
17         CorrectBoundaryConstrainViolations(TrialIndividual);
18         EvaluateFitness(TrialIndividual);
19         if TrialIndividual is better than Population[i] then
20             NextPopulation[i] = TrialIndividual;
21         else
22             NextPopulation[i] = Population[i];
23         end
24     end
25     Population = NextPopulation;
26 end
```

- Na linha 8, são selecionados 3 índices aleatórios e diferentes entre si: $r1$, $r2$ e $r3$;
- Na linha 9, um número aleatório variando entre zero e o número total de requisitos é armazenado em *jrand*;
- Na linha 10, um *loop* por cada elemento do vetor solução é iniciado;
- Na linha 11, um condicional verifica se o valor do elemento precisa mudar devido à taxa

de *crossover* ou porque o índice dele é igual ao índice contido em $jrand$. Em caso positivo o elemento equivalente no vetor $TrialIndividual$ é preenchido de acordo com a Equação 6 e, em caso negativo, o valor do elemento sendo analisado é apenas copiado para seu equivalente em $TrialIndividual$;

- Ao final do *loop* da linha 11 o vetor $TrialIndividual$ está completamente preenchido e a linha 17 faz um ajuste de valores. Se o valor de algum elemento ficou fora do intervalo [0,1] ele é recolocado dentro dos limites;
- Na linha 18, o *fitness* do vetor $TrialIndividual$ é calculado;
- Na linha 19, um condicional analisa se o *fitness* do vetor $TrialIndividual$ é melhor que o *fitness* do indivíduo sendo analisado pelo *loop* da linha 7. Em caso positivo, o $TrialIndividual$ é propagado para a próxima geração e, em caso negativo o indivíduo sendo analisado permanece é quem é propagado para a próxima geração.

Ao final, a população atual é substituída pela nova população gerada dentro do *loop* da linha 7 e o *loop* da linha 6 é reiniciado para a próxima iteração.

4.4.3 OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS

Os detalhes do algoritmo de PSO que foi utilizado neste trabalho são apresentados no Algoritmo 7 e os detalhes do funcionamento do código são apresentados a seguir.

O algoritmo tem início com a inicialização das partículas, no *loop* da linha 1, no qual para cada partícula:

- Na linha 3, a posição x da partícula é inicializada dentro do intervalo [0,1];
- Na linha 4, a velocidade v da partícula é inicializada no intervalo [-vMax, +vMax], sendo vMax a máxima velocidade da partícula;
- Na linha 6, a variável $bestParticle$ é inicializada com a partícula sendo criada;
- Na linha 7, um condicional verifica se o *fitness* da melhor partícula local é inferior ao *fitness* da melhor partícula global. Em caso positivo, ela melhor partícula global é atualizada para a melhor partícula local.

Na linha 11 o *loop* principal do algoritmo tem início e é repetido até que a condição terminal seja atingida. A cada iteração acontecem dois *loops* dentro dele:

- O primeiro *loop*, na linha 12, atualiza as melhores partículas local e global verificando quais possuem o melhor *fitness* (linhas 13 e 16 respectivamente);
- O segundo *loop*, na linha 20, é o responsável por "mover"as partículas pelo espaço de busca, atualizando a posição e a velocidade de cada partícula de acordo com as Equações (11) e (9).

Algorithm 7: Pseudo-código do algoritmo de PSO

```
1  for i = 1 to ParticlesCount do
2      for d = 1 to RequirementCount do
3          x[i, d] = randon(0, 1);
4          v[i, d] = random(-vMax, +vMax);
5      end
6      bestParticle = x[i];
7      if fitness(bestParticle) < fitness(bestGlobal) then
8          bestGlobal = bestParticle ;
9      end
10 end
11 while StopConditionSatisfied = false do
12     for i = 1 to ParticlesCount do
13         if fitness(x[i]) < fitness(bestParticle) then
14             bestParticle = x[i];
15         end
16         if fitness(bestParticle) < fitness(bestGlobal) then
17             bestGlobal = bestParticle
18         end
19     end
20     for i = 1 to ParticlesCount do
21         for d = 1 to RequirementCount do
22             v[i,d] = v[i,d] + c1 * random(0,1) * (partibleBest[d] – x[i,d]) + c2 *
                random(0,1) * (globalBest[d] - x[i,d]);
23             x[i,d] = x[i,d] + v[i,d];
24         end
25     end
26 end
```

4.5 MÉTODO EXATO

A maior desvantagem de se utilizar e propor novos conjuntos de dados para um determinado problema é que a solução nem sempre é conhecida. Este é o caso dos conjuntos de dados propostos neste trabalho. Para avaliar a qualidade das soluções geradas pelos algoritmos, o ideal seria conhecer a solução exata para cada um dos projetos. Como os dados estão sendo propostos pela primeira vez neste trabalho, a solução exata não é conhecida em nenhum dos projetos.

Para tentar encontrar a solução exata foi desenvolvido o algoritmo de busca exaustiva apresentado no Algoritmo 8, e consiste em:

- Na linha 1, a variável com a solução exata, $exactSolution$, é inicializada com zero;
- Na linha 2, a variável $solutions$ é um vetor com todas as combinações possíveis de requisitos, que são gerados pela função $generateAllPossibleCombinations$;
- Na linha 3, o *loop* principal é iniciado, iterando por cada uma das possíveis soluções;
- Na linha 4, o custo total de desenvolvimento, $effort$, é calculado para a solução sendo analisada;
- Na linha 5, um condicional verifica se o custo é maior que o orçamento do projeto e em caso positivo interrompe a análise e parte para a próxima solução;
- Na linha 8, um condicional verifica se alguma das interdependências não foi respeitada e em caso positivo interrompe a análise e parte para a próxima solução;
- Na linha 11, o valor total de satisfação da solução é calculado, $solutionScore$, pois se chegou neste ponto, é porque é uma solução válida;
- Na linha 12, caso o valor total de satisfação da solução seja maior que o melhor valor encontrado até o momento, ele passa a ser o novo melhor valor encontrado até o momento ($exactSolution$).

Ao final do algoritmo, a variável $exactSolution$ possui o valor da maior satisfação possível para o problema em questão, ou seja, a solução exata do mesmo.

Algorithm 8: Pseudo-código do algoritmo de busca exaustiva

```
1  exactSolution = 0;
2  solutions = generateAllPossibleCombinations();
3  foreach solution in solutions do
4      effort = calculateEffort(solution);
5      if effort > projectBudget then
6          continue;
7      end
8      if solution did not respect any interdependendy then
9          continue;
10     end
11     solutionScore = calculateSolutionScore(solution);
12     if solutionScore > exactSolution then
13         exactSolution = solutionScore;
14     end
15 end
```

4.6 ESTUDO DE CASO COM ALUNOS

Neste estudo de caso, alunos de graduação e de mestrado da UTFPR participaram como líderes do projeto CBS e deveriam determinar quais requisitos seriam implementados, sem exceder o orçamento e tentando maximizar o nível de satisfação dos usuários. O contexto e as dificuldades do projeto foram explicados aos alunos e foram disponibilizadas 2 horas para que eles chegassem a uma solução.

Um total de 35 alunos participaram deste experimento, selecionando quais requisitos eles iriam desenvolver caso fossem responsáveis pelo projeto.

Apesar de nem todos alunos terem experiência com seleção de requisitos, o objetivo desse estudo de caso foi avaliar como os alunos se comportariam e a qualidade das respostas geradas.

4.7 FERRAMENTA NRP SOLVER

Ao longo da evolução deste trabalho observou-se a necessidade de uma interface para executar os testes com os algoritmos com maior agilidade e também visualizar os resultados

de forma mais simples. Assim sendo, o protótipo de software apresentado nesta seção foi implementado. A interface consiste de uma tela principal do protótipo que pode ser vista na Figura 9.

Figura 9: Tela principal do NRP Solver

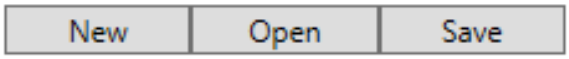
Fonte: Autoria Própria

A tela principal é dividida em 5 áreas classificadas de acordo com as funcionalidades que agregam. A primeira área é bem simples e diz respeito às funções de salvamento dos resultados e posterior carregamento para visualização dos resultados. Como pode ser visto na Figura 10, ela consiste apenas de 3 botões com as seguintes funcionalidades:

- **New:** limpa toda a interface e cria um novo arquivo de resultados em branco;
- **Open:** carrega os dados de um arquivo de resultados que foi salvo anteriormente. Esses resultados incluem as configurações dos algoritmos, *dataset* selecionado e gráficos de *fitness* de cada execução independente;

- **Save:** salva os resultados de todas execuções independentes que estão na memória para um arquivo em disco.

Figura 10: Área de carregamento/salvamento de resultados



Fonte: Autoria Própria

A segunda área corresponde às configurações do *dataset* que será utilizado para gerar os resultados, como pode ser visto na Figura 11. A interface carrega automaticamente todos os *datasets* presentes na pasta DATA do sistema e os lista numa caixa de listagem para seleção. Após selecionado o *dataset* o usuário pode escolher como quer setar o limite de orçamento do projeto:

- **fixed:** com esta opção o usuário especifica o orçamento do projeto através de um valor absoluto;
- **proportional:** utilizando esta opção o usuário pode especificar uma proporção do valor total necessário para desenvolver o projeto. Por exemplo, se o projeto necessitar de 1000 unidades para implementar todos os requisitos e o usuário digitar 0.5 para o orçamento disponível, o algoritmo entenderá que o orçamento total disponível é de 500 unidades.

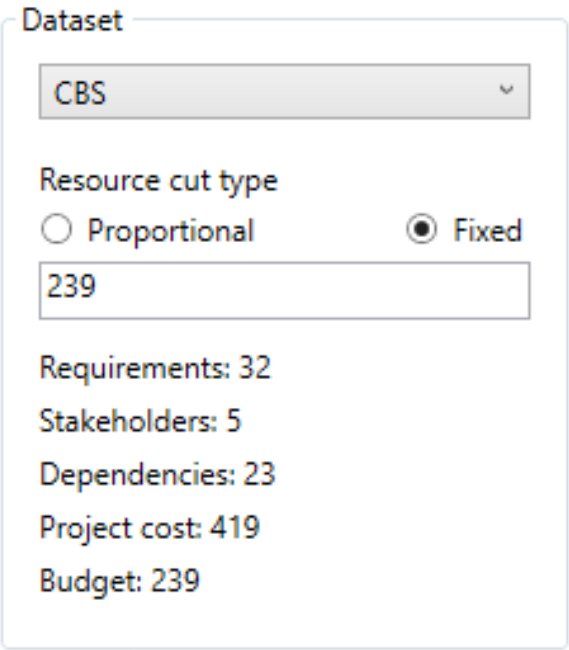
Quando um *dataset* é selecionado esta área também exibe alguns dados do mesmo:

- **Requirements:** total de requisitos presentes no *dataset*;
- **Stakeholders:** total de *stakeholders* presentes no *dataset*;
- **Dependencies:** total de interdependências associadas aos requisitos;
- **Project cost:** custo total necessário para implementar todos os requisitos;
- **Budget:** orçamento disponível para o algoritmo utilizar ao determinar quais requisitos devem ser selecionados;

A terceira e mais importante área tem por objetivo a configuração das meta-heurísticas e é dividida em 3 outras áreas, uma para cada meta-heurística. As Figuras 12 a 14 mostram os parâmetros que podem ser configurados para cada um dos algoritmos.

Para o algoritmo DE, as configurações podem ser vistas na Figura 12 e são as seguintes:

Figura 11: Área de configuração do *dataset*



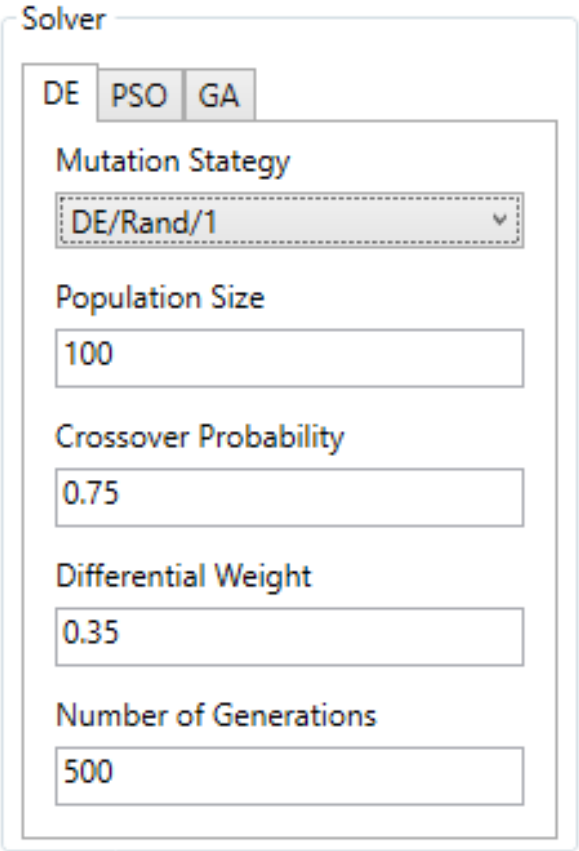
Fonte: Autoria Própria

- **Mutation Stategy:** estratégia de mutação a ser usada pelo algoritmo. São elas: DE/Rand/1, DE/Best-1, DE-Target-to-best/1, DE/Rand/2 e DE/Best/2;
- **Population Size:** quantidade de indivíduos na população;
- **Crossover Probability:** probabilidade de *crossover*, o C_r apresentado na Equação (7);
- **Differential Weight:** peso diferencial, o F apresentado na Equação (6);
- **Number of Generations:** número máximo de gerações que o algoritmo irá evoluir até parar.

Para o algoritmo PSO, as configurações podem ser vistas na Figura 13 e são as seguintes:

- **Number of Particles:** número total de partículas na população;
- **Innertial Weight:** peso inercial, o w apresentado na Equação (11);
- **Cognitive/local Weight:** fator local, o c_1 apresentado na Equação (11);
- **Social/global Weight:** fator global, o c_2 apresentado na Equação (11);

Figura 12: Área de configuração do algoritmo DE



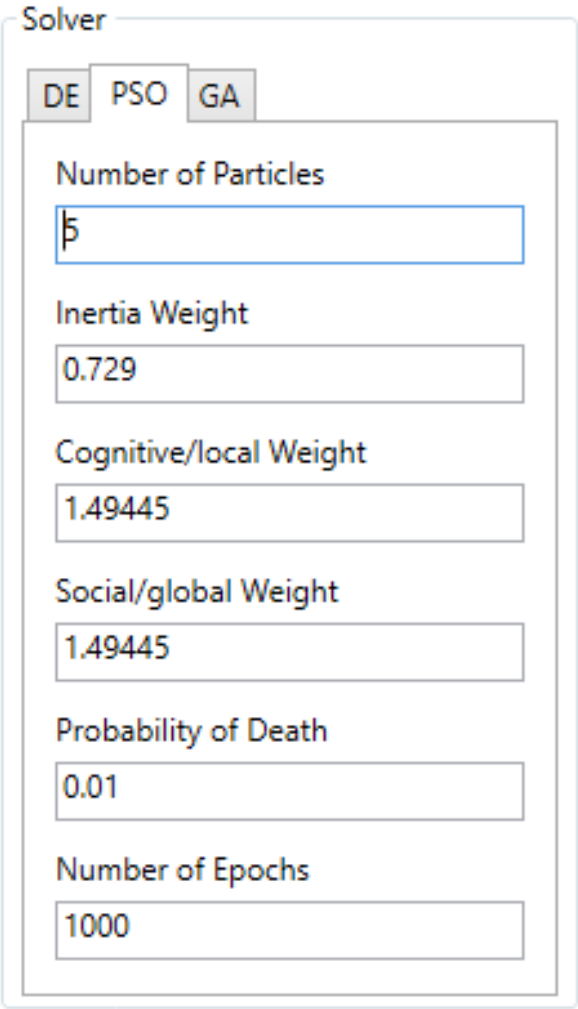
Fonte: Autoria Própria

- **Probability of Death:** chance de uma partícula ser completamente reiniciada com um novo valor, atua como uma espécie de mutação no algoritmo.
- **Number of Epochs:** número máximo de gerações que o algoritmo irá evoluir até parar.

Para o algoritmo GA, as configurações podem ser vistas na Figura 14 e são as seguintes:

- **Gene Size:** tamanho, em bits, de cada gene do cromossomo;
- **Population Size:** quantidade de indivíduos na população;
- **Crossover Probability:** taxa de cruzamento entre os indivíduos;
- **Crossover Type:** como o cromossomo filho será gerado a partir dos dados dos cromossomos pais;
- **Mutation Probability:** taxa de mutação dentro da população;

Figura 13: Área de configuração do algoritmo PSO

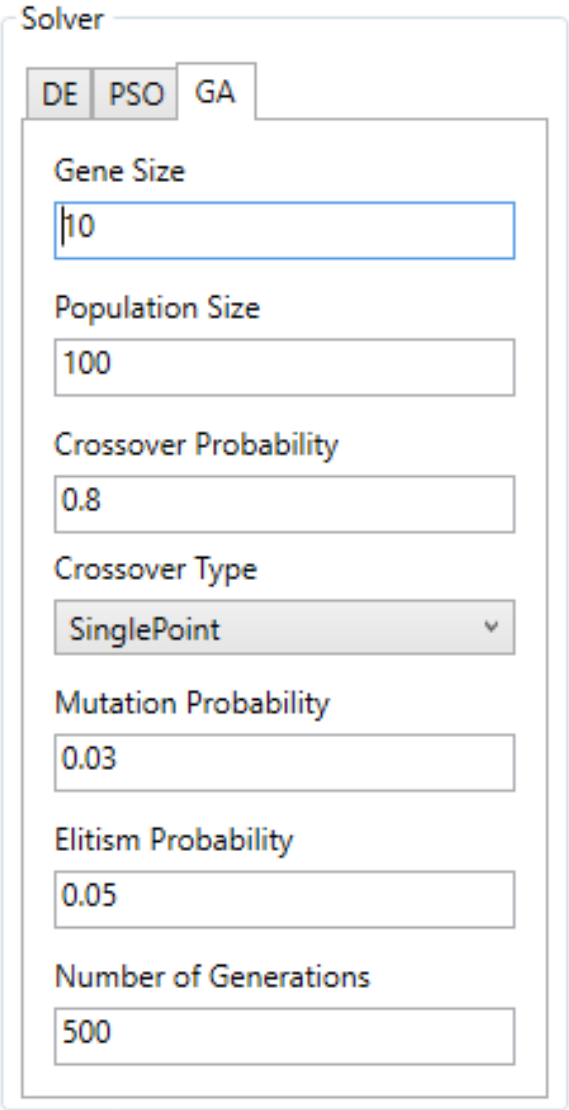


Fonte: Autoria Própria

- **Elitism Probability:** porcentagem dos melhores indivíduos que serão propagados para a próxima geração sem nenhuma alteração;
- **Number of Generations:** número máximo de gerações que o algoritmo irá evoluir até parar.

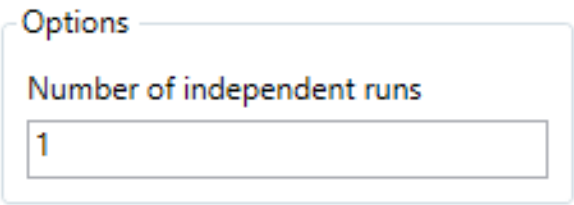
A quarta área é bem simples, pode ser visualizada na Figura 15, e permite ao usuário selecionar quantas execuções independentes serão realizadas com as configurações atuais. Por exemplo, se o usuário preencher com o número 10, o *dataset* selecionado será resolvido com o algoritmo e configurações especificados 10 vezes consecutivas. Esta funcionalidade é importante porque os algoritmos nem sempre produzem exatamente os mesmos resultados e é importante que ele seja executado várias vezes para avaliar a média dos resultados e qual foi o melhor resultado.

Figura 14: Área de configuração do algoritmo GA



Fonte: Autoria Própria

Figura 15: Área de configuração do número de execuções independentes



Fonte: Autoria Própria

A quinta e última área da tela principal pode ser vista na Figura 16 e tem como objetivo exibir os resultados gerados pelos algoritmos. Ela se divide em 3 grupos, de acordo com os

resultados exibidos por elas:

- **Dados da execução atual:** quando o algoritmo for executado com mais de uma execução independente, é possível navegar pelo resultado de cada execução utilizando os botões de navegação na parte superior da tela. Nesse caso, esta área exibe o valor total de satisfação (*Profit*), custo de desenvolvimento (*Cost*) e tempo de execução (*Elapsed Time*) da execução selecionada.
- **Dados de média de todas as execuções:** esta exibe o total de execuções independentes (*Number of Solutions*); lucro mínimo, médio e máximo de todas as execuções independentes (*Profit*) e o valor de satisfação da melhor de todas as soluções (*Best Solution*).
- **Gráfico de *fitness* da execução atual:** o gráfico de evolução da função de *fitness* da execução selecionada é exibido nessa área, onde a linha azul representa o *fitness* máximo da população em cada geração, a linha amarela o *fitness* médio e a linha vermelha o *fitness* mínimo. O eixo X representa as gerações e o eixo Y o valor do *fitness*.

4.8 CONSIDERAÇÕES SOBRE ESTE CAPÍTULO

Neste capítulo foram apresentados os métodos propostos para atingir o objetivo da pesquisa. O trabalho consistiu basicamente em levantar novos *datasets* de requisitos de projetos reais, desenvolver uma nova codificação para aplicar nos *datasets* propostos e o desenvolvimento dos algoritmos das meta-heurísticas selecionadas: GA, PSO e DE.

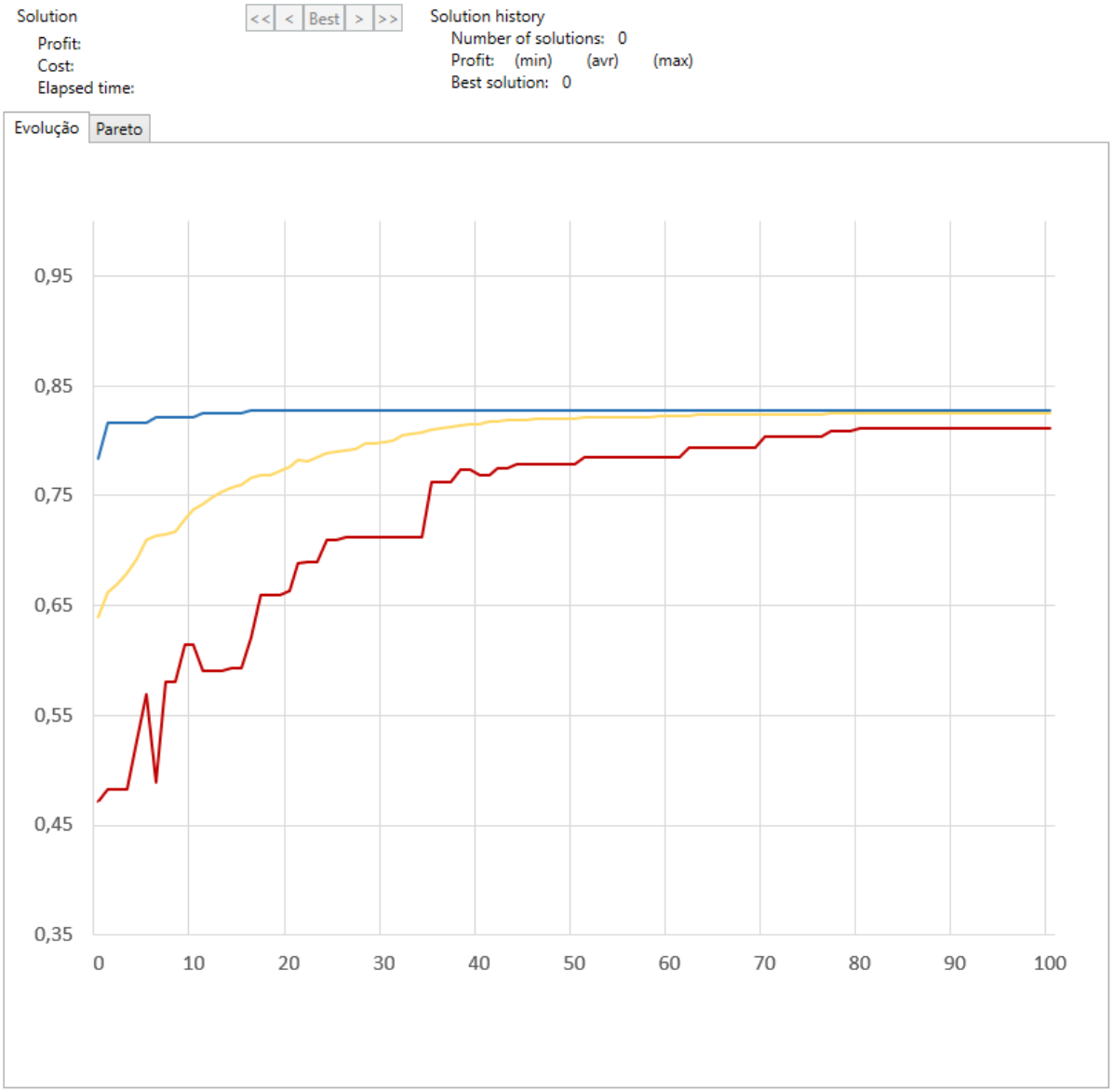
Além disso, foi elaborado um algoritmo para calcular a solução exata das instâncias propostas e elaborado um estudo de caso com alunos de graduação e mestrado.

Por fim, o trabalho resultou no desenvolvimento de um protótipo de software para aplicar os algoritmos das meta-heurísticas e coletar os resultados mais facilmente. Essa ferramenta junto dos datastes propostos e todo seu código fonte foram disponibilizados publicamente no Github do autor [2].

O capítulo que se segue apresentará os resultados encontrados nos experimentos propostos neste capítulo.

[2] https://github.com/rbasniak/NRP

Figura 16: Área de exibição de resultados



Fonte: Autoria Própria

5 RESULTADOS E DISCUSSÃO

Este capítulo apresenta todos os resultados obtidos pelos algoritmos e estudos de caso apresentados no capítulo anterior. Os problemas referentes aos três projetos selecionados foram resolvidos utilizando os algoritmos de Evolução Diferencial, Algoritmos Genéticos e Otimização por Enxame de Partículas. Com relação a estes algoritmos todos eles foram executados em duas versões: com codificação binária e penalização progressiva e com codificação contínua.

Para o projeto CBS, além das abordagens citadas acima, ainda foram utilizadas duas outras soluções. A primeira foi encontrar a solução exata por meio de um algoritmo de busca exaustiva e a segunda abordagem foi encontrar a solução manual do projeto com o auxílio de alunos de graduação e mestrado de cursos de Computação da UTFPR.

Todos os experimentos computacionais foram realizados utilizando um computador com processador Intel i7-2600 3.4GHz sem utilização de *multi threading* ou algoritmos em paralelo.

5.1 BASELINE

Como os *datasets* propostos neste trabalho são inéditos na literatura não existem soluções ótimas ou exatas conhecidas ainda. Por isso os resultados de referência (*baseline*) adotados serão os resultados que foram utilizados durante o desenvolvimento destes projetos, que são os únicos resultados conhecidos antes da solução através das meta-heurísticas. Estes resultados são apresentados na Tabela 5.

Tabela 5: Resultados de *baseline*. Fonte: Autoria Própria

Dataset	Satisfação
CBS	1733
FPG	3896
SRT	11596

5.2 SOLUÇÃO EXATA

Para o projeto CBS, que possui apenas 32 requisitos, o total de soluções possíveis para o problema é dado pela Equação (12) e existem um total de 4.294.967.296 soluções.

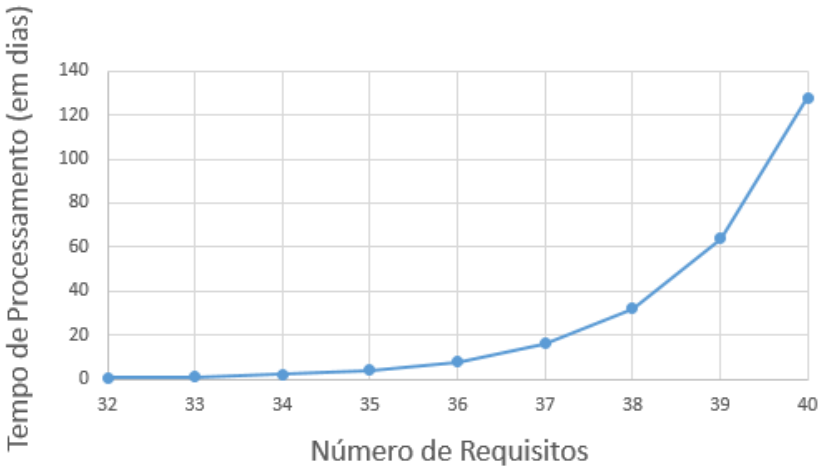
$$\text{Soluções} = 2^{\text{Número de Requisitos}} \tag{12}$$

Apesar de ser um problema bem pequeno e com poucos requisitos, o espaço de busca é relativamente grande, o total de horas gastas para encontrar a solução exata do projeto CBS foi de 12 horas.

O maior valor de satisfação que pode ser obtido é de 1733, e apenas 8 soluções das 4.2 bilhões possíveis possuem este valor. A diferença entre elas é que o custo varia de 233 a 238 e o risco de 61 a 67. Baseado no hardware utilizado, a Figura 17 mostra o tempo de processamento necessário, em dias, de acordo com o número total de requisitos.

Se fossem adicionados mais 8 requisitos ao projeto CBS, alterando o número de requisitos para 40, seriam necessários 120 dias para encontrar a solução exata. Infelizmente este mesmo procedimento não pode ser realizado para os outros dois projetos valendo-se da mesma metodologia e hardware utilizados. Com a Equação (12) é possível determinar que existem 1.115E43 possíveis soluções para o projeto FPG e 1.13E74 para o SRT. Assumindo que cada iteração do algoritmo para estes projetos duraria o mesmo tempo que uma iteração do algoritmo do CBS, seriam necessários 6.64E35 e 1.32E64 dias para os projetos FPG e SRT respectivamente.

Figura 17: Evolução do tempo de processamento com o aumento dos requisitos



Fonte: Autoria Própria

Como foi inviável tentar encontrar a solução exata para os projetos FPG e SRT estes projetos não puderam ter sua solução exata descoberta neste trabalho.

5.3 ESTUDO DE CASO COM ALUNOS

Um total de 35 alunos participaram deste experimento e foram disponibilizadas 2 horas para que eles apresentassem uma solução par o projeto. Os resultados podem ser vistos na Tabela 6. Os alunos que não possuem dados nos resultados são alunos que não conseguiram obter uma solução válida, ou seja, não respeitaram alguma restrição do problema.

Tabela 6: Resultados obtidos pelos alunos. Fonte: Autoria Própria

Aluno	Valor	Custo	Risco
1	1728	236	63
2	1728	236	63
3	1441	238	51
4	1728	236	63
5	-	-	-
6	-	-	-
7	-	-	-
8	-	-	-
9	1673	228	58
10	-	-	-
11	-	-	-
12	-	-	-
13	1733	236	61
14	-	-	-
15	-	-	-
16	-	-	-
17	-	-	-
18	-	-	-

Aluno	Valor	Custo	Risco
19	1728	236	63
20	-	-	-
21	-	-	-
22	1728	236	63
23	-	-	-
24	-	-	-
25	1700	236	57
26	1733	236	61
27	1733	236	61
28	1688	223	61
29	-	-	-
30	1733	236	61
31	1699	236	60
32	1713	228	61
33	-	-	-
34	1733	236	61
35	1733	236	61

Este *dataset* foi adicionado ao trabalho justamente por ser pequeno, para que fosse possível realizar este estudo de caso. Ele é um *dataset* com poucos requisitos e por isso relativamente fácil de resolver, portanto era esperado que alguns alunos encontrassem a solução exata. O que não era esperado era que uma parcela tão grande de alunos não conseguisse encontrar uma solução válida. Apenas 48.6% dos alunos conseguiram obter uma resposta válida para o problema proposto e 17.1% conseguiram obter a solução ideal para o problema.

O principal ponto a ser observado neste experimento é que mais da metade dos alunos não conseguiram achar uma solução válida, o que reforça a complexidade do problema quando são consideradas todas as interdependências entre os requisitos.

Assim como na solução exata, esta abordagem só foi utilizada no projeto CBS, por ser um projeto pequeno. Os outros projetos seriam inviáveis de se utilizar em sala de aula, pois são projetos complexos e não seria possível fazer com que os alunos entendessem todos os detalhes dos projetos no tempo disponível em sala de aula.

5.4 CODIFICAÇÃO BINÁRIA COM PENALIZAÇÃO PROGRESSIVA

Esta codificação foi utilizada neste trabalho para servir como comparação da codificação proposta e por ser um método muito utilizado na literatura. Porém, ao adicionar as interdependências no problema ele se torna um problema altamente restrito e essa codificação teve problemas para encontrar soluções nos projetos FRG e SRT devido ao grande número de requisitos e restrições.

Como a população inicial é gerada aleatoriamente nos métodos meta-heurísticos selecionados para este trabalho, muitos indivíduos da população serão inválidos. Ou seja, não atenderão a alguma das restrições do problema, seja devido a uma das interdependências ou do custo.

Para tentar amenizar esse efeito foi adicionado um sistema de penalização progressiva, onde a cada geração um valor é subtraído do *fitness* da solução para cada restrição que ela não atende. Esse valor vai crescendo ao longo das gerações para favorecer as soluções válidas, e dar mais diversidade nas populações iniciais.

Mesmo o projeto CBS, que é o menor, possui 24 restrições no total e fazendo com que poucas soluções sejam válidas na população inicial. Além disso, a grande quantidade de restrições dificulta a geração de uma nova solução válida a partir de uma existente, já que o cruzamento de duas soluções pode não atender a alguma restrição do problema. A Tabela 7 mostra os resultados obtidos utilizando esta codificação.

Tabela 7: Resultados obtidos utilizando a codificação binária. Fonte: Autoria Própria

	CBS		FPG		SRT	
Método	**Média**	**Melhor Valor**	**Média**	**Melhor Valor**	**Média**	**Melhor Valor**
GA	1573,6	1664	-	-	-	-
PSO	1708,8	1733	-	-	-	-
DE	1398,6	1625	-	-	-	-

Os três métodos conseguiram chegar em uma solução válida par ao problema e, o PSO até chegou na solução exata em uma das tentativas. Entretanto, para os projetos FPG e SRT os algoritmos não conseguem gerar nenhuma solução válida. O número de restrições nesses

dois projetos é grande e mesmo tentando ajustar o sistema de penalização com vários valores os algoritmos não conseguiram resolver o problema.

Tal fato só reforça a situação existente na literatura, na qual na maioria das vezes, os estudos não levam em conta as interdependências entre os requisitos. Ao remover as interdependências do problema, ele passa a ter apenas uma restrição, a de custo, e com isso qualquer método consegue gerar mais soluções válidas na população inicial.

5.5 CODIFICAÇÃO CONTÍNUA

A codificação contínua proposta neste trabalho obteve bons resultados e conseguiu resolver os 3 *datasets* sem problemas. A seguir serão apresentados os resultados obtidos por cada meta-heurística separadamente.

5.5.1 EVOLUÇÃO DIFERENCIAL

Para o projeto CBS, o algoritmo de DE encontra a solução exata com praticamente qualquer parâmetro de configuração. O que muda é a qualidade geral das soluções encontradas e quão rápido o algoritmo converge para a solução exata.

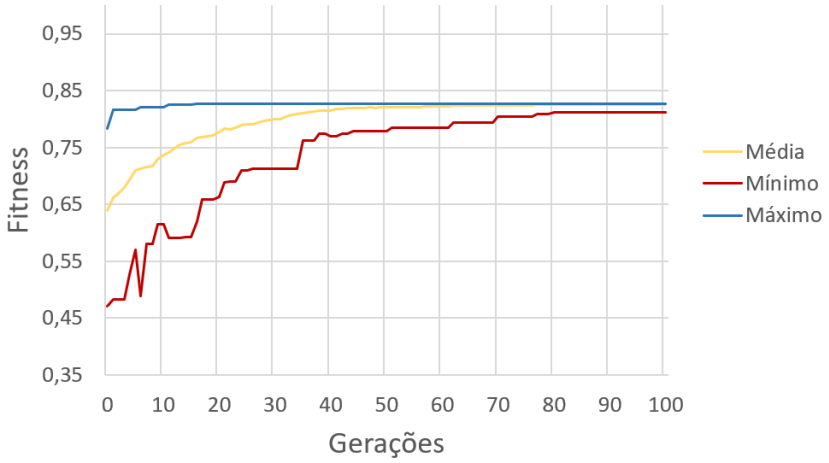
O parâmetro que mais afeta os resultados é o peso diferencial, que deve ser 0.85. Quaisquer combinações com os outros parâmetros resultam no algoritmo sempre encontrando a solução exata em todas as tentativas, apenas variando o número de gerações que ele leva para chegar na solução. Quando o algoritmo converge em menos de 50 gerações existe uma tendência de ficar preso em algum máximo local, mesmo chegando na solução exata algumas vezes. Em contrapartida, todas as configurações que convergem em aproximadamente 100 gerações tendem a chegar na solução exata na maioria das vezes.

A Figura 18 mostra uma típica curva de convergência para este projeto. É possível notar que apesar do algoritmo convergir em aproximadamente 100 gerações a solução exata já havia sido gerada antes da 20ª geração.

Para os projetos FPG e SRT, as curvas podem ser vistas nas Figuras 19 e 20. As três curvas ficaram parecidas, com uma convergência suave e contínua. O aumento do número de requisitos no problema estende o número de gerações necessárias para o algoritmo convergir. Então, para problemas maiores, é necessário deixar o algoritmo rodando por mais tempo para chegar nas melhores soluções.

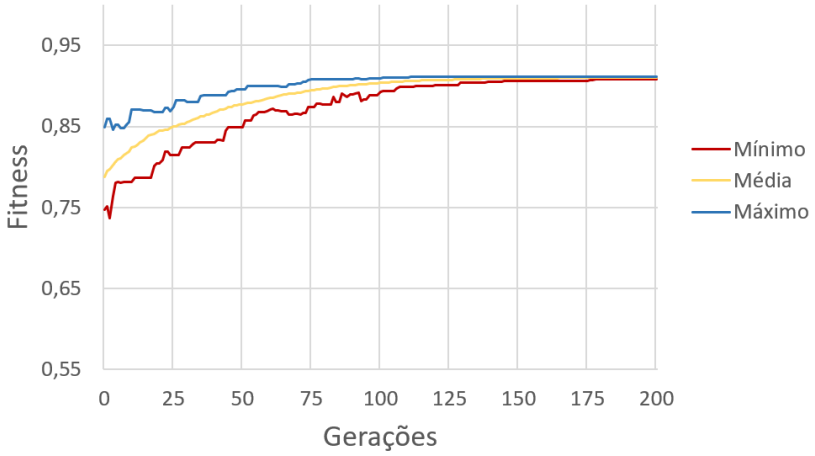
Baseado nos dados obtidos, esta proporção parece ser linear e é possível aproximar o

Figura 18: Curva de *fitness* do projeto CBS



Fonte: Autoria Própria

Figura 19: Curva de *fitness* do projeto FPG



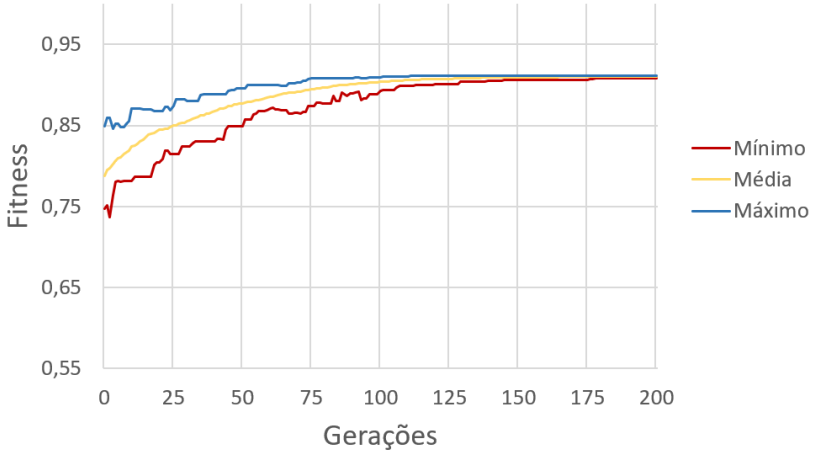
Fonte: Autoria Própria

número de gerações necessárias para se chegar na solução utilizando a Equação (13). Sendo n é o número de requisitos do projeto e i é o número de gerações necessárias para o algoritmo de DE chegar na sua melhor solução. Apenas 3 projetos é um número pequeno para confirmar este comportamento, mas ele serve como um valor de referência para problemas maiores.

$$\mathrm{i} = 0.95n + 70 \tag{13}$$

A Tabela 8 apresenta os melhores resultados encontrados por meio de evolução diferencial para cada um dos projetos. A coluna "Iterações" representa a quantidade de vezes que a função objetivo foi resolvida, e é obtida pelo produto do número aproximado de gerações para encontrar a melhor solução e o tamanho da população. O Apêndice B apresenta todos os

Figura 20: Curva de *fitness* do projeto SRT



Fonte: Autoria Própria

resultados gerados utilizando diversas configurações dos parâmetros de controle.

Tabela 8: Resultados obtidos utilizando a codificação contínua e DE. Fonte: Autoria Própria

Instância	População	Crossover	Peso Diferencial	Iterações	Média	Resultado
CBS	50	0,15	0,85	5000	1733	1733
FPG	50	0,15	0,15	10000	4185	4185
SRT	50	0,15	0,15	15000	13645,8	13659

Este algoritmo se mostrou muito eficiente para esta instância do problema, convergindo para a solução exata analisando apenas 0.00011% de todo o espaço de busca disponível.

O aumento do número de requisitos no projeto FPG deixou o problema mais complexo, pois o algoritmo não consegue mais encontrar a melhor solução conhecida com qualquer combinação dos parâmetros de configuração como no projeto CBS.

Para o projeto FPG, a melhor solução encontrada por este método foi de 4185, utilizando população de 50 indivíduos, taxa de crossover de 0.15 e peso diferencial de 0.15. Esta é a única configuração que consegue encontrar o valor em todas as tentativas. Algumas outras configurações também conseguem, mas apenas em algumas das 10 tentativas. O Apêndice B apresenta todos os resultados gerados utilizando diversas configurações dos parâmetros de controle.

Enfim, o projeto SRT confirma que o aumento do número de requisitos deixa o projeto bem mais complexo para o algoritmo resolver. Apenas com taxa de crossover de 0.15 e peso diferencial de 0.15 é possível chegar na melhor solução conhecida, e não em todas as 10 tentativas. Com estes parâmetros o melhor valor de satisfação encontrado para o projeto SRT foi de

13659. O Apêndice B apresenta todos os resultados gerados utilizando diversas configurações dos parâmetros de controle.

O número de iterações necessárias para chegar na solução aumentou com o aumento da quantidade de requisitos. Do projeto CBS para o SRT a quantidade de requisitos aumentou 7,7 vezes enquanto o número de iterações necessárias aumentou apenas 3 vezes. Ou seja, a taxa do esforço necessário para resolver o problema cresce com menos da metade da taxa de crescimento do problema.

5.5.2 OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS

A Tabela 9 apresenta os melhores resultados encontrados com otimização por enxame de partículas para cada um dos projetos. A coluna "Iterações" representa a quantidade de vezes que a função objetivo foi resolvida, e é obtida pelo produto do número aproximado de gerações para encontrar a melhor solução e a quantidade de partículas.

Tabela 9: Resultados obtidos utilizando a codificação contínua e PSO. Fonte: Autoria Própria

Instância	Partículas	Peso Inercial	Fator Cognitivo	Fator Social	Iterações	Média	Resultado
CBS	100	0.75	1.50	1.50	5000	1731.9	1733
FPG	75	0.25	2.00	2.00	37500	4168.3	4185
SRT	75	0.25	2.00	2.00	37500	13444.7	13576

O PSO se mostrou bem mais dependente dos parâmetros de configuração, mostrando resultados mais variáveis e nem sempre chegando nas melhores soluções conhecidas, principalmente nos projetos maiores.

O projeto CBS continuou se mostrando bem simples para o algoritmo, visto que todas combinações de parâmetros utilizadas acabaram chegando na solução exata em ao menos uma das 10 rodadas.

Porém, ao contrário do DE, não houve nenhuma configuração que conseguiu chegar na solução exata em todas as rodadas. A configuração que apresentou melhor desempenho foi utilizando 100 partículas, peso inercial de 0.75, fatores social e cognitivo de 1.50. Esta configuração chega na solução exata depois de 5000 iterações pela função objetivo e a média das soluções é de 1731.9.

Para este projeto, valores maiores de peso inercial tendem a gerar melhores soluções. O Apêndice B apresenta todos os resultados gerados utilizando diversas configurações dos parâmetros de controle.

No projeto FPG o melhor resultado foi 4185, a mesma solução encontrada pelo algoritmo de evolução diferencial. Somente uma combinação de parâmetros chegou neste resultado, e foi utilizando 75 partículas, peso inercial de 0.25, fatores social e cognitivo de 2.00. Esta solução foi obtida depois de 37500 iterações pela função objetivo e a média foi 4168.3.

No projeto SRT, a melhor solução encontrada foi 13576, ou seja, inferior à solução encontrada pelo algoritmo de evolução diferencial. Esta solução foi encontrada depois de 37500 iterações pela função objetivo utilizando exatamente a mesma configuração do projeto FPG.

Assim como na evolução diferencial, o projeto CBS obteve melhores soluções utilizando parâmetros de controle um pouco diferentes dos outros dois projetos. Entretanto, sem mais projetos diferentes para testar, não é possível identificar o motivo. Apesar disso, tal fato ressalta a importância de realizar testes com diferentes parâmetros, pois é possível observar que, mesmo dentro de uma mesma família de problemas, os algoritmos podem se comportar de forma diferente em diferentes problemas.

5.5.3 ALGORITMOS GENÉTICOS

O último método a ser aplicado para resolver as três instâncias propostas foi o de algoritmos genéticos. Por ser uma meta-heurística bem difundida e fácil de aplicar ela vai ajudar a balizar os resultados obtidos pelos outros métodos aplicados.

A Tabela 10 apresenta os melhores resultados encontrados utilizando algoritmos genéticos para cada um dos projetos. A coluna "Iterações" representa a quantidade de vezes que a função objetivo foi resolvida, e é obtida pelo produto do número aproximado de gerações para encontrar a melhor solução e o tamanho da população.

Tabela 10: Resultados obtidos utilizando a codificação contínua e GA. Fonte: Autoria Própria

Instância	População	Mutação	Crossover	Elitismo	Iterações	Média	Resultado
CBS	50	0.05	0.80	0.05	2500	1732.5	1733
FPG	100	0.03	0.85	0.05	20000	4183.6	4185
SRT	100	0.03	0.80	0.05	30000	13567.2	13644

Novamente o projeto CBS chegou em seus melhores resultados com configurações um pouco diferentes dos outros dois. O projeto SRT não chegou no melhor resultado conhecido, apesar de ter ficado mais próximo que o resultado obtido pelo PSO.

5.6 COMPILAÇÃO DE RESULTADOS

A Tabela 11 apresenta uma compilação dos resultados obtidos por todos os métodos utilizado em cada projeto. A primeira linha, apresenta o *baseline* de resultados (obtido durante o desenvolvimento destes projetos) e por isso não possuem a informação de média visto que só existe uma solução.

Tabela 11: Compilação de resultados obtidos por todos os métodos. Fonte: Autoria Própria

		CBS		FPG		SRT	
Método		**Média**	**Melhor Valor**	**Média**	**Melhor Valor**	**Média**	**Melhor Valor**
Real		-	1733,0	-	3896,0	-	11596,0
Exato		-	1733,0	-	-	-	-
Alunos		827,2	1733,0	-	-	-	-
GA	Tradicional	1573,6	1664,0	-	-	-	-
	Proposto	1732,5	1733,0	4183,6	4185,0	13567,2	13644,0
PSO	Tradicional	1708,8	1733,0	-	-	-	-
	Proposto	1731,9	1733,0	4168,3	4185,0	13444,7	13576,0
DE	Tradicional	1398,6	1625,0	-	-	-	-
	Proposto	1733,0	1733,0	4185,0	4185,0	13645,8	13659,0

No geral, todos os métodos computacionais utilizados obtiveram uma boa performance, até chegando nas melhores soluções conhecidas para os projetos CBS e FPG. Entretanto a performance do DE foi superior, pois em todas as tentativas e utilizando a combinação certa de parâmetros ele sempre chegou na melhor solução conhecida para ambos os projetos.

O projeto SRT foi o que deu o maior trabalho para os algoritmos, pois foi o único projeto que a melhor solução divergiu nos três algoritmos. Ainda assim, o DE foi o que obteve a melhor solução. A diferença é pequena, mas ainda assim, superior.

A codificação binária fica atrás de todos os métodos testados, o que condiz com a literatura, visto que esta codificação geralmente é utilizada apenas em projetos de até 20 requisitos (quase metade do menor projeto utilizado aqui, com 32 requisitos), ou em projetos sem interdependências.

O fato de que entre os algoritmos com a codificação binária o PSO obteve o melhor resultado também vai de encontro com os resultados encontrados na revisão sistemática sobre o Problema de Mochila. Na revisão o PSO é o segundo algoritmo mais utilizado e com ótimos resultados mas grande parte destes trabalhos é sobre o problema de mochila básico e sem restrições.

Entretanto, conforme a quantidade de interdependências começa a crescer, como nos projetos FPG e SRT, a codificação binária não consegue mais resolver os problemas devido ao elevado número de restrições. Em contrapartida a codificação contínua sempre obtém ótimos resultados indiferente do tamanho do projeto e da quantidade de restrições.

Outro ponto é que a codificação proposta não só conseguiu resolver o problema mas também obteve um melhor resultado do que a solução que foi implementada no projeto real. Conforme o projeto aumenta a distância entre as soluções cresce também. Para o projeto FPG a melhor solução encontrada pelos algoritmos foi 7,4% melhor e para o projeto SRT 14,7% melhor.

Assim pode-se concluir que se a distância entre as soluções implementadas e as encontradas por métodos computacionais continuarem se distanciando para projetos maiores a codificação contínua, aliada com o algoritmo DE, pode ser uma ferramenta de grande auxílio no processo decisório de quais requisitos devem ser implementados ou não.

5.7 AMEAÇAS À VALIDADE

Os resultados obtidos foram satisfatórios ao utilizar a codificação contínua proposta, mas em contrapartida, alguns pontos limitadores foram identificados:

- os conjuntos de dados levantados não possuem um número muito grande de requisitos, e equivalem a projetos de pequeno e médio porte;
- como os conjuntos de requisitos propostos são inéditos na literatura, não existem soluções ótimas conhecidas;
- o estudo realizado com profissionais só pôde ser feito com alunos de graduação e pós-graduação e nem todos trabalham na área de seleção de requisitos;
- a solução exata, de acordo com o modelo proposto, só pôde ser obtida para o menor das bases de dados.

5.8 CONSIDERAÇÕES SOBRE ESTE CAPÍTULO

Neste capítulo, pudemos observar que os resultados obtidos pelas meta-heurísticas foram muito satisfatórios quando aliadas à codificação contínua, pois com a codificação binária, somente problemas menores foram resolvidos neste trabalho. Ainda assim nem todas as técnicas conseguiram chegar na solução exata, mesmo sendo um projeto pequeno. A codificação

contínua também obteve resultados superiores aos obtidos pelos especialistas durante a implementação dos projetos.

Além disso, a dificuldade do problema ficou bem clara ao se estimar o tempo de processamento que seria necessário para resolver os *datasets* maiores e com a grande parcela de alunos que não conseguiram se quer gerar uma solução válida para o menor projeto.

O próximo capítulo apresentará as conclusões finais sobre tudo que foi exposto ao longo deste trabalho, bem como as contribuições e as oportunidades de trabalhos futuros.

6 CONCLUSÃO

Com o passar dos anos a capacidade de produção de software aumentou, e com isso a complexidade dos sistemas também. O processo de seleção de requisitos tem um papel muito importante no desenvolvimento e no projeto todo. Idealmente, os requisitos selecionados devem atender às necessidades dos *stakeholders* da melhor forma possível. Desta forma, a motivação da pesquisa foi a aplicação de técnicas de otimização para priorizar os requisitos levando em consideração o grau de satisfação dos *stakeholders*, atendendo às restrições do projeto e dos requisitos solicitados.

Como a abordagem escolhida para o tratamento do problema inclui as dependências entre os requisitos, o problema fica muito complexo e com muitas restrições. Portanto, o desenvolvimento do sistema de codificação contínua proposto e que pode ser utilizado com qualquer meta-heurística foi de vital importância para o sucesso dos resultados e acabou se tornando uma das principais contribuições deste trabalho.

O modelo matemático foi especificado e planejado para ser utilizado com técnicas meta-heurísticas, tais como Algoritmos Genéticos, Evolução Diferencial e Otimização por Enxame de Partículas, utilizados neste trabalho. Para resolução do modelo, foram levantados conjuntos de requisitos de projetos reais e que já foram concluídos, nos quais é possível comparar a solução realmente implementada com as soluções propostas pelos algoritmos. Além disso, os dados numéricos destes projetos serão disponibilizados publicamente para que outros pesquisadores possam utilizá-los.

Todos os algoritmos se mostraram muito estáveis no que diz respeito à variações nos parâmetros de configuração devido à codificação proposta, o que é bom, pois, quer dizer que a codificação proposta gera uma superfície de soluções mais uniforme, sem grandes picos ou vales, e isto por sua vez influencia diretamente na capacidade dos algoritmos de chegarem próximo das soluções exatas.

Com isso, a codificação proposta se mostrou bem promissora e, principalmente, robusta para problemas com muitas interdependências entre os requisitos. Isso se deve ao fato de

que ela sempre irá gerar respostas válidas para o problema, mesmo sem usar sistemas de penalização ou operadores de reparação. Ela se mostrou como uma das maiores contribuições deste trabalho, pois pode ser utilizada em outros problemas de otimização com um elevado número de restrições e acoplamento entre os itens.

Dos três algoritmos testados, o DE se mostrou superior ao PSO e GA não só no quesito de qualidade e estabilidade das soluções geradas, mas também no número de iterações necessárias para chegar nas soluções ótimas, tempo de execução e complexidade de implementação.

Considera-se que os objetivos elencados na Seção 1.2 foram atingidos ao longo do trabalho, apesar das limitações levantadas durante a pesquisa. Apesar de maiores que as bases de dados reais, as bases utilizadas neste trabalho podem não refletir a realidade de projetos de grande porte. Por serem bases de dados inéditas ainda não possuem soluções ótimas conhecidas, e por isso não foi possível avaliar a performance real do método proposto. Foi por este motivo que diversos algoritmos foram utilizados e dois tipos diferente de codificação, para ter uma avaliação de performance relativa entre eles. Apesar disso, a solução exata foi encontrada para a menor base de dados e o método proposto conseguir chegar nela, enquanto o método geralmente utilizado na literatura não conseguiu. Isso leva a bons indícios de que a performance nas bases de dados maiores pode ter sido próxima ao método exato. Por fim, no experimento realizado com alunos nem todos tinham experiência ou trabalham na área de requisitos de software, pois alguns ainda estão na graduação.

Finalmente, pode-se concluir que a interpretação e a modelagem correta do problema é essencial para o sucesso do processo de otimização, pois nem sempre mudar o algoritmo para um mais eficiente melhorará os resultados obtidos se uma codificação ineficiente estiver sendo utilizada.

6.1 CONTRIBUIÇÕES

As principais contribuições deste trabalho foram a) a nova codificação contínua para problemas de otimização altamente restritos e b) novos *datasets* públicos para a seleção de requisitos.

Já foi explicado de porque a codificação se mostrou eficiente para o NRP, mas a principal contribuição relacionada à codificação é que ela possui indícios de escalabilidade. O fato de que ela pode ser usada em qualquer problema de otimização combinatorial com elevado número de restrições, independente do número ou tipo de restrição, à torna muito versátil.

Os *datasets* propostos diferem dos outros já propostos na literatura em alguns pontos:

a) são baseados em projetos reais; b) possuem um grande número de interdependências entre os requisitos. Isso os torna diferentes e como foram disponibilizados publicamente, podem ser utilizados outros pesquisadores desta área de estudo.

Também foi aceito um artigo com uma revisão sistemática sobre o *Next Release Problem*, no CBSoft 2017 - Workshop de Engenharia de Software (CBSoft 2017 - WESB) [1].

- BASNIAK, R. et al. Métodos Evolucionários e o Next Release Problem. In: VIII WORKSHOP DE ENGENHARIA DE SOFTWARE BASEADO EM BUSCA (WESB), 2017, Fortaleza: CBSOFT, 2017

6.2 TRABALHOS FUTUROS

Entre as oportunidades para trabalhos futuros, é possível observar a) criação de um sistema funcional para que se permita a aplicação do método proposto neste trabalho para solução de projetos reais em empresas de software; b) utilização e testes da abordagem proposta em projetos de grande porte e em mais projetos reais; c) utilização de outras meta-heurísticas para obtenção dos resultados e comparação com os apresentados neste trabalho; e) aplicação do método de codificação proposto em outros problemas de otimização com elevado número de restrições; f) Como os *datasets* propostos possuem a informação de grau de risco do requisito mudar ao longo do projeto, seria possível explorar uma versão multiobjetivo do problema, maximizando a satisfação e minimizando o risco total.

[1] http://www.lia.ufc.br/ cbsoft2017/programacao-wesb/. Último Acesso: 14/08/2019

REFERÊNCIAS

ACHIMUGU, P. et al. A systematic literature review of software requirements prioritization research. **Information and Software Technology**, v. 56, 06 2014.

ALLIANCE, A. **Agile Practice Guide, Project Management Institute, 2017: Agile Practice Guide**. Project Management Institute, Incorporated, 2017. (Agile Practice Guide). ISBN 9781628251999. Disponível em: <https://books.google.com.br/books?id=zbzjAQAACAAJ>.

BAGNALL, A.; RAYWARD-SMITH, V.; WHITTLEY, I. The next release problem. **Information and Software Technology**, v. 43, n. 14, p. 883 – 890, 2001. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S095058490100194X>.

BAKER, P. et al. Search based approaches to component selection and prioritization for the next release problem. In: . [S.l.: s.n.], 2006. p. 176–185.

BECCENERI, J. et al. Meta-heurísticas e otimização combinatória: Aplicações em problemas ambientais. 01 2008.

BHATTACHARJEE, K.; SARMAH, S. P. A binary firefly algorithm for knapsack problems. In: . [S.l.: s.n.], 2015. p. 73–77.

BONABEAU, E.; DORIGO, M.; THERAULAZ, G. Swarm intelligence : From natural to artificial systems / e. bonabeau, m. dorigo, g. theraulaz. 01 2001.

CARLSHAMRE, P. et al. An industrial survey of requirements interdependencies in software product release plannin. In: . [S.l.: s.n.], 2001. p. 84 – 91. ISBN 0-7695-1125-2.

CHAVES-GONZáLEZ, J. M.; PÉREZ-TOLEDANO, M. A. Differential evolution with pareto tournament for the multi-objective next release problem. **Appl. Math. Comput.**, Elsevier Science Inc., New York, NY, USA, v. 252, n. C, p. 1–13, fev. 2015. ISSN 0096-3003. Disponível em: <http://dx.doi.org/10.1016/j.amc.2014.11.093>.

CHENG, S.-L.; HWANG, C. Optimal approximation of linear systems by a differential evolution algorithm. **IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans**, v. 31, n. 6, p. 698–707, Nov 2001. ISSN 1083-4427.

DANESH, A. S.; MORTAZAVI, S. M.; DANESH, S. Y. S. Requirements prioritization in online banking systems: Using value-oriented framework. In: **2009 International Conference on Computer Technology and Development**. [S.l.: s.n.], 2009. v. 1, p. 158–161.

DANEVA, M. et al. Agile requirements prioritization in large-scale outsourced system projects: An empirical study. **Journal of Systems and Software**, v. 86, p. 1333–1353, 03 2013.

DEEP, K.; BANSAL, J. C. A socio-cognitive particle swarm optimization for multi-dimensional knapsack problem. In: **2008 First International Conference on Emerging Trends in Engineering and Technology**. [S.l.: s.n.], 2008. p. 355–360. ISSN 2157-0477.

DENG, C. et al. Modified dynamic differential evolution for 0-1 knapsack problems. 12 2009.

DUAN, C. et al. Towards automated requirements prioritization and triage. **Requir. Eng.**, v. 14, p. 73–89, 06 2009.

DURILLO, J. J. et al. A study of the bi-objective next release problem. **Empirical Software Engineering**, Springer, v. 16, n. 1, p. 29–60, 2011.

FEHLMANN, T. New lanchester theory for requirements prioritization. In: . [S.l.: s.n.], 2008. p. 35 – 40.

FINKELSTEIN, A. et al. A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making. **Requirements engineering**, Springer, v. 14, n. 4, p. 231–245, 2009.

GAREY, M. R. Computers and intractability : A guide to the theory of np-completeness. **Revista Da Escola De Enfermagem Da U S P**, v. 44, n. 2, p. 340, 1979. Disponível em: <https://ci.nii.ac.jp/naid/10030608028/en/>.

GHERBOUDJ, A.; SAID, L. A new hybrid binary particle swarm optimization algorithm for multidimensional knapsack problem. In: . [S.l.: s.n.], 2012. v. 166.

GOLDBERG, D. E.; DEB, K. A comparative analysis of selection schemes used in genetic algorithms. In: **FOGA**. [S.l.: s.n.], 1990.

GREER, D.; RUHE, G. Software release planning: An evolutionary and iterative approach. **Information and Software Technology**, v. 46, p. 243–253, 03 2004.

GUO, Z. et al. A thermodynamical selection-based discrete differential evolution for the 0-1 knapsack problem. **Entropy**, v. 16, p. 6263–6285, 12 2014.

HADDAR, B. et al. A hybrid quantum particle swarm optimization for the multidimensional knapsack problem. **Engineering Applications of Artificial Intelligence**, v. 55, p. 1–13, 10 2016.

HAN, K.-H.; KIM, J.-H. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. **Evolutionary Computation, IEEE Transactions on**, v. 6, p. 580 – 593, 01 2003.

HE, F. An improved particle swarm optimization for knapsack problem. In: . [S.l.: s.n.], 2010. p. 1 – 4.

HOLLAND, J. H. **Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence**. Oxford, England: U Michigan Press, 1975.

HOTA, A. R.; PAT, A. An adaptive quantum-inspired differential evolution algorithm for 0–1 knapsack problem. In: . [S.l.: s.n.], 2011. p. 703 – 708.

ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary. **ISO/IEC/IEEE 24765:2010(E)**, p. 1–418, Dec 2010.

JANTUNEN, S. et al. The challenge of release planning. In: **2011 Fifth International Workshop on Software Product Management (IWSPM)**. [S.l.: s.n.], 2011. p. 36–45.

JUN, S.; JIAN, L. Solving 0-1 knapsack problems via a hybrid differential evolution. **3rd International Symposium on Intelligent Information Technology Application, IITA 2009**, v. 3, 01 2009.

KARLSSON, C.; AHLSTRöM, P. The difficult path to lean product development. **Journal of Product Innovation Management**, v. 13, n. 4, p. 283 – 295, 1996. ISSN 0737-6782. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0737678296000331>.

KARLSSON, J.; RYAN, K. A cost-value approach for prioritizing requirements. **Software, IEEE**, v. 14, p. 67 – 74, 10 1997.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: **Proceedings of ICNN'95 - International Conference on Neural Networks**. [S.l.: s.n.], 1995. v. 4, p. 1942–1948 vol.4.

KRAUSE, J.; LOPES, H. A comparison of differential evolution algorithm with binary and continuous encoding for the mkp. In: . [S.l.: s.n.], 2013. p. 381–387. ISBN 978-1-4799-3194-1.

KUKREJA, N. et al. Value-based requirements prioritization: Usage experiences. **Procedia Computer Science**, v. 16, p. 806–813, 12 2013.

KUMARI, A. C.; SRINIVAS, K.; GUPTA, M. P. Software requirements optimization using multi-objective quantum-inspired hybrid differential evolution. In: SCHÜTZE, O. et al. (Ed.). **EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 107–120.

LEE, K. Y. **Modern Heuristic Optimization Techniques : Theory and Applications to Power Systems**. [S.l.: s.n.], 2008. ISBN 0470225858.

LEFFINGWELL, D.; WIDRIG, D. **Managing Software Requirements: A Unified Approach**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000. ISBN 0-201-61593-2.

LI, X. R. An improved quantum-behaved particle swarm optimization algorithm for the knapsack problem. **Applied Mechanics and Materials**, v. 373-375, p. 1178–1181, 08 2013.

LI, Z.; LI, N. A novel multi-mutation binary particle swarm optimization for 0/1 knapsack problem. In: **2009 Chinese Control and Decision Conference**. [S.l.: s.n.], 2009. p. 3042–3047. ISSN 1948-9439.

LIBAO, D. et al. A hybrid mutation scheme-based discrete differential evolution algorithm for multidimensional knapsack problem. In: . [S.l.: s.n.], 2016. p. 1009–1014.

LIBAO, D. et al. A hybrid mutation scheme-based discrete differential evolution algorithm for multidimensional knapsack problem. In: . [S.l.: s.n.], 2016. p. 1009–1014.

LIM, S. L.; FINKELSTEIN, A. Stakerare: Using social networks and collaborative filtering for large-scale requirements elicitation. **IEEE Transactions on Software Engineering**, v. 38, n. 3, p. 707–735, May 2012. ISSN 0098-5589.

OUYANG, L.; WANG, D. New particle swarm optimization algorithm for knapsack problem. 05 2012.

PENG, C.; JIAN, L.; ZHIMING, L. Solving 0-1 knapsack problems by a discrete binary version of differential evolution. **Intelligent Information Technology Applications, 2007 Workshop on**, v. 2, p. 513–516, 12 2008.

PENG, H. et al. Dichotomous binary differential evolution for knapsack problems. **Mathematical Problems in Engineering**, v. 2016, p. 1–12, 01 2016.

PERINI, A. et al. An empirical study to compare the accuracy of ahp and cbranking techniques for requirements prioritization. In: . [S.l.: s.n.], 2007. p. 23 – 35. ISBN 978-0-7695-3378-0.

PETERSEN, K. et al. Systematic mapping studies in software engineering. **Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering**, v. 17, 06 2008.

PITANGUEIRA, A. M.; MACIEL, R. S.; 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, 10 2014.

POPLI, R.; CHAUHAN, N.; SHARMA, H. Prioritising user stories in agile environment. In: . [S.l.: s.n.], 2014. p. 515–519. ISBN 978-1-4799-2900-9.

QASIM, S. Z.; ISMAIL, M. A. Research problems in search-based software engineering for many-objective optimization. In: **2017 International Conference on Innovations in Electrical Engineering and Computational Technologies (ICIEECT)**. [S.l.: s.n.], 2017. p. 1–6.

REYNOLDS, C. W. Flocks, herds and schools: A distributed behavioral model. In: **Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques**. New York, NY, USA: ACM, 1987. (SIGGRAPH '87), p. 25–34. ISBN 0-89791-227-6. Disponível em: <http://doi.acm.org/10.1145/37401.37406>.

RIEGEL, N.; DOERR, J. A systematic literature review of requirements prioritization criteria. In: . [S.l.: s.n.], 2015. p. 300–317.

RUHE, G. **Software Release Planning**. 1st. ed. New Jersey, USA: World Publishing Co, Inc., 2001.

RUHE, G.; SALIU, M. The art and science of software release planning. **Software, IEEE**, v. 22, p. 47– 53, 12 2005.

SAGRADO, J. del; ÁAGUILA, I. M.; ORELLANA, F. J. Requirements interaction in the next release problem. In: **Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation**. New York, NY, USA: ACM, 2011. (GECCO '11), p. 241–242. ISBN 978-1-4503-0690-4. Disponível em: <http://doi.acm.org/10.1145/2001858.2001994>.

SAGRADO, J. del; ÁGUILA, I.; ORELLANA, F. Multi-objective ant colony optimization for requirements selection. **Empirical Software Engineering**, v. 20, p. 577–610, 06 2015.

SANTOS, R.; ALBUQUERQUE, A.; PINHEIRO, P. Towards the applied hybrid model in requirements prioritization. **Procedia Computer Science**, v. 91, p. 909–918, 12 2016.

SAWYER, P.; SOMMERVILLE, I.; VILLER, S. Capturing the benefits of requirements engineering. **IEEE Softw.**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 16, n. 2, p. 78–85, mar. 1999. ISSN 0740-7459. Disponível em: <http://dx.doi.org/10.1109/52.754057>.

SHAO, F. et al. Drank: A semi-automated requirements prioritization method based on preferences and dependencies. **Journal of Systems and Software**, v. 126, 09 2016.

SHER, F. et al. Requirements prioritization techniques and different aspects for prioritization a systematic literature review protocol. In: . [S.l.: s.n.], 2014.

SOUZA, J. T. de et al. An ant colony optimization approach to the software release planning with dependent requirements. In: **Proceedings of the Third International Conference on Search Based Software Engineering**. Berlin, Heidelberg: Springer-Verlag, 2011. (SSBSE'11), p. 142–157. ISBN 978-3-642-23715-7. Disponível em: <http://dl.acm.org/citation.cfm?id=2042243.2042264>.

STORN, R.; PRICE, K. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. **Journal of Global Optimization**, v. 11, p. 341–359, 01 1997.

WAGDY, A.; ALMAZYAD, A. Differential evolution with novel mutation and adaptive crossover strategies for solving large scale global optimization problems. **Applied Computational Intelligence and Soft Computing**, v. 2017, p. 1–18, 03 2017.

WANG, L.; WANG, X. ting; FEI, M. An adaptive mutation-dissipation binary particle swarm optimisation for multidimensional knapsack problem. **International Journal of Modelling, Identification and Control**, v. 8, 12 2009.

WANG, Y. et al. A novel quantum swarm evolutionary algorithm for solving 0-1 knapsack problem. In: . [S.l.: s.n.], 2005. v. 3611, p. 433–433.

XIANG, W.-L. et al. An enhanced differential evolution algorithm based on multiple mutation strategies. **Computational Intelligence and Neuroscience**, v. 2015, p. 1–15, 11 2015.

XIE, W. et al. Vulnerability detection in iot firmware: A survey. In: **2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)**. [S.l.: s.n.], 2017. p. 769–772. ISSN 1521-9097.

XUAN, J. et al. Solving the large scale next release problem with a backbone-based multilevel algorithm. **IEEE Transactions on Software Engineering**, v. 38, p. 1195–1212, 09 2012.

YOUSUF, M.; BOKHARI, M. U.; ZEYAUDDIN, M. An analysis of software requirements prioritization techniques: A detailed survey. In: **2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)**. [S.l.: s.n.], 2016. p. 3966–3970.

ZHANG, Y.; FINKELSTEIN, A.; HARMAN, M. Search based requirements optimisation: Existing work and challenges. In: . [S.l.: s.n.], 2008. v. 5025, p. 88–94.

ZHANG, Y.; HARMAN, M.; MANSOURI, S. A. The multi-objective next release problem. In: **Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation**. New York, NY, USA: ACM, 2007. (GECCO '07), p. 1129–1137. ISBN 978-1-59593-697-4. Disponível em: <http://doi.acm.org/10.1145/1276958.1277179>.

ZOUACHE, D.; MOUSSAOUI, A. Quantum-inspired differential evolution with particle swarm optimization for knapsack problem. **Journal of Information Science and Engineering**, v. 31, p. 1779–1795, 09 2015.

ZOUACHE, D.; MOUSSAOUI, A.; ABDELAZIZ, F. A cooperative swarm intelligence algorithm for multi-objective discrete optimization with application to the knapsack problem. **European Journal of Operational Research**, 07 2017.

APÊNDICE A - MAPEAMENTO SISTEMÁTICO DE LITERATURA

Esta seção descreve o método utilizado para a realização dos mapeamentos sistemáticos de literatura. Para ter um melhor entendimento da área de estudo escolhida, bem como do estado da arte e das lacunas de pesquisa, os temas escolhidos foram: seleção de requisitos utilizando métodos computacionais e o problema da mochila. A primeira por ser o tema de estudo e a segunda, porque o problema de seleção de requisitos é modelado como um problema de mochila, então decidiu-se que também seria muito importante entender o estado da arte dessa outra área de pesquisa.

Um mapeamento sistemático de literatura é uma técnica utilizada para identificar, avaliar e interpretar pesquisas em uma área de interesse, responder a perguntas de pesquisa ou um fenômeno de interesse (KITCHENHAM, 2004). Em outras palavras, essa abordagem utiliza métodos explícitos e rigorosos para identificar, avaliar criticamente e sintetizar estudos relevantes em uma área particular de pesquisa.

Os mapeamentos foram realizadas seguindo a metodologia proposta por Petersen et al., (2008), e as etapas são mostradas na Figura 1.

Figura A1: Processo de Mapeamento de Literatura

Fonte: Adaptado de Petersen et al. (2008)

As etapas essenciais desse processo são:

- Definição das questões de pesquisa
- Busca dos artigos relacionados
- Extração dos dados

A1.1 QUESTÕES DE PESQUISA

Considerando os objetivos propostos, as seguintes perguntas de pesquisa foram elaboradas, para o primeiro mapeamento sistemático:

[RQ1.1] Quais são os modelos de seleção de requisitos mais explorados (satisfação parcial ou total)? A ideia é levantar em que tipo de situação cada tipo de modelagem tem sido utilizada e em que contexto, dessa forma será possível determinar se, por exemplo, um tipo de modelagem é mais adequado para o uso em um ambiente de desenvolvimento ágil ou convencional.

[RQ1.2] Quais famílias de algoritmos são as mais utilizadas na seleção de requisitos? O objetivo dessa pergunta é fazer um levantamento de todos os algoritmos aplicados na seleção de requisitos, agrupá-los por famílias e determinar quais famílias são as mais utilizadas e com qual nível de sucesso.

[RQ1.3] Como as interdependências entre requisitos são tratadas nos modelos? Em um projeto real, interdependências entre requisitos são comuns, e essa pergunta visa determinar se elas são utilizadas ou não, quais os tipos considerados e os mais comuns.

Para o segundo mapeamento, as questões de pesquisa foram as seguintes:

[RQ2.1] Quais famílias de algoritmos são mais utilizadas no problema de mochila? Essa pergunta tem o mesmo objetivo da RQ1.1 porém voltada para o problema da mochila.

[RQ2.2] Existem variações do problema de mochila que podem ser utilizadas no problema de seleção de requisitos? O problema de seleção de requisitos é modelado como uma variação do problema de mochila, sendo esse próprio problema uma variação do problema original de mochila. A ideia seria determinar que outras variações existem, e se elas poderiam ser utilizadas na seleção de requisitos.

[RQ2.3] Como a área de estudo sobre esse problema tem evoluído? O problema em questão é uma aplicação do problema de mochila, que possui uma área de estudo própria, visto que possui inúmeras outras aplicações práticas. Portanto, é de se esperar que a maioria das

novidades apareçam em trabalhos focados no problema básico e não em aplicações, por isso essa pergunta visa determinar se essa área de pesquisa continua em evolução, baseado na quantidade de trabalhos sendo publicados.

A1.2 BUSCA PRELIMINAR DE TRABALHOS

O levantamento inicial de trabalhos relevantes para os temas sendo pesquisados deve ser feito em bases de dados científicas, por meio de pesquisas por palavras-chave. Os termos a serem pesquisados foram definidos com base nas áreas de conhecimento envolvidas nas questões de pesquisa, e podem ser vistos na Tabela A1, para o mapeamento sobre a seleção de requisitos. O Grupo A possui as palavras-chave relacionadas ao tema "seleção de requisitos" e o Grupo B as palavras chave relacionadas ao tema "algoritmos de otimização".

Tabela A1 - Palavras-chave utilizadas na pesquisa

Grupo A	Grupo B
next release problem	search based
requirements	multi-objective optimization
release planning	genetic algorithm
software release	genetic programming
	Heuristic
	simulated annealing
	ant colony optimization
	particle swarm optimization
	artificial bee colony
	differential evolution
	evolutionary algorithm

Fonte: O Autor.

A construção dos termos de pesquisa e a notação lógica para a busca foram formados da seguinte maneira:

(A1 OU A2 ... OU An) E (B1 OU B2 ... OU Bn)

Na qual Ai e Bj são os termos dos Grupos A e B respectivamente e os termos E e OU são operadores booleanos.

Para levantar o estado da arte sobre o problema de mochila, foi utilizado apenas o termo *knapsack*, de modo a fazer uma análise básica de todos os trabalhos relacionados a esse tema. Não foram incluídos outros termos, pois nesse caso é seguro assumir que se o trabalho não tem o termo *knapsack* no título ou no *abstract*, então ele não está relacionado ao assunto.

Bases de dados eletrônicas na área de ciência da computação e tecnologia em geral foram consultadas. A quantidade de resultados encontrados em cada base é apresentada na Tabela A2.

Tabela A2 – Bases acadêmicas utilizadas na pesquisa

Base Acadêmica	Endereço Eletrônico	Priorização de Requisitos	Problema de Mochila 0/1
ACM Digital Library	http://dl.acm.org	39	977
IEEE Xplore	http://ieeexplore.ieee.org	79	1164
SBSE Repository	http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/repository.html	66	0
Science Direct	http://www.sciencedirect.com	47	7905
Scopus	http://www.scopus.com.br	51	5316
Web of Science	http://www.isiknowledge.com	39	4548

Fonte: O autor

A1.3 PROCESSO DE SELEÇÃO DOS TRABALHOS

Após definidas as palavras chave, e a busca inicial ter sido realizada nas bases de dados, é necessário definir e aplicar um processo de seleção dos trabalhos, de modo a eliminar os que não são relevantes.

A1.3.1 Seleção dos Trabalhos sobre Seleção de Requisitos

O fluxo apresentado na Figura A2 mostra o processo de seleção aplicado nos trabalhos sobre seleção de requisitos. Inicialmente foram obtidos 332 trabalhos, que numa triagem inicial, baseada no título, possuíam alguma relação com o assunto.

Essa triagem inicial visou remover trabalhos que não tinham relação nenhuma com o assunto, em caso de dúvida sobre o artigo era mantido para uma análise detalhada posteriormente. Em seguida foram removidos os itens duplicados, resultando em 75 trabalhos.

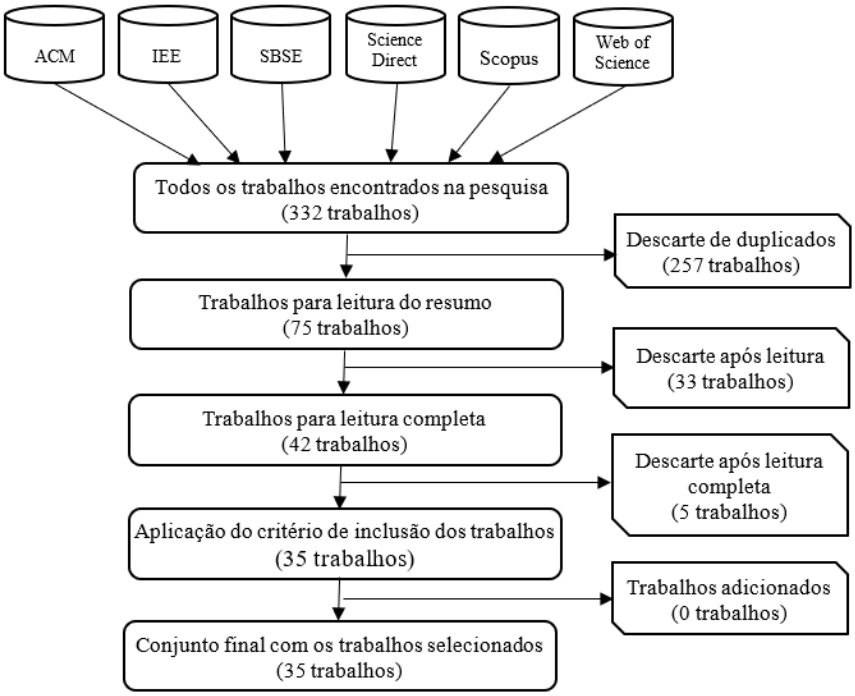
Uma leitura do resumo desses trabalhos foi realizada, com o intuído de remover trabalhos que não estavam ligados à proposta do presente trabalho, resultando em 42 trabalhos para leitura e análise completa. Essa análise resultou na remoção de mais 5 trabalhos, que se tratavam de resumo ou revisões de literatura.

A etapa final foi adicionar trabalhos que possam não ter sido encontrados na pesquisa inicial das bases de dados. Isso foi feito observando as referências bibliográficas utilizadas em outros trabalhos, principalmente as revisões de literatura. Nenhum novo trabalho foi encontrado durante esse processo.

Ao final desse processo restaram 35 artigos, que foram usados para responder as questões de pesquisa RQ1.1, RQ1.2 e RQ1.3. Esses trabalhos foram categorizados nos seguintes grupos:

- **Ano:** ano em que o trabalho foi publicado.
- **Métodos:** algoritmos e técnicas utilizadas para resolver o problema.
- **Modelagem:** tipo de modelagem utilizada no problema, simples ou multi-objetivo.
- **Datasets:** origem dos datasets utilizados para testar as soluções propostas.
- Interdependências: como foram consideradas as interdependências entre os requisitos, se foi considerada apenas a interdependência de precedência ou todas as levantadas por Carlshamre (2001).
- **Tipo de modelo matemático:** Se o modelo utilizado foi o de satisfação parcial ou total, conforme explicado na seção 2.5.

Figura A2: Processo de revisão sobre seleção de requisitos



Fonte: O Autor.

A1.3.2 Seleção dos Trabalhos sobre o Problema de Mochila

Devido ao grande número de trabalhos resultantes nas buscas sobre o problema de mochila, foi utilizada uma ferramenta computacional, desenvolvida pelo autor, para fazer a triagem e seleção desses trabalhos. Na pesquisa inicial foram encontrados 19910 trabalhos ao todo. A fase consistiu em eliminar os artigos duplicados entre as bases, isso foi feito observando o título e o número DOI do trabalho, caso existisse, resultando em 12218 trabalhos. Em seguida foram separados os trabalhos que não possuem o termo *knapsack* no título ou nas palavras-chave. Isso foi feito porque existe uma boa chance desses artigos não terem relação alguma ao problema de mochila. Portanto uma análise básica apenas pelo título pode descartar os trabalhos de outras áreas ou temas, resultando na remoção de 9498 artigos. Desses artigos a grande maioria são aplicações do problema de mochila em outras áreas como medicina, segurança, meio ambiente e criptografia.

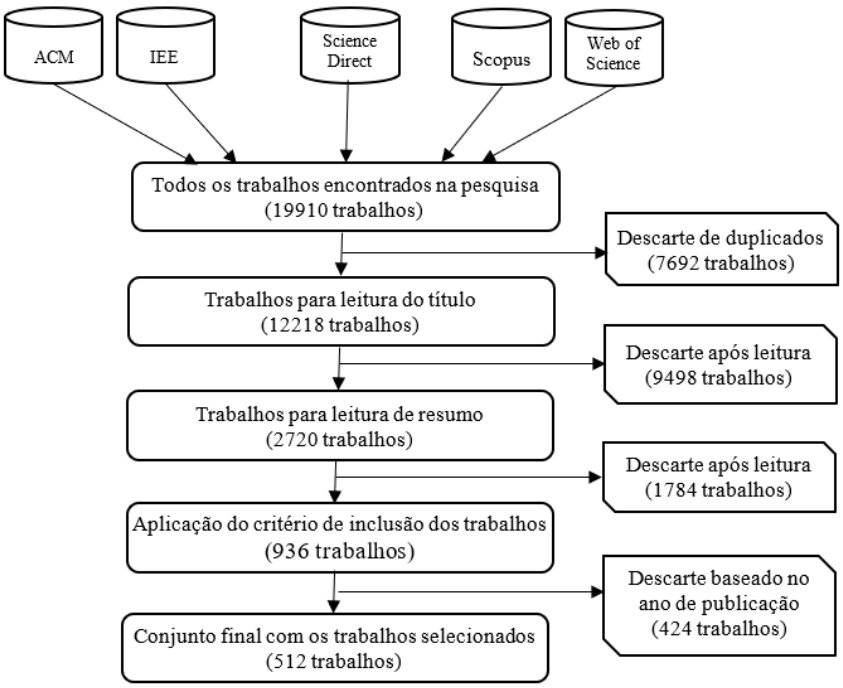
Os 2720 trabalhos restantes estão relacionados diretamente à solução do problema de mochila, mas ainda existem certos tipos de estudos que não são interessantes para aplicação no problema de seleção de requisitos, portanto foram definidos novos critérios de descarte: 1)

estudos focados em computação paralela, pois são específicos para aplicações em que o cálculo da função objetivo demanda muito tempo de processamento, o que não é o caso na seleção de requisitos; 2) soluções baseadas em técnicas heurísticas, pois assumem um conhecimento prévio do problema para gerar uma melhor solução, e portanto são específicas para os problemas em que foram aplicadas; 3) soluções que utilizam computação baseada em DNA; 4) problemas de mochila quadráticos, pois são uma subclasse do problema de mochila que permite termos quadráticos na função objetivo, que não é o caso desse estudo, conforme pode ser visto nas Equações 1 a 8; 5) Outras classes de problemas não lineares ou que permitem a seleção de mais de um item do mesmo tipo, e; 6) Aplicações do problema de mochila que não foram detectadas nas fases anteriores. Ao final dessa fase restaram 936 trabalhos.

Observando essa população menor de trabalhos, fica claro que existe uma diferença entre os trabalhos publicados antes e depois da década de 90. Até meados da década de 90, os trabalhos utilizavam em sua maioria aproximações e simplificações para tentar chegar a alguma solução exata e métodos computacionais mais simples como divisão e conquista ou algoritmos gulosos. Quando os algoritmos genéticos começaram a se tornar mais populares essa área de pesquisa mudou completamente seu foco em pouco tempo. No início dos anos 2000 os métodos meta-heurísticos já eram bem populares na solução do problema de mochila, portanto decidiu-se focar nessa fase, que também foi quando surgiu o problema de seleção de requisitos foi modelado como um problema de otimização e busca. Isso resultou no descarte de mais 411 trabalhos, reduzindo o número de trabalhos para 512, que serão usados para responder as questões de pesquisa RQ2.1, RQ2.2 e RQ2.3. A Figura A3 mostra esse processo de seleção. Os artigos que chegaram até o final do processo foram categorizados utilizando os seguintes grupos:

- **Ano:** ano em que o trabalho foi publicado.
- **Tipo:** subclasse do problema de mochila considerado no trabalho.
- **Métodos:** método utilizado para solucionar o problema apresentado no trabalho.
- **Algoritmo:** no caso de métodos bio-inspirados, essa coluna apresenta o tipo de algoritmo utilizado no trabalho. Para outros métodos, essa coluna não possui nenhuma informação.
- **Modelagem:** Tipo de modelagem, simples ou multi-objetivo.

Figura A3: Processo de revisão sobre o problema de mochila

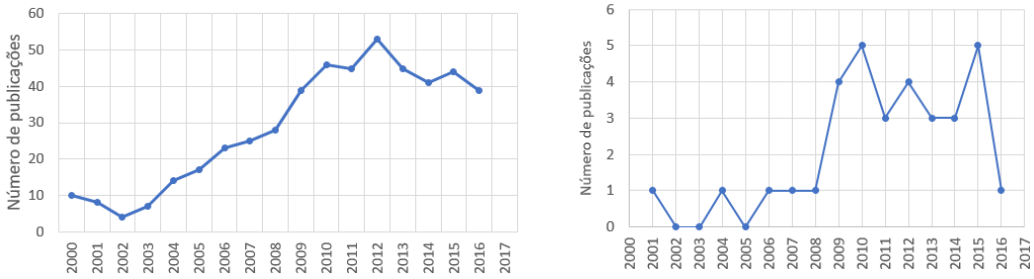


Fonte: O Autor.

A1.4 RESULTADOS

O problema de mochila tem sido largamente estudado, e a partir de 2002 observou-se um grande aumento no número de trabalhos publicados sobre o assunto, até 2010, e permanecendo num patamar relativamente estável até 2016, como pode ser visto na Figura 4, no gráfico da esquerda (os trabalhos de 2017 foram removidos desse gráfico, visto que outros trabalhos ainda serão publicados no restante do ano). Um comportamento bem parecido é observado no número de publicações anuais sobre a seleção de requisitos, na Figura 4, no gráfico da direita. Tais dados respondem à pergunta RQ2.3, mostrando que ambas as áreas de estudo têm evoluído de forma similar.

Figura A4: Número de publicações anuais



Fonte: O Autor.

Com relação aos tipos de técnicas utilizadas, os algoritmos bio-inspirados são muito mais utilizados do que qualquer outra técnica, representando 59,6% e 63,4% nos problemas de mochila e requisitos respectivamente. As outras técnicas levantadas podem ser vistas na Tabela A3 e na Tabela A4. Nas duas tabelas o grupo "Outros" engloba métodos que foram utilizados menos de duas vezes na população de trabalhos analisados.

Tabela A3: Técnicas utilizadas no problema de mochila

Técnicas	Utilizações
Algoritmos Bio-Inspirados	305
Programação Dinâmica	25
Métodos Exatos	18
Aproximações	17
Divisão e Conquista	15
Aprox. de Tempo Polinomial	15
Algoritmos Gulosos	12
Busca Local	8
Busca Tabu	8
Programação Inteira	7
Modelo Baseado em Agente	6
Busca Dispersa	6
Fuzzy	5
Busca de Vizinhança	4
Soma dos Subconjuntos	4
Programação Linear	3
Outros	54

Fonte: O Autor.

Tabela A4: Técnicas utilizadas no problema de seleção de requisitos

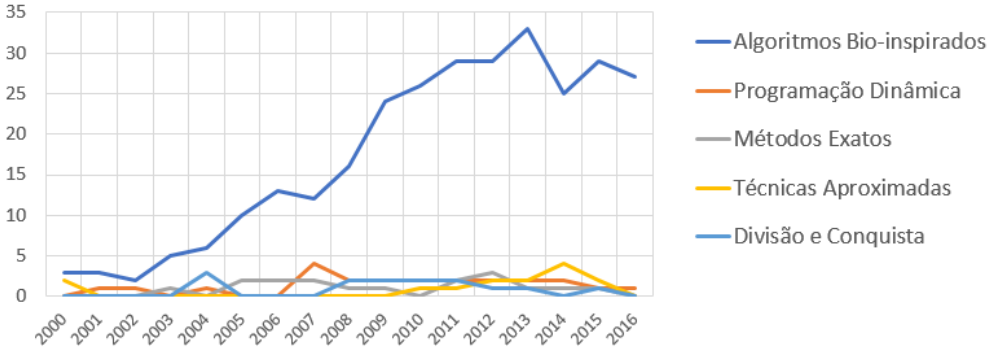
Técnicas	Utilizações
Algoritmos Bio-Inspirados	64
Simulated Annealing	13
Algoritmos Gulosos	9
Busca Aleatória	7
Hill Climbing	4
Outros	4

Fonte: O Autor.

A Figura A1 mostra a evolução das principais técnicas utilizadas para resolver o problema de mochila, e já era de se esperar que os algoritmos bio-inspirados tivessem uma curva de evolução muito parecida com a da Figura A4 (Problema da Mochila), já que representam a população de maior número. Porém, nota-se que as outras técnicas não tiveram

nenhuma evolução no período estudado, se mantendo estáveis e com um número de publicações muito inferior ao dos algoritmos bio-inspirados.

Figura A1: Publicações com as técnicas mais utilizadas no problema de seleção de requisites



Fonte: O Autor

Com isto constatou-se que, tanto para as perguntas RQ1.2 quanto para a RQ2.1, algoritmos bio-inspirados são os mais utilizados nos trabalhos publicados. Como esse grupo de algoritmos é composto por um grande número de técnicas, é importante levantar quais dessas técnicas estão sendo mais utilizadas. A Tabela A5 e a Tabela A6 mostram os algoritmos mais utilizados e o grupo "Outros" representa métodos que foram utilizados 2 ou menos vezes na literatura considerada. Mais uma vez se observa que as duas áreas estão relativamente sintonizadas, com algoritmos genéticos representando a maior parte das aplicações, sendo usados em 26,2% dos trabalhos de mochila e em 36,0% de seleção de requisitos. Porém observa-se que para o problema de mochila, em segundo lugar e representando 14,4% dos trabalhos, está o algoritmo de enxame de partículas, que só foi explorado uma única vez para seleção de requisitos.

Algoritmos Evolucionários	Utilizações
Algoritmos Genéticos	80
Enxame de Partículas	44
Colônia de Formigas	40
Baseado em População	34
Evolução Diferencial	14
Busca Harnônica	13
Abelha Artificial	12
Sistemas Imunológicos	12
Cuco	7
Meméticos	5
Membrana	3
Alcatéia de Lobos	3
Ameba	2
Cardume de Peixes	2
Morcego	3
Vagalume	2
Frog Leaping	2
Fruit Fly	2
Monarch Butterfly	2
Outros	23

Tabela 5: Algoritmos evolucionários utilizados no problema de mochila

Algoritmos Evolucionários	Utilizações
Algoritmos Genéticos	9
Colônia de Formigas	7
Baseado em População	5
Evolução Diferencial	2
Enxame de Partículas	1
Abelha Artificial	1

Tabela 6: Algoritmos evolucionários utilizados no problema de seleção de requisitos

Da literatura considerada, foram identificadas várias variações do problema de mochila, sendo que 54,6% dos trabalhos focam na formulação básica do problema, e 29,5% focam na variação chamada de *Multidimensional Knapsack Problem* (MKP) (ver Tabela A7).

Tabela A7: Variações do problema de mochila

Variação	Trabalhos
Knapsack Problem	279
Mutidimensional Knapsack Problem	151
Multiple-Choice Knapsack Problem	22
Constrained Knapsack Problem	17
Unbounded Knapsack Problem	12
Nonlinear Knapsack Problem	9
Min-Max Knapsack Problem	7
Precedence Constrained Knapsack Problem	4
Dynamic Knapsack Problem	3
Parametric Knapsack Problem	2
Bounded Knapsack Problem	2
Collapsing Knapsack Problem	1
Fixed-Charge Knapsack Problem	1
Setup Knapsack Problem	1
Setup Knapsack Problem	1

Fonte: O Autor

Nos trabalhos de seleção de requisitos, 26% não cita como a interdependência entre os requisitos foi tratada na solução do problema e 20% não considera interdependência alguma. Dos 54% que consideram, 53% utiliza apenas a interdependência de precedência e 47% consideram as interdependências de forma completa, conforme proposto por Sandhal (2001). Esses dados são suficientes para responder a RQ1.3, mostrando que quase metade dos trabalhos não levou em conta a interdependência entre os requisitos, o que é uma grande deficiência, visto que um projeto real de software raramente terá todos os requisitos independentes e fazendo com que esses trabalhos tenham pouca validade do ponto de vista prático.

Para responder à pergunta RQ1.1, foram analisadas as formulações matemáticas do problema nos trabalhos selecionados para determinar qual o tipo de modelagem foi adotada. Na seção 2.5 foram apresentadas as duas formulações mais comuns, denominadas de Modelos de Satisfação Parcial e Total. A maior parte, ou 67,6% dos trabalhos, utiliza o Modelo de Satisfação Total, e 17,6% utiliza o outro modelo. Os 14,7% restantes utiliza outras abordagens, como propor modelos próprios ou fazer a seleção dos requisitos automatizando métodos não computacionais.

APÊNDICE B - RESULTADOS DETALHADOS DE CADA ALGORITMO

Tabela B1 – Resultados de Evolução Diferencial para o projeto CBS

Estratégia	População	Crossover	Peso Diferencial	Gerações	Mínimo	Média	Máximo
DE/Rand/1	50	0,15	0,15	100	1733,0	1733,0	1733,0
DE/Best/1	50	0,15	0,15	50	1728,0	1732,5	1733,0
DE/Target-to-Best/1	50	0,15	0,15	150	1728,0	1732,0	1733,0
DE/Best/2	50	0,15	0,15	50	1733,0	1733,0	1733,0
DE/Rand/2	50	0,15	0,15	100	1733,0	1733,0	1733,0
DE/Rand/1	50	0,15	0,50	100	1733,0	1733,0	1733,0
DE/Best/1	50	0,15	0,50	100	1733,0	1733,0	1733,0
DE/Target-to-Best/1	50	0,15	0,50	100	1728,0	1732,3	1733,0
DE/Best/2	50	0,15	0,50	50	1733,0	1733,0	1733,0
DE/Rand/2	50	0,15	0,50	100	1733,0	1733,0	1733,0
DE/Rand/1	50	0,15	0,85	100	1733,0	1733,0	1733,0
DE/Best/1	50	0,15	0,85	100	1733,0	1733,0	1733,0
DE/Target-to-Best/1	50	0,15	0,85	100	1733,0	1733,0	1733,0
DE/Best/2	50	0,15	0,85	100	1733,0	1733,0	1733,0
DE/Rand/2	50	0,15	0,85	100	1733,0	1733,0	1733,0
DE/Rand/1	50	0,50	0,15	50	1733,0	1733,0	1733,0
DE/Best/1	50	0,50	0,15	25	1700,0	1722,4	1733,0
DE/Target-to-Best/1	50	0,50	0,15	200	1728,0	1729,5	1733,0
DE/Best/2	50	0,50	0,15	25	1719,0	1730,0	1733,0
DE/Rand/2	50	0,50	0,15	50	1733,0	1733,0	1733,0
DE/Rand/1	50	0,50	0,50	100	1733,0	1733,0	1733,0
DE/Best/1	50	0,50	0,50	25	1728,0	1732,5	1733,0
DE/Target-to-Best/1	50	0,50	0,50	50	1733,0	1733,0	1733,0
DE/Best/2	50	0,50	0,50	50	1728,0	1732,5	1733,0
DE/Rand/2	50	0,50	0,50	100	1733,0	1733,0	1733,0
DE/Rand/1	50	0,50	0,85	100	1733,0	1733,0	1733,0
DE/Best/1	50	0,50	0,85	50	1733,0	1733,0	1733,0
DE/Target-to-Best/1	50	0,50	0,85	100	1733,0	1733,0	1733,0
DE/Best/2	50	0,50	0,85	100	1733,0	1733,0	1733,0
DE/Rand/2	50	0,50	0,85	100	1733,0	1733,0	1733,0
DE/Rand/1	50	0,85	0,15	25	1728,0	1731,5	1733,0
DE/Best/1	50	0,85	0,15	10	1641,0	1706,7	1733,0
DE/Target-to-Best/1	50	0,85	0,15	50	1719,0	1730,1	1733,0
DE/Best/2	50	0,85	0,15	15	1658,0	1718,0	1733,0
DE/Rand/2	50	0,85	0,15	50	1728,0	1732,0	1733,0
DE/Rand/1	50	0,85	0,50	100	1733,0	1733,0	1733,0
DE/Best/1	50	0,85	0,50	25	1728,0	1732,0	1733,0
DE/Target-to-Best/1	50	0,85	0,50	50	1728,0	1731,5	1733,0
DE/Best/2	50	0,85	0,50	25	1704,0	1726,2	1733,0
DE/Rand/2	50	0,85	0,50	100	1733,0	1733,0	1733,0
DE/Rand/1	50	0,85	0,85	100	1733,0	1733,0	1733,0
DE/Best/1	50	0,85	0,85	100	1733,0	1733,0	1733,0
DE/Target-to-Best/1	50	0,85	0,85	100	1733,0	1733,0	1733,0
DE/Best/2	50	0,85	0,85	100	1733,0	1733,0	1733,0
DE/Rand/2	50	0,85	0,85	100	1733,0	1733,0	1733,0

Tabela B2 – Resultados do algoritmo PSO para o projeto CBS.

Partículas	Peso Inercial	Fator Cognitivo	Fator Social	Prob. De Morte	Gerações	Mínimo	Média	Máximo
5	0,25	1,50	1,50	0,01	50	1617,0	1684,0	1733,0
10	0,25	1,50	1,50	0,01	50	1685,0	1723,4	1733,0
15	0,25	1,50	1,50	0,01	50	1700,0	1724,9	1733,0
25	0,25	1,50	1,50	0,01	50	1691,0	1725,5	1733,0
50	0,25	1,50	1,50	0,01	50	1719,0	1729,1	1733,0
5	0,75	1,50	1,50	0,01	50	1684,0	1715,4	1733,0
10	0,75	1,50	1,50	0,01	50	1728,0	1731,0	1733,0
15	0,75	1,50	1,50	0,01	50	1709,0	1727,3	1733,0
25	0,75	1,50	1,50	0,01	50	1733,0	1733,0	1733,0
50	0,75	1,50	1,50	0,01	50	1733,0	1733,0	1733,0
5	0,5	2,00	1,00	0,01	50	1612,0	1698,3	1733,0
10	0,5	2,00	1,00	0,01	50	1719,0	1729,1	1733,0
15	0,5	2,00	1,00	0,01	50	1728,0	1731,0	1733,0
25	0,5	2,00	1,00	0,01	50	1728,0	1730,5	1733,0
50	0,5	2,00	1,00	0,01	50	1728,0	1732,0	1733,0
5	0,5	1,00	2,00	0,01	50	1638,0	1697,9	1733,0
10	0,5	1,00	2,00	0,01	50	1688,0	1717,8	1733,0
15	0,5	1,00	2,00	0,01	50	1719,0	1730,1	1733,0
25	0,5	1,00	2,00	0,01	50	1728,0	1731,5	1733,0
50	0,5	1,00	2,00	0,01	50	1728,0	1730,5	1733,0
5	0,25	2,00	2,00	0,01	50	1684,0	1715,7	1733,0
10	0,25	2,00	2,00	0,01	50	1709,0	1728,2	1733,0
15	0,25	2,00	2,00	0,01	50	1704,0	1728,1	1733,0
25	0,25	2,00	2,00	0,01	50	1728,0	1732,0	1733,0
50	0,25	2,00	2,00	0,01	50	1728,0	1732,5	1733,0
5	0,75	2,00	2,00	0,01	50	1703,0	1719,3	1733,0
10	0,75	2,00	2,00	0,01	50	1719,0	1729,7	1733,0
15	0,75	2,00	2,00	0,01	50	1719,0	1729,6	1733,0
25	0,75	2,00	2,00	0,01	50	1728,0	1732,5	1733,0
50	0,75	2,00	2,00	0,01	50	1733,0	1733,0	1733,0

Tabela B3 – Resultados obtidos com DE para o projeto FPG

Estratégia	População	Crossover	Peso Diferencial	Gerações	Mínimo	Média	Máximo
DE/Rand/1	50	0,15	0,15	200	4185,0	4185,0	4185,0
DE/Best/1	50	0,15	0,15	200	4151,0	4169,0	4182,0
DE/Target-to-Best/1	50	0,15	0,15	300	4126,0	4157,8	4180,0
DE/Best/2	50	0,15	0,15	500	4152,0	4165,4	4178,0
DE/Rand/2	50	0,15	0,15	500	4185,0	4185,0	4185,0
DE/Rand/1	50	0,15	0,50	300	4183,0	4184,7	4185,0
DE/Best/1	50	0,15	0,50	200	4184,0	4184,8	4185,0
DE/Target-to-Best/1	50	0,15	0,50	200	4175,0	4183,9	4185,0
DE/Best/2	50	0,15	0,50	200	4184,0	4184,5	4185,0
DE/Rand/2	50	0,15	0,50	200	4182,0	4184,7	4185,0
DE/Rand/1	50	0,15	0,85	500	4183,0	4183,9	4185,0
DE/Best/1	50	0,15	0,85	500	4183,0	4184,8	4185,0
DE/Target-to-Best/1	50	0,15	0,85	500	4183,0	4184,5	4185,0
DE/Best/2	50	0,15	0,85	200	4182,0	4184,7	4185,0
DE/Rand/2	50	0,15	0,85	300	4170,0	4178,8	4184,0
DE/Rand/1	50	0,50	0,15	100	4167,0	4180,1	4185,0
DE/Best/1	50	0,50	0,15	50	4069,0	4144,1	4185,0
DE/Target-to-Best/1	50	0,50	0,15	100	4090,0	4130,4	4159,0
DE/Best/2	50	0,50	0,15	50	4037,0	4107,4	4160,0
DE/Rand/2	50	0,50	0,15	100	4175,0	4182,5	4185,0
DE/Rand/1	50	0,50	0,50	300	4167,0	4179,3	4185,0
DE/Best/1	50	0,50	0,50	100	4151,0	4167,5	4180,0
DE/Target-to-Best/1	50	0,50	0,50	100	4161,0	4179,0	4185,0
DE/Best/2	50	0,50	0,50	100	4138,0	4160,0	4177,0
DE/Rand/2	50	0,50	0,50	400	4138,0	4171,1	4185,0
DE/Rand/1	50	0,50	0,85	400	4122,0	4138,9	4153,0
DE/Best/1	50	0,50	0,85	300	4122,0	4161,1	4185,0
DE/Target-to-Best/1	50	0,50	0,85	200	4176,0	4180,5	4185,0
DE/Best/2	50	0,50	0,85	300	4178,0	4182,6	4185,0
DE/Rand/2	50	0,50	0,85	500	4116,0	4137,9	4165,0
DE/Rand/1	50	0,85	0,15	100	4162,0	4172,0	4185,0
DE/Best/1	50	0,85	0,15	50	3931,0	3996,2	4100,0
DE/Target-to-Best/1	50	0,85	0,15	100	4060,0	4093,7	4123,0
DE/Best/2	50	0,85	0,15	25	3962,0	4017,3	4080,0
DE/Rand/2	50	0,85	0,15	100	4163,0	4163,0	4163,0
DE/Rand/1	50	0,85	0,50	300	4175,0	4180,7	4185,0
DE/Best/1	50	0,85	0,50	100	4049,0	4106,3	4149,0
DE/Target-to-Best/1	50	0,85	0,50	50	4084,0	4139,6	4172,0
DE/Best/2	50	0,85	0,50	50	4063,0	4114,1	4160,0
DE/Rand/2	50	0,85	0,50	400	4172,0	4179,3	4185,0
DE/Rand/1	50	0,85	0,85	300	4070,0	7086,5	4114,0
DE/Best/1	50	0,85	0,85	200	4168,0	4180,3	4185,0
DE/Target-to-Best/1	50	0,85	0,85	300	4170,0	4180,1	4185,0
DE/Best/2	50	0,85	0,85	300	4175,0	4180,9	4185,0
DE/Rand/2	50	0,85	0,85	400	4078,0	4094,3	4108,0

Tabela B4 – Resultados do algoritmo PSO para o projeto FPG.

Partículas	Peso Inercial	Fator Cognitivo	Fator Social	Prob. De Morte	Gerações	Mínimo	Média	Máximo
5	0,25	1,50	1,50	0,01	500,0	3968,0	4055,6	4109,0
10	0,25	1,50	1,50	0,01	500,0	4053,0	4090,1	4149,0
15	0,25	1,50	1,50	0,01	500,0	4075,0	4116,9	4159,0
25	0,25	1,50	1,50	0,01	500,0	4106,0	4137,9	4170,0
50	0,25	1,50	1,50	0,01	500,0	4117,0	4123,7	4154,0
5	0,75	1,50	1,50	0,01	500,0	4045,0	4098,5	4136,0
10	0,75	1,50	1,50	0,01	500,0	4082,0	4131,7	4157,0
15	0,75	1,50	1,50	0,01	500,0	4109,0	4137,7	4162,0
25	0,75	1,50	1,50	0,01	500,0	4091,0	4144,5	4181,0
50	0,75	1,50	1,50	0,01	500,0	4097,0	4141,4	4182,0
5	0,5	2,00	1,00	0,01	500,0	4059,0	4119,2	4156,0
10	0,5	2,00	1,00	0,01	500,0	4095,0	4141,7	4171,0
15	0,5	2,00	1,00	0,01	500,0	4121,0	4154,8	4185,0
25	0,5	2,00	1,00	0,01	500,0	4098,0	4152,0	4175,0
50	0,5	2,00	1,00	0,01	500,0	4127,0	4157,4	4179,0
5	0,5	1,00	2,00	0,01	500,0	4029,0	4104,1	4160,0
10	0,5	1,00	2,00	0,01	500,0	4080,0	4118,7	4161,0
15	0,5	1,00	2,00	0,01	500,0	4057,0	4126,4	4151,0
25	0,5	1,00	2,00	0,01	500,0	4105,0	4134,4	4158,0
50	0,5	1,00	2,00	0,01	500,0	4125,0	4156,0	4168,0
5	0,25	2,00	2,00	0,01	500,0	4106,0	4142,6	4175,0
10	0,25	2,00	2,00	0,01	500,0	4130,0	4159,7	4182,0
15	0,25	2,00	2,00	0,01	500,0	4154,0	4168,3	4185,0
25	0,25	2,00	2,00	0,01	500,0	4147,0	4174,0	4185,0
50	0,25	2,00	2,00	0,01	500,0	4170,0	4178,8	4185,0
5	0,75	2,00	2,00	0,01	500,0	3981,0	4033,0	4089,0
10	0,75	2,00	2,00	0,01	500,0	4029,0	4057,3	4092,0
15	0,75	2,00	2,00	0,01	500,0	4006,0	4046,9	4103,0
25	0,75	2,00	2,00	0,01	500,0	4028,0	4061,8	4109,0
50	0,75	2,00	2,00	0,01	500,0	4035,0	4067,5	4106,0

Tabela B5 – Resultados obtidos com DE para o projeto SRT

Estratégia	População	Crossover	Peso Diferencial	Gerações	Mínimo	Média	Máximo
DE/Rand/1	50	0,15	0,15	300	13611,0	13645,8	13659,0
DE/Best/1	50	0,15	0,15	200	13251,0	13358,0	13410,0
DE/Target-to-Best/1	50	0,15	0,15	500	12995,0	13150,0	13280,0
DE/Best/2	50	0,15	0,15	200	13293,0	13401,8	13495,0
DE/Rand/2	50	0,15	0,15	300	13629,0	13655,1	13659,0
DE/Rand/1	50	0,15	0,50	500	13460,0	13491,3	13524,0
DE/Best/1	50	0,15	0,50	500	13595,0	13627,2	13658,0
DE/Target-to-Best/1	50	0,15	0,50	300	13577,0	13628,6	13658,0
DE/Best/2	50	0,15	0,50	300	13593,0	13627,6	13656,0
DE/Rand/2	50	0,15	0,50	500	13432,0	13478,6	13538,0
DE/Rand/1	50	0,15	0,85	500	13045,0	13091,9	13154,0
DE/Best/1	50	0,15	0,85	500	13282,0	13354,5	13436,0
DE/Target-to-Best/1	50	0,15	0,85	500	13242,0	13325,3	13354,0
DE/Best/2	50	0,15	0,85	500	13264,0	13348,5	13423,0
DE/Rand/2	50	0,15	0,85	500	13055,0	13114,6	13154,0
DE/Rand/1	50	0,50	0,15	200	13559,0	13593,3	13643,0
DE/Best/1	50	0,50	0,15	50	12301,0	12518,6	12726,0
DE/Target-to-Best/1	50	0,50	0,15	200	12405,0	12710,1	12963,0
DE/Best/2	50	0,50	0,15	50	12152,0	12515,7	12780,0
DE/Rand/2	50	0,50	0,15	400	13566,0	13627,7	13658,0
DE/Rand/1	50	0,50	0,50	500	13280,0	13337,5	13437,0
DE/Best/1	50	0,50	0,50	200	13214,0	13312,2	13500,0
DE/Target-to-Best/1	50	0,50	0,50	100	13278,0	13385,9	13464,0
DE/Best/2	50	0,50	0,50	200	13144,0	13395,5	13554,0
DE/Rand/2	50	0,50	0,50	500	13280,0	13371,3	13457,0
DE/Rand/1	50	0,50	0,85	500	12530,0	12641,1	12744,0
DE/Best/1	50	0,50	0,85	500	13170,0	13289,9	13459,0
DE/Target-to-Best/1	50	0,50	0,85	500	13194,0	13310,4	13406,0
DE/Best/2	50	0,50	0,85	500	13243,0	13359,9	13474,0
DE/Rand/2	50	0,50	0,85	500	12517,0	12593,9	12728,0
DE/Rand/1	50	0,85	0,15	200	13275,0	13425,8	13552,0
DE/Best/1	50	0,85	0,15	25	11405,0	11790,4	12197,0
DE/Target-to-Best/1	50	0,85	0,15	100	12201,0	12552,1	12994,0
DE/Best/2	50	0,85	0,15	25	11598,0	11788,3	12070,0
DE/Rand/2	50	0,85	0,15	200	13299,0	13405,0	13556,0
DE/Rand/1	50	0,85	0,50	500	13434,0	13489,4	13543,0
DE/Best/1	50	0,85	0,50	50	12140,0	12582,6	12792,0
DE/Target-to-Best/1	50	0,85	0,50	50	12689,0	12991,0	13191,0
DE/Best/2	50	0,85	0,50	50	12214,0	12695,0	13112,0
DE/Rand/2	50	0,85	0,50	500	13433,0	13504,2	13590,0
DE/Rand/1	50	0,85	0,85	500	12194,0	12351,8	12577,0
DE/Best/1	50	0,85	0,85	500	13358,0	13436,4	13539,0
DE/Target-to-Best/1	50	0,85	0,85	500	13377,0	13474,5	13536,0
DE/Best/2	50	0,85	0,85	500	13373,0	13435,9	13562,0
DE/Rand/2	50	0,85	0,85	500	12185,0	12332,7	12617,0

Tabela B6 – Resultados do algoritmo PSO para o projeto SRT

Partículas	Peso Inercial	Fator Cognitivo	Fator Social	Prob. De Morte	Gerações	Mínimo	Média	Máximo
5	0,25	1,50	1,50	0,01	500	11825,0	12183,7	12406,0
10	0,25	1,50	1,50	0,01	500	12307,0	12650,8	12924,0
15	0,25	1,50	1,50	0,01	500	11948,0	12579,1	12888,0
25	0,25	1,50	1,50	0,01	500	12346,0	12856,4	13154,0
50	0,25	1,50	1,50	0,01	500	12838,0	13040,9	13151,0
5	0,75	1,50	1,50	0,01	500	12469,0	12724,9	12908,0
10	0,75	1,50	1,50	0,01	500	12537,0	12842,7	13018,0
15	0,75	1,50	1,50	0,01	500	12613,0	12986,4	13254,0
25	0,75	1,50	1,50	0,01	500	12892,0	13160,7	13406,0
50	0,75	1,50	1,50	0,01	500	12797,0	13153,4	13431,0
5	0,5	2,00	1,00	0,01	500	12134,0	12512,2	12717,0
10	0,5	2,00	1,00	0,01	500	12405,0	12933,6	13192,0
15	0,5	2,00	1,00	0,01	500	12715,0	12983,7	13175,0
25	0,5	2,00	1,00	0,01	500	12862,0	13062,9	13246,0
50	0,5	2,00	1,00	0,01	500	12968,0	13268,8	13471,0
5	0,5	1,00	2,00	0,01	500	12326,0	12580,5	12846,0
10	0,5	1,00	2,00	0,01	500	12434,0	12750,9	13121,0
15	0,5	1,00	2,00	0,01	500	12442,0	12892,1	13032,0
25	0,5	1,00	2,00	0,01	500	12332,0	12977,1	13303,0
50	0,5	1,00	2,00	0,01	500	12908,0	13181,1	13462,0
5	0,25	2,00	2,00	0,01	500	12716,0	13082,2	13307,0
10	0,25	2,00	2,00	0,01	500	13076,0	13290,6	13434,0
15	0,25	2,00	2,00	0,01	500	12993,0	13293,3	13460,0
25	0,25	2,00	2,00	0,01	500	13274,0	13444,7	13576,0
50	0,25	2,00	2,00	0,01	500	13312,0	13453,9	13544,0
5	0,75	2,00	2,00	0,01	500	11836,0	12088,1	12199,0
10	0,75	2,00	2,00	0,01	500	11893,0	12091,5	12362,0
15	0,75	2,00	2,00	0,01	500	11989,0	12135,4	12297,0
25	0,75	2,00	2,00	0,01	500	12061,0	12302,0	12553,0
50	0,75	2,00	2,00	0,01	500	12144,0	12364,6	12525,0