Universidade Federal Rural de Pernambuco
Departamento de Estatística e Informática

Algoritmos de otimização para solução do Problema do Próximo Release

Mariana Alves Moura

Recife

Janeiro de 2015

Mariana Alves Moura

Algoritmos de otimização para solução do Problema do Próximo Release

Orientadora: Silvana Bocanegra
Co-orientadora: Ana Rouiller

Monografia apresentada ao Curso Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Recife

Janeiro de 2015

Aos meus pais,
à minha irmã,
às minhas orientadoras,
à equipe SWQuality
e a todos os meus amigos.

Agradecimentos

Agradeço primeiramente a Deus.

Agradeço aos meus pais, Jozemal e Marlene, à minha irmã, Paloma e a toda a minha família por todo apoio, que foi fundamental para mim nesses cinco anos de graduação e pela compreensão e paciência com meus aperreios.

Agradeço também à professora Silvana por ter aceitado me orientar neste trabalho em um momento bastante difícil, por ter acreditado que eu iria conseguir e me dado uma segurança maior para que eu não desistisse. Sua dedicação e atenção foram essenciais para que eu tenha conseguido desenvolvê-lo e concluí-lo. Agradeço também a todos os professores da UFRPE que tanto contribuíram para a construção do meu conhecimento e me serviram de inspiração.

Agradeço a Ana por toda orientação, tanto profissional e acadêmica quanto pessoal e pelo apoio para me ajudar a conseguir pensar sobre o problema a ser abordado neste TCC. Agradeço também a todos que fazem parte da SWQuality pelas experiências trocadas e pelo conhecimento que adquiri ao longo desses anos, que também foram de grande importância para a minha formação e para a realização deste trabalho.

Também gostaria de dedicar um agradecimento especial a todos que fazem a equipe Persistir, que me ajudaram a não pirar e não desistir do trabalho quando pensei que não conseguiria terminar. Agradeço a Emeson pelos esforços para me deixar menos perdida, tentando me orientar no começo do trabalho, agradeço a Victor pela parceria, por ter dividido as atividades da SWQ comigo para que eu pudesse me dedicar mais ao TCC, a Erick, Flaviano, Eça, Thays e até Delando por estarem sempre torcendo por mim e sempre dispostos e preocupados em tentar me ajudar.

Agradeço à minha querida turma do 3º BA que foi meu principal incentivo para realizar a graduação na época da escola e continuam sendo meus exemplos para trilhar o meu caminho profissional, além de parceiros para a vida.

E por fim, agradeço a todos os meus amigos, tanto os mais antigos quanto os mais novos, que tiveram um papel primordial nesta fase da minha vida, sempre me incentivando, apoiando, compreendendo minhas ausências e muitas vezes até se dispondo a me ajudar. Obrigada por vocês existirem na minha vida!

Resumo

Este trabalho tem como objetivo apresentar sob a perspectiva de problema de otimização, um problema da Engenharia de Software, mais especificamente da Engenharia de Requisitos, comumente vivenciado pelas empresas de desenvolvimento e manutenção de software. O problema consiste na dificuldade em priorizar um conjunto ideal de requisitos para serem desenvolvidos, buscando equilibrar da melhor forma a satisfação dos clientes e as restrições de tempo e esforço às quais a equipe de desenvolvimento está sujeita. Este problema é abordado na área de Otimização em Engenharia de Software como o Problema do Próximo Release ou Next Release Problem (NRP).

A partir da sua formulação matemática como um problema de otimização combinatória, é possível adequar o NRP ao problema NP-Completo conhecido como Problema da Mochila. Com base nesta adequação, alguns algoritmos exatos e heurísticos que normalmente são utilizados para resolver este problema podem ser aplicados e comparados, com as devidas adaptações para satisfazer o Problema do Próximo Release.

Neste trabalho foram abordados os algoritmos exatos Busca Exaustiva, Programação Dinâmica e Branch and Bound, a heurística Algoritmo Guloso e as metaheurísticas Algoritmo Genético e Simulated Annealing. Os algoritmos foram testados em instâncias geradas aleatoriamente e também em uma instância de um caso real na qual o problema acontece em uma empresa de Recife, com algumas adequações na modelagem do problema para satisfazer a realidade desta empresa.

Palavras-chave: Engenharia de Software, Engenharia de Requisitos, Otimização, Algoritmos, Abordagem Exata, Heurísticas, Metaheurísticas.

Abstract

This work aims to present from the perspective of optimization problem, a problem of Software Engineering, specifically of Requirements Engineering, commonly lived by software development and maintenance companies. The problem is the difficulty in prioritizing an ideal set of requirements to be developed, in order to balance the best customer satisfaction and the restrictions of time and effort which the development team is subject. This issue is discussed in the area of Search Based Software Engineering (SBSE) as the Next Release Problem (NRP).

From its mathematical formulation as a combinatorial optimization problem, it's possible adjust the NRP to the NP-Complete problem known as Knapsack Problem. Based on this adaptation, some exact and heuristic algorithms that normally apply to this problem can be applied and compared with the necessary adaptations to meet the Next Release Problem.

In this work were approached the exact algorithms Exhaustive Search, Dynamic Programming and Branch and Bound, the heuristic Greedy Algorithm and the metaheuristics Genetic Algorithm and Simulated Annealing. The algorithms were tested on randomly generated instances and also in an instance of a real case in which the problem occurs in a company from Recife, with some adjustments in the problem of modeling to meet the reality of the company.

Keywords: Software Engineering, Requirements Engineering, Optimization, Algorithms, Exact Approach, Heuristics, Metaheuristics.

Sumário

1 Introdução **1**

1.1 Visão geral 1

1.2 Objetivos 2

1.3 Justificativa 2

1.4 Organização do documento 3

2 Formulação do problema **4**

3 Métodos de solução **9**

3.1 Algoritmos exatos 9

3.1.1 Enumeração Exaustiva ou Força Bruta 10

3.1.2 Programação Dinâmica 11

3.1.3 Branch and Bound 12

3.2 Algoritmos heurísticos e metaheurísticas 14

3.2.1 Algoritmo Guloso 15

3.2.2 GRASP - Greed Randomized Adaptative Search Procedure 16

3.2.3 Simulated Annealing 17

3.2.4 Algoritmo Genético 19

3.2.5 Otimização da Colônia de Formigas 21

3.3 Estado da Arte 22

4 Estudo de caso **25**

4.1 O problema na Empresa X 25

4.2 Implementação dos algoritmos e detalhamento das instâncias e dos testes . . 29

4.3 Testes e comparação entre as metodologias de solução 33

4.3.1 Validação 33

4.3.2 Testes na instância da Empresa X 35

4.3.3 Testes nas instâncias aleatórias 37

5 Conclusão e trabalhos futuros **41**

Lista de Tabelas

4.1 Associação dos elementos do problema original aos do estudo de caso. 27

4.2 Custos dos requisitos do caso de validação. 31

4.3 Matriz de importância dos requisitos para os clientes do caso de validação. . 31

4.4 Pesos de importância dos clientes do caso de validação. 31

4.5 Custos dos requisitos da Empresa X. 32

4.6 Pesos de importância dos clientes da Empresa X. 32

4.7 Algoritmos exatos e Guloso na instância de validação. 34

4.8 Aplicação do algoritmo Simulated Annealing na instância de validação. . . . 34

4.9 Aplicação do Algoritmo Genético na instância de validação. 34

4.10 Algoritmos exatos e Guloso na instância da Empresa X. 35

4.11 Algoritmo Simulated Annealing na instância da Empresa X. 35

4.12 Algoritmo Genético na instância da Empresa X. 35

4.13 SA na instância da Empresa X considerando precedência. 36

4.14 AG na instância da Empresa X considerando precedência. 36

4.15 Algoritmos exatos e Guloso - tamanho das instâncias. 37

4.16 Busca Gulosa - aproximação da solução ótima. 38

4.17 Simulated Annealing na instância da tamanho 100. 39

4.18 Simulated Annealing na instância da tamanho 2000. 39

4.19 Algoritmo Genético na instância da tamanho 100. 39

4.20 Algoritmo Genético na instância da tamanho 2000. 39

Lista de Figuras

2.1 Representação gráfica do exemplo do Problema do Próximo Release. 7

Lista de Algoritmos

1 Pseudocódigo do algoritmo Enumeração Exaustiva para o Problema da Mochila. 10

2 Pseudocódigo de Programação Dinâmica para o Problema da Mochila. 11

3 Pseudocódigo do algoritmo Branch and Bound para o Problema da Mochila. . 13

4 Pseudocódigo da função Limite Superior do Branch and Bound. 14

5 Pseudocódigo do Algoritmo Guloso para o Problema da Mochila. 15

6 Pseudocódigo da função principal do GRASP. 17

7 Pseudocódigo da função de construção do GRASP. 17

8 Pseudocódigo do algoritmo simulated annealing. 19

9 Pseudocódigo do algoritmo genético. 21

10 Pseudocódigo da função de avaliação do AG para o Problema da Mochila. . . 21

11 Pseudocódigo do ACO para o Problema da Mochila. 22

Capítulo 1

Introdução

1.1 Visão geral

Na Engenharia de Software é comum nos depararmos com problemas que envolvem encontrar um equilíbrio adequado entre objetivos concorrentes e potencialmente conflitantes. Normalmente são problemas que envolvem tomar decisões sob um conjunto de inúmeras opções, o que dificulta a escolha de boas soluções. De acordo com [17], alguns exemplos dessas situações são:

- Escolher a melhor maneira de estruturar a arquitetura de um sistema;
- Determinar um conjunto de requisitos que equilibre o custo de desenvolvimento e a satisfação do cliente;
- Selecionar o menor conjunto de casos de teste que abrangem todo o sistema;
- Indicar a alocação de recursos ideal para um projeto;
- Definir a melhor sequência de etapas de refatoração para aplicar em um sistema.

À primeira vista, respostas para estes problema podem ser encontradas por meio de conhecimentos na literatura das diversas áreas da Engenharia de Software, com pouca coisa em comum entre eles. Porém, todas essas questões são essencialmente problemas de otimização, ou seja, problemas onde o objetivo é encontrar o valor mínimo ou máximo que uma função pode assumir por meio da escolha sistemática de variáveis reais ou inteiras dentro de um

conjunto viável. São problemas típicos abordados por uma área de pesquisa conhecida como Search Based Software Engineering (SBSE) ou, Otimização em Engenharia de Software [16]. Esta área tem como objetivo resolver problemas de Engenharia de Software reformulando-os como um problema de otimização e aplicando técnicas de busca para solucioná-los.

Neste trabalho será abordado o Problema do Próximo Release, conhecido na literatura como The Next Release Problem (NRP) [3] cujo objetivo é selecionar o conjunto ideal de requisitos que irá compor a próxima release (ou versão) de um software a ser liberada, buscando equilibrar da melhor maneira o custo de desenvolvimento dos requisitos do sistema e a satisfação dos clientes.

1.2 Objetivos

Este trabalho tem como principais objetivos:

- Entender o Problema do Próximo Release;
- Entender o funcionamento de alguns algoritmos de otimização comumente aplicados ao Problema do Próximo Release;
- Adaptar os algoritmos para resolver o problema e aplicá-los a uma instância real, originada em uma empresa de Tecnologia da Informação de Recife;
- Comparar a eficiência destes algoritmos na solução do problema em instâncias geradas aleatoriamente e também ao problema da empresa.

1.3 Justificativa

Um problema bastante comum para as empresas que trabalham com evolução e manutenção de produtos de software desenvolvidos para vários clientes é priorizar o que será implementado no próximo release do software. Isto porque diversos fatores podem influenciar nessa escolha, como:

- A crescente demanda dos clientes por melhorias nos softwares;

- A dependência de desenvolvimento entre os requisitos do software;
- O valor de negócio que cada cliente representa para a organização;
- A quantidade de tempo e esforço diferenciados que cada requisito demandará da equipe de desenvolvimento.

Esta escolha pode tornar-se bastante complicada quando trata-se de avaliar manualmente cada um desses parâmetros e muitas vezes é negligenciada pelas empresas, já que é despendido um tempo considerável para que uma boa solução seja encontrada. Dessa forma, por meio da reformulação deste problema como um problema de otimização e aplicando a ele os algoritmos apropriados, é possível tornar este processo mais automatizado e atingir resultados bastante satisfatórios, tanto em relação à qualidade das soluções quanto ao tempo gasto, que seriam difíceis de se alcançar apenas por meio de esforço humano.

1.4 Organização do documento

O presente trabalho está organizado em cinco capítulos dos quais o primeiro é a introdução e os próximos quatro capítulos estão descritos abaixo:

- No capítulo 2 é descrita a formulação matemática para o Problema do Próximo Release;
- No capítulo 3 são apresentados os métodos de solução para o problema, assim como alguns algoritmos comumente aplicados para resolvê-lo. Neste capítulo também é exposto o estado da arte, onde são apresentados alguns trabalhos já existentes na literatura que abordam a aplicações destes algoritmos no NRP;
- No capítulo 4 é apresentado um caso real ao qual os algoritmos puderam ser aplicados para fins de testes. Ainda neste capítulo estão registrados os resultados das validações e dos testes realizados com os algoritmos e uma breve comparação entre seus resultados;
- No capítulo 5 são descritas as conclusões do trabalho e os trabalhos futuros.

Capítulo 2

Formulação do problema

Neste capítulo será apresentada a formulação matemática de um problema frequentemente vivenviado em empresas de desenvolvimento de software. O problema é originado da dificuldade enfrentada pelas equipes de desenvolvimento para selecionar o conjunto de requisitos que irá compor a release seguinte, levando em consideração que estes requisitos são demandados por diferentes clientes, com graus de importância variados para o negócio. Cada requisito possui um custo de desenvolvimento e o conjunto selecionado deve caber em uma limitação de custos pré-definida. Logo, o problema é conseguir selecionar um conjunto ideal de requisitos que satisfaça os clientes da melhor maneira e respeite a limitação de custos da organização.

Este problema foi denominado como *Next Release Problem* (NRP) ou Problema do Próximo Release e foi modelado matematicamente pela primeira vez em [3], artigo utilizado como base para a modelagem descrita abaixo.

Seja R o conjunto de todos os requisitos de software a serem desenvolvidos por uma equipe de desenvolvimento. Cada cliente, i, terá um conjunto de requisitos, $R_i \subseteq R$ e um valor $v_i \in \mathbb{Z}^+$ que mensura a importância do cliente para a organização.

Cada $r \in R$ possui um custo associado, $c_r \in \mathbb{Z}^+$, que é o custo de implementação do requisito r. E a empresa possui uma limitação de custo de implementação L, que é o limite de capacidade de desenvolvimento da equipe.

Associado ao conjunto R está um grafo $G = (R, E)$, direcionado e acíclico, onde R é o conjunto de vértices, E é o conjunto de arestas e $(r, r') \in E(G)$ se, e somente se, r é um

pré-requisito de r'. G não é apenas acíclico, ele também é transitivo, desde que $(r, r') \in E(G)$ & $(r', r'') \in E(G) \rightarrow (r, r'') \in E(G)$.

Se a empresa decidir satisfazer todos requisitos do cliente i, ela não deverá desenvolver apenas R_i, mas também

$$parents(R_i) = \{r \in R | (r, r') \in E(G), r' \in R\}.$$

Assumindo que a empresa possui n clientes, o desafio é encontrar um subconjunto de clientes, $S \subseteq \{1, 2, ..., n\}$ cujos requisitos serão satisfeitos.

Este problema pode ser formulado como um problema de otimização, no qual temos:

1. **Variáveis de decisão:**

   $$y_i = \begin{cases} 1 & \text{se o cliente i é atendido} \\ 0 & \text{caso contrário} \end{cases}, \forall i \in S.$$

   $$x_j = \begin{cases} 1 & \text{se o requisito j é implementado} \\ 0 & \text{caso contrário} \end{cases}, \forall j \in R.$$

2. **Função objetivo:**

   $$max \sum_{i \in S} v_i y_i,$$

   onde v_i representa a importância do cliente i para a empresa.

3. **Restrições:**

   3.1. *Limitação de custo de implementação* - Diversos fatores podem influenciar esta limitação dentro de uma empresa. Alguns exemplos são prazos e recursos limitados. Dessa forma, temos

   $$\sum_{j \in R} c_j x_j \leq L,$$

   onde c_j é o custo de implementação do requisito r_j e L é a limitação de custo de implementação da empresa.

3.2. *Dependência entre requisitos* - A dependência entre os requisitos pode ser representada pela seguinte equação:

$$x_r \geq x_{r'}, \forall (r, r') \in E(G),$$

isso significa que um dado requisito r' só pode ser implementado se seu pré-requisito r também for.

3.3. *Garantia que se o cliente é atendido, todos os requisitos solicitados por ele devem ser implementados.*

$$x_j \geq y_i, \forall i, j, \text{ tal que } r_j \text{ é requisito do cliente } i.$$

Sintetizando, tem-se:

$$max \sum_{i \in S} v_i y_i,$$

$$\text{sujeito a } \begin{cases} \sum_{j \in R} c_j x_j \leq L & \\ x_r \geq x_{r'}, & \forall (r, r') \in E(G) \\ x_j \geq y_i, & \forall \text{i,j, tal que } r_j \text{ é requisito do cliente } i \end{cases}$$

onde v_i representa o valor do cliente i, c_j representa o custo de implementação do requisito r_j e L é a limitação de custo de implementação da equipe de desenvolvimento.

Um exemplo simples para facilitar o entendimento pode ser observado a seguir:

$R = \{r_1, r_2, ..., r_7\}$,
$c_1 = 10, c_2 = 6, c_3 = 7, c_4 = 1, c_5 = 4, c_6 = 6, c_7 = 1$,
$R_1 = \{r_6\}, R_2 = \{r_6, r_7\}, R_3 = \{r_5\}$,
$E = \{(1,3), (1,4), (1,6), (1,7), (2,5), (2,7), (3,6), (4,7), (5,7)\}$,
$\hat{R_1} = \{r_1, r_3, r_6\}, \hat{R_2} = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}, \hat{R_3} = \{r_2, r_5\}$,
$v_1 = 50, v_2 = 60, v_3 = 70$.

Uma representação gráfica da estrutura deste problema é mostrado na Figura 2.1.

Expandindo a formulação do problema temos que nosso objetivo é maximizar

$50y_1 + 60y_2 + 70y_3$,

Sujeito a

- Limitação de custo de implementação:

  $10x_1 + 6x_2 + 7x_3 + 1x_4 + 4x_5 + 6x_6 + c_7x_7 \leq L.$

- Dependência entre requisitos:

  $x_1 \geq x_3$
  $x_1 \geq x_4$
  $x_2 \geq x_5$
  $x_3 \geq x_6$
  $x_4 \geq x_7$
  $x_5 \geq x_7,$
  $x_j \in \{0,1\}, y_i \in \{0,1\}.$

- Garantia que se o cliente é atendido, todos os requisitos solicitados por ele devem ser implementados

  $x_6 \geq y_1$
  $x_6 \geq y_2$
  $x_7 \geq y_2$
  $x_5 \geq y_3$
  $x_j \in \{0,1\}, y_i \in \{0,1\}.$

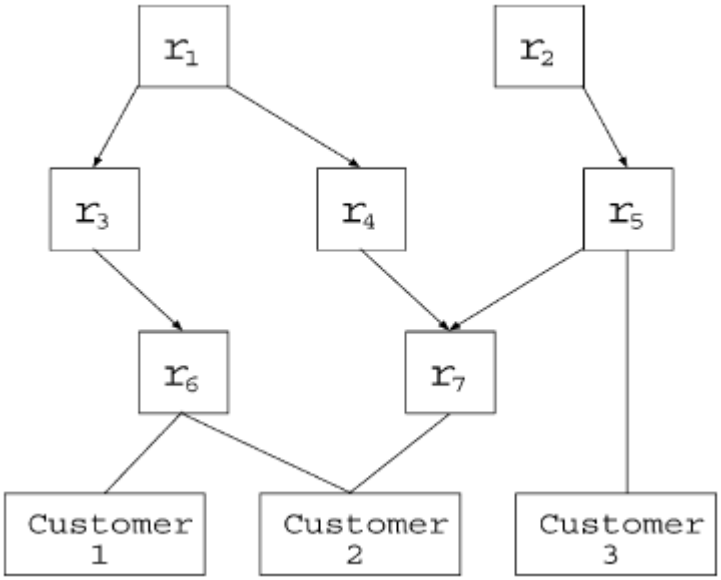

Figura 2.1: Representação gráfica do exemplo do Problema do Próximo Release.

Em um caso especial da formulação do problema, onde a única restrição é a limitação de custo de implementação, o problema é básico e pode ser formulado como o Problema da Mochila, que pode ser enunciado da seguinte maneira:

"Um viajante levará para sua viagem apenas uma mochila, de capacidade limitada, a qual ele deverá preencher com objetos que lhe serão úteis para a viagem. Cada objeto ocupa uma certa capacidade da mochila e possui um determinado valor de importância para o viajante. O problema é decidir quais objetos deverão ser colocados na mochila de tal forma que o valor de importância seja maximizado."

O Problema da Mochila possui diferentes abordagens de formulação. Para o problema aqui tratado, a abordagem utilizada será a 0/1 unidimensional, que considera que a mochila possui apenas uma dimensão e assume que cada objeto possui apenas um peso e um valor pertencentes ao conjunto dos números naturais. Para cada um dos n objetos j, $x_j = 1$ se este objeto estiver na mochila e caso não esteja, $x_j = 0$. p_j é o valor do objeto j, w_j seu peso e W é a capacidade total da mochila. A solução é encontrar um vetor $x(x_1, x_2, ..., x_n)$ que maximize

$$\sum_{j=1}^{n} p_j x_j (p_j \in \mathbb{N}^*)$$

sujeito a

$$\sum_{j=1}^{n} w_j x_j \leq W (w_j, W \in \mathbb{N}^*),$$

onde $x_j \in \{0, 1\}, 1 \leq j \leq n$.

Se as variáveis do problema pudessem assumir valores reais, ele seria um problema de Programação Linear e uma solução ótima poderia ser encontrada em um tempo razoável, já que problemas de Programação Linear podem ser resolvidos em tempo polinomial. Como as variáveis são inteiras, podendo apenas assumir valores binários, temos um caso de Programação Linear Inteira que pode ser classificado como NP-Difícil [5], ou seja, não é conhecido nenhum algoritmo de complexidade polinomial para solucioná-lo.

Capítulo 3

Métodos de solução

O Problema do Próximo Release [3] é considerado um problema bastente conhecido na área de Otimização em Engenharia de Software [16]. Diversas soluções são encontradas na literatura: algumas exatas, que garantem uma solução ótima, porém nem sempre podem chegar a essa solução em tempo polinomial e outras aproximadas, que não garantem uma solução ótima mas permitem que sejam encontradas soluções aproximadas em tempo polinomial.

A seguir, serão apresentados alguns algoritmos comumente aplicados para solucionar o Problema da Mochila Booleana.

3.1 Algoritmos exatos

Algoritmos exatos utilizam operações matemáticas sobre os dados do problema e buscam a solução ótima, ou seja, para o NRP, um algoritmo exato deve encontrar o conjunto de requisitos que possui o maior valor para a organização, dentro das limitações de custo estabelecidas.

Alguns desses algoritmos que podem ser aplicados ao problema estão descritos abaixo:

3.1.1 Enumeração Exaustiva ou Força Bruta

Por ser um problema combinatório, é possível ser resolvido por Enumeração Exaustiva, ou seja, comparando todas as possibilidades possíveis até descobrir a melhor. O melhor resultado é guardado numa variável e, ao final de todas as comparações, a melhor solução fica armazenada nesta variável.

Este algoritmo sempre traz a solução ótima, porém, seu tempo de execução é exponencial já que ele gera todas as 2^n configurações candidatas à solução, sendo n, no caso do Problema da Mochila, a quantidade de objetos candidatos a preencherem mochila. Isto torna seu custo computacional bastante alto, mesmo para uma quantidade pequena de dados.

Algoritmo 1: Pseudocódigo do algoritmo Enumeração Exaustiva para o Problema da Mochila.

Entrada: Conjunto de itens I, Capacidade da mochila c
Saída: Solução ótima M
inicio
  Inicialização da mochila vazia: $M = \emptyset$ Peso inicial: $melhorpeso = 0$;
  Valor inicial: $melhorvalor = 0$;
  Gerar o conjunto C de todas as combinações de itens possíveis: $C =$ `Gerar Combinações(`I`)`;
  para cada $combinacao \in C$ **faça**
    $p = \sum_{i=1}^{n}$ `Peso(`i`)`, $n =$ Total de itens da combinação de itens;
    $v = \sum_{i=1}^{n}$ `Valor(`i`)`, $n =$ Total de itens da combinação de itens;
    se $(v > melhorvalor) \wedge (p \leq c)$ **então**
      $melhorpeso = p$;
      $melhorvalor = v$;
      $M = combinacao$;
    fim
  fim
fim

3.1.2 Programação Dinâmica

Programação Dinâmica é uma técnica para resolver problemas cujas soluções satisfazem relações de recorrência com subproblemas sobrepostos. Esta técnica resolve cada um dos problemas menores apenas uma vez e registra os resultados em uma tabela para evitar recálculo e esses subproblemas sobrepostos compõem o problema original.[18]

Para que a Programação Dinâmica seja aplicável a um problema de otimização ele deve ter uma subestrutura ótima e superposição de subproblemas, ou seja, um problema apresenta uma subestrutura ótima quando uma solução ótima para o problema contém em seu interior soluções ótimas para subproblemas e a superposição de subproblemas é dada quando um algoritmo recursivo reavalia o mesmo problema diversas vezes.

Algoritmo 2: Pseudocódigo de Programação Dinâmica para o Problema da Mochila.

Entrada: Número de itens candidatos, Lista de pesos $[p_1, ..., p_n]$, Lista de valores $[v_1, ..., v_n]$, Tabela iniciada com 0's: $[0, .., n][0...N, 0...Capacidade]$

Saída: Tabela com a solução ótima: [N,Capacidade]

```
inicio
    N = número de itens candidatos;
    Pesos = lista de pesos;
    Valores = lista de valores;
    Tabela = Tabela iniciada com 0's;
    para i = 0 → N faça
        para j = 0 → Capacidade faça
            se j < Pesos[i] então
                Tabela[i, j] ← Tabela[i − 1, j];
            fim
            senão
                Tabela[i, j] ←
                maximo{Tabela[i − 1, j] ∧ Valores[i] + Tabela[i − 1, j − Pesos[i]]}
            fim
        fin
    fin
fim
```

3.1.3 Branch and Bound

O método Branch and bound, proposto pela primeira vez em [20], utiliza a estratégia de dividir para conquistar e também encontra sempre a solução ótima, porém, diferente da enumeração exaustiva, não percorre todo o espaço de soluções: ele percorre uma árvore de enumeração (branch) que particiona o conjunto de soluções do problema e, sempre que percorre um ramo que não é promissor ou que é inviável, ele poda-o (bound) sem percorrê-lo.

A poda é realizada de acordo com a estimativa dos limites superior e inferior para o valor mínimo (caso a função objetivo seja minimizar) ou máximo (caso a função objetivo seja maximizar), dado um subconjunto S. Com a comparação desses limites é possível eliminar subespaços de S que possuam soluções fracas.

Em geral, o método Branch and Bound possui um custo computacional bem menor que a enumeração exaustiva, porém, no pior caso seu custo também é exponencial.

Algoritmo 3: Pseudocódigo do algoritmo Branch and Bound para o Problema da Mochila.

Entrada: Número de itens candidatos, Lista de pesos $[p_1, ..., p_n]$, Lista de valores $[v_1, ..., v_n]$, Capacidade da mochila c. Obs.: Os itens devem estar ordenados de acordo com a razão valor/peso.

Saída: Solução ótima *melhorsolucao*

inicio

N = número de itens candidatos;

Pesos = lista de pesos;

Valores = lista de valores;

Conjunto de nós ativos: $NosAtivos \leftarrow \{raiz\}$;

valormaximo = `Valor` (raiz);

Inicialização do conjunto da melhor solução: $melhorsolucao = \emptyset$;

repita

Escolher um nó para ramificar: no = `Escolher(`*NosAtivos*`)`;

se `Limite Superior` *(no, c) > valormaximo* **então**

Gerar filho esquerdo (considera que o próximo item entra na mochila) do nó corrente para incluir o próximo item;

se `Valor Acumulado` *(filho esquerdo) > valormaximo* **então**

valormaximo = `Valor Acumulado` (filho esquerdo);

Atualizar melhorsolucao;

fim

se `Limite Superior` *(filho esquerdo, c) > valormaximo* **então**

$NosAtivos \leftarrow \{filhoesquerdo\}$;

fim

Gerar filho direito (considera que o próximo item não entra na mochila) do nó corrente para incluir o próximo item;

se `Limite Superior` *(filho direito, c) > valormaximo* **então**

$NosAtivos \leftarrow \{filhodireito\}$;

fim

fim

até $NosAtivos = \emptyset$;

fim

Algoritmo 4: Pseudocódigo da função Limite Superior do Branch and Bound.

Entrada: no, Capacidade da mochila

Saída: Limite superior: *limitesuperior*

inicio

limitesuperior = `Valor Acumulado` (no) + (Capacidade da mochila - `Peso` (no)) $* \left(\frac{\texttt{Valor}(proximo - no)}{\texttt{Peso}(proximo - no)} \right)$;

fim

3.2 Algoritmos heurísticos e metaheurísticas

Algoritmos heurísticos buscam soluções para o problema sem a garantia de que essas soluções são ótimas. No entanto, geralmente baseiam-se em sucessivas aproximações direcionadas a um ponto ótimo, tentando encontrar as melhores soluções possíveis para um determinado problema. Porém, em alguns casos, é possível que sejam encontrados resultados arbitrariamente ruins, assim como também resultados ótimos. O uso dessa abordagem é motivado pela busca do custo-benefício, tendo em vista que talvez não tenhamos a melhor solução possível, mas geralmente obtemos uma solução viável com um custo computacional aceitável.

Dentro da categoria das heurísticas existem as metaheurísticas. Metaheurísticas são estratégias genéricas para a construção de heurísticas e podem ser utilizadas para encontrar soluções para problemas para os quais não se conhece um algoritmo eficiente especializado em resolvê-lo.

De acordo com [1], uma metaheurística é um conjunto de conceitos que podem ser utilizados para definir métodos heurísticos aplicáveis a um extenso conjunto de diferentes problemas. Em outras palavras, uma metaheurística pode ser vista como uma estrutura algorítmica geral que pode ser aplicada a diferentes problemas de otimização com relativamente poucas modificações que possam adaptá-la a um problema específico.

A seguir estão alguns exemplos de heurísticas e metaheurísticas comumente aplicadas ao Problema do Próximo Release.

3.2.1 Algoritmo Guloso

Algoritmo Guloso ou Ganancioso é uma técnica heurística de resolução de problemas que busca solucionar problemas de otimização realizando a escolha que parece ser a melhor no momento de acordo com uma função heurística, também chamada de função gulosa. Ele faz uma escolha ótima local e adiciona na solução parcial, na esperança de que esta escolha leve à solução ótima global. Esta escolha sempre é definitiva, ou seja, uma vez que o método guloso define que um elemento está na solução, este não sairá mais dela. Estes algoritmos nem sempre chegam na solução ótima, mas, quando chegam, geralmente são avaliados como os mais simples e eficientes em uma comparação com outros algoritmos. A versão proposta por [6] para o Problema da Mochila consiste em ordenar os itens em ordem decrescente da razão "*valor/peso*". Então, ele começa a inserir os itens na mochila até atingir o limite de peso pré-estabelecido.

Algoritmo 5: Pseudocódigo do Algoritmo Guloso para o Problema da Mochila.

Entrada: Conjunto de itens I, Capacidade da mochila c, Função gulosa f
Saída: Melhor solução encontrada M
inicio
  Inicialização da mochila vazia: $M = \emptyset$ Peso inicial da mochila: $pesomochila = 0$;
  Valor inicial: $valormochila = 0$;
  Ordenar os itens da mochila de acordo com a função gulosa:
  $Itensordenados =$ `Ordenar(`I, f`)`;
  repita
    Seleciona o primeiro item da lista ordenada e apaga ele da lista;
    se `Peso`$(item) + pesomochila \leq c$ **então**
      $M \leftarrow item$;
      $pesomochila+ =$ `Peso`$(item)$;
      $valormochila+ =$ `Valor`$(item)$;
    fim
    senão
      Sair do loop;
    fim
  até $Itensordenados = \emptyset$;
fim

3.2.2 GRASP - Greed Randomized Adaptative Search Procedure

GRASP [13] é um processo iterativo cujas iterações se dividem em duas fases: uma fase de construção e outra de busca local.

Na primeira fase uma solução viável é construída de forma iterativa, um elemento por vez. Em cada iteração o próximo elemento é escolhido ao ordenar uma lista de elementos candidatos, respeitando uma função gulosa. Esta função mede o benefício da seleção de cada elemento. A heurística é adaptativa, pois os benefícios associados a cada elemento são atualizados a cada iteração da fase de construção, para refletir as mudanças trazidas pela seleção do elemento anterior. O componente probabilístico de um GRASP é caracterizado pela escolha aleatória de um dos melhores candidatos na lista, mas não necessariamente o melhor candidato. Esta técnica de escolha permite que diferentes soluções sejam obtidas em cada iteração do GRASP, sem necessariamente comprometer a potência do componente guloso adaptativo do método. A primeira fase termina quando algum critério de parada é satisfeito, quando alcança o número máximo de iterações ou quando encontra a solução procurada.

Assim como em muitos métodos determinísticos, as soluções geradas na fase de construção do GRASP não tem garantia de que são ótimas, por isso, o método possui uma segunda fase onde aplica-se uma busca local para tentar melhorar a solução construída. Um algoritmo de busca local funciona de forma iterativa, substituindo sucessivamente a solução atual por um vizinho que seja uma solução melhor. Ele termina quando não for encontrada uma solução melhor.

Algoritmo 6: Pseudocódigo da função principal do GRASP.

Entrada: Conjunto de itens da mochila *Itens*

Saída: Melhor solução encontrada: *melhorsolucaoatual*

inicio

Inicialização da melhor solucão corrente: $melhorsolucaoatual = \emptyset$;

repita

$solucao =$ `Construir Solução(`*Itens*`)`;

$novasolucao =$ `Busca Local(`*solucao*`)`;

se *novasolucao é melhor que melhorsolucaoatual* **então**

$melhorsolucaoatual = novasolucao$;

fim

até *Critério de parada satisfeito.*;

fim

Algoritmo 7: Pseudocódigo da função de construção do GRASP.

Entrada: Conjunto de itens do problema *Itens*

Saída: Solução: *solucao*

inicio

Inicialização da solucão: $solucao = \emptyset$;

Calcular custos dos itens do problema: c = `Calcular Custo(`*Itens*`)`;

repita

Construir Lista Restrita de Candidatos a partir dos custos e das restrições do problema: $LRC =$ `Lista Restrita de Candidatos(`*c, Itens*`)`;

Selecionar aleatoriamente um elemento de LRC:

$s =$ `Selecionar Elemento(`LRC`)` $solucao = solucao \cup s$;

Recalcular os custos;

até *solucao não é uma solução completa*;

fim

3.2.3 Simulated Annealing

O Simulated Annealing é um algoritmo de busca local que explora a analogia entre os problemas de otimização combinatória e os da mecânica estatística [19]. É uma metaheurística probabilística e genérica utilizada para encontrar uma solução ótima global, ou aproximada,

em um vasto espaço de busca. Ela é baseada em uma analogia a um processo da Termodinâmica, que emula as mudanças de energia que ocorrem num sistema de partículas quando sua temperatura é reduzida, até que ele atinja um estado de equilíbrio. Neste processo, o calor leva os átomos a se desprenderem de suas posições iniciais (um mínimo local da energia interna) e se moverem aleatoriamente; o resfriamento lento aumenta a probabilidade destes átomos acharem configurações com energias internas menores do que a inicial.

Analogamente a este processo, esta técnica começa sua busca a partir de uma solução inicial qualquer e substitui esta solução por uma solução próxima (na vizinhança), escolhida de acordo com uma função objetivo e uma variável T (Temperatura). O procedimento principal consiste em um laço que gera aleatoriamente, a cada iteração, um vizinho da solução corrente e tenta substituir a solução atual pela nova solução. A nova solução pode ser aceita com uma probabilidade que depende da diferença dos valores de função correspondentes às soluções e de T, que é gradualmente diminuído durante o processo.

Quando T é um valor alto, a aceitação é quase aleatória. Conforme T diminui, a aceitação fica mais restrita, eventualmente aceitando somente soluções melhores. Este processo evita que o Simulated Annealing fique preso em ótimos locais (o grande problema de métodos gulosos).

Algoritmo 8: Pseudocódigo do algoritmo simulated annealing.

Entrada: Conjunto de itens I, Capacidade da mochila c

Saída: Melhor solução encontrada *melhorsolucao*.

inicio

  Geração da solução inicial: $S = [0, ..., 0]$;

  Inicialização da temperatura: $T = T_0$;

  $melhorsolucao = S$;

  repita

    Gerar uma solução aleatória Y;

    $\Delta E = \texttt{Valor}(Y) - \texttt{Valor}(S)$;

    se $\texttt{Peso}(Y) \leq c$ **então**

      se $\Delta E > 0$ **então**

        $S = Y$;

        se $\texttt{Valor}(S) > \texttt{Valor}(melhorsolucao)$ **então**

          $melhorsolucao = S$;

        fim

      fim

      senão

        Aceitar Y com uma probabilidade $e\frac{-\Delta E}{T}$;

      fim

    fim

    Atualização da temperatura: $T = g(T)$;

  até *Critério de parada satisfeito. (Por exemplo,* $T \geq T_{max}$*)*;

fim

3.2.4 Algoritmo Genético

Algoritmos Genéticos [14] são uma classe particular de algoritmos evolutivos que usam técnicas inspiradas pela biologia evolutiva como hereditariedade, mutação, seleção natural e recombinação (ou crossing-over). Estes algoritmos mantém uma população formada por um subconjunto de soluções criadas aleatoriamente e, através dos operadores genéticos e de critérios de seleção, cada iteração do algoritmo substitui uma população de indivíduos por outra, com valores de aptidão, em média, melhores.

A idéia básica consiste em acreditar que os melhores indivíduos sobrevivem e geram descendentes com suas características hereditárias, de maneira semelhante à teoria biológica das espécies. De forma análoga, os algoritmos genéticos partem de um conjunto de soluções iniciais geradas aleatoriamente, avaliam seus custos, selecionam um subconjunto das melhores soluções e realizam recombinações e mutações sobre elas, gerando um novo conjunto de soluções ou uma nova população. A nova população então é utilizada como entrada para a próxima iteração do algoritmo.

Nessa abordagem, os indivíduos são portadores do seu código genético, e a relação com a otimização trata dessa como uma representação do espaço de busca do problema a ser resolvido, em geral na forma de sequências de bits[21].

A recombinação, ou crossing-over é um processo que imita o processo biológico homônimo. Os descendentes recebem em seu código genético parte do código genético dos genitores. Esta recombinação garante que os melhores indivíduos sejam capazes de trocar entre si as informações que os levam a ser mais aptos a sobreviver, e assim gerar descendentes ainda mais aptos. Finalmente, as mutações, que são feitas de forma aleatória, tem como objetivo permitir maior variabilidade genética na população, impedindo que a busca permaneça estagnada em um mínimo local [14].

Algoritmo 9: Pseudocódigo do algoritmo genético.

Entrada: Conjunto de itens candidatos I, Capacidade da mochila c

Saída: Melhor solução encontrada.

inicio

Geração atual: $t = 0$;

Inicializa uma população: P = `Iniciar População`(I, t);

Avalia aptidão dos indivíduos da população: `Avaliação`(P, t);

repita

$t = t + 1$;

Seleciona os pares para cruzamento: `Avaliação`(P, t);

Realiza cruzamento: `Recombinação`(P, t);

Perturba o grupo gerado pelo cruzamento: `Mutação`(P, t);

Avalia as novas aptidões: `Avaliação`(P, t);

Seleciona os sobreviventes: `Sobrevivem`(P, t);

até *Critério de parada do algoritmo*;

fim

Algoritmo 10: Pseudocódigo da função de avaliação do AG para o Problema da Mochila.

inicio

para *cada indivíduo da população* **faça**

Aptidão do indivíduo = valor do indivíduo;

se *peso do indivíduo excede c* **então**

Penalizar aptidão do indivíduo;

fim

fin

fim

3.2.5 Otimização da Colônia de Formigas

A Otimização da Colônia de Formigas [11] é uma metaheurística para otimização combinatorial, proposta por [11], criada para solução de problemas computacionais que envolvem procura de caminhos em grafos. Este algoritmo foi inspirado na observação do comportamento das formigas ao saírem de sua colônia para encontrar comida. Esta atividade de

busca é realizada por meio de uma substância chamada feromônio, utilizada pelas formigas para se comunicar com as outras. Elas deixam um rastro de feromônio no chão que marca o caminho encontrado. Se outras formigas encontrarem e seguirem o mesmo caminho, esta trilha de feromônios é fortalecida, atraindo outras formigas para segui-la. Uma característica do feromônio é que ele evapora com o tempo, tornando as trilhas menos percorridas menos desejáveis. À primeira vista, quando nenhum rastro de feromônio está presente no ambiente, a movimentação pode parecer aleatória para as formigas, porém, a partir do momento em que as formigas começam a se movimentar pelo ambiente, os rastros são deixados e começam a influenciar as outras formigas.

Algoritmo 11: Pseudocódigo do ACO para o Problema da Mochila.

Entrada: Conjunto de formigas F, Conjunto de itens I, Número de itens N, Capacidade da mochila c

Saída: Melhor solução encontrada ou um conjunto de soluções.

```
inicio
    Inicializar trilha de feromônios a partir de N;
    repita
        para cada formiga ∈ F faça
            Adicionar um objeto aleatório à mochila;
            Adicionar objetos à mochila de acordo com seus valores até que alcance c;
            Construir uma solução usando a trilha de feromônios;
        fim
        Ajustar a melhor solução encontrada por cada colônia de formigas;
        Atualizar a trilha de feromônios utilizando a melhor solução da colônia e a
        melhor solução gerada nesta rodada pela colônia.;
    até Critério de parada satisfeito.;
fim
```

3.3 Estado da Arte

A primeira modelagem matemática para o Problema do Próximo Release foi apresentada por [3], como um problema de otimização mono-objetivo que busca maximizar a satisfação dos clientes sujeito a limitações de recursos e interdependência entre requisitos. Neste artigo foram comparadas algumas técnicas exatas, algoritmos gulosos e técnicas de busca local

para tentar obter uma solução para o problema. Dentre estas abordagens, as técnicas exatas foram menos eficientes para instâncias com valores altos e o melhor resultado foi encontrado com o uso da metaheurística Simulated Annealing[27].

Em [15] o problema é direcionado ao planejamento de *releases* para o desenvolvimento incremental de software, considerando todas as limitações técnicas, riscos, limitações de recursos e de orçamento. Foi utilizada uma abordagem chamada ENVOLVE+, que combina a robustez computacional de algoritmos genéticos com a flexibilidade de um método de solução iterativo.

Considerando a importância das capacidades humanas de intuição e negociação, [23] faz um estudo do processo de planejamento de release e propõe uma abordagem híbrida de planejamento que integra a inteligência computacional com o conhecimento e as experiências humanas, extendendo a formulação da pesquisa anterior, registrada em [24] para oferecer flexibilidade no número de releases a serem planejadas. A solução computacional adotada por eles é baseada em resolver uma sequência de problemas de Programação Linear sem condições inteiras, combinada com heurísticas para acelerar o processo e gerar mais de uma boa solução.

Em [28] o problema foi formulado de forma multiobjetiva, com o propósito de maximizar a satisfação do cliente e minimizar o custo de implementação dos requisitos. Para resolver o problema, os autores realizaram experimentos com algumas metaheurísticas multiobjetivas das quais o NSGA-II (Nondominated Sorting Genetic Algorithm II) [7] superou as outras técnicas em problemas de maior escala.

Utilizando uma ferramenta de otimização baseada em Programação Linear Inteira, [26] apresentou uma formulação matemática para a determinação do próximo release que assume que um conjunto ideal de requisitos é aquele que apresenta o máximo de receita projetada contra os recursos disponíveis. Foi desenvolvida e demonstrada uma técnica de otimização cuja entrada é dupla: o primeiro tipo de dados é a lista de requisitos, com as receitas previstas e os recursos necessários e o segundo se refere a mecanismos de direção gerenciais que permitem uma análise what-it [10], técnica para identificação de riscos baseada no levantamento de hipóteses, no ambiente de otimização.

Uma comparação das metaheurísticas NSGA II [7] e MOCell (Multiobjective Cellular Genetic Algorithm) [22] é apresentada por [12], levando em conta a versão multiobjetiva do NRP.

Foram utilizadas métricas de desempenho e as soluções obtidas são analisadas com base em indicadores de qualidade e na quantidade de soluções. Neste estudo, o algoritmo MOCell apresentou os melhores resultados.

Em [9], foi aplicada uma adaptação do algoritmo ACS (Ant Colony System) [11] ao Problema do Próximo Release, comparado com outras duas metaheurísticas que também já foram utilizadas para esse problema: Algoritmo Genético [14] e Simulated Annealing [27]. Os mesmos autores, em [8] compararam com o ACS, duas metaheurísticas bastante utilizadas para resolver o NRP, que são: GRASP [13] e novamente Algoritmo Genético. Este último apresentou os piores resultados, enquanto o GRASP trouxe as melhores soluções.

No artigo [4], é proposta uma solução para o problema em sua versão multiobjetiva, utilizando a abordagem evolucionária multiobjetiva NSGA II [7] e comparando-a com outros algoritmos evolucionários por meio de métricas de performance.

Capítulo 4

Estudo de caso

Esta seção tem por objetivo apresentar um caso real onde há a ocorrência de problemas com o planejamento e priorização de requisitos e a sua adaptação ao já conhecido na literatura, Problema do Próximo Release ou NRP (Next Release Problem)[3]. Algumas definições na modelagem matemática original do problema foram ajustadas para se adequar ao cenário estudado.

A partir desta adaptação, alguns algoritmos de otimização aplicados ao NRP puderam ser testados para este caso e comparados em termos de eficiência e eficácia na obtenção das soluções.

O estudo foi realizado com dados de uma empresa de Tecnologia da Informação de Recife, cujo nome foi eticamente preservado, recebendo a denominação fictícia de "Empresa X".

4.1 O problema na Empresa X

A Empresa X é uma empresa que atua no ramo de evolução e manutenção de software, tendo como principal característica a demanda contínua de solicitações por parte dos seus clientes para correções ou melhorias nos seus produtos de software.

O processo de desenvolvimento da empresa é iterativo e incremental, utilizando alguns conceitos da metodologia ágil Scrum [25]. Esta metodologia é baseada no conceito de "dividir para conquistar" e consiste na repetição de ciclos de PDCA (método iterativo de gestão

usada para melhoria contínua de processos e produtos cuja sigla vem do inglês: Plan - Do - Check - Act) [2] de curta duração, com o intuito de facilitar o gerenciamento do trabalho ao dividi-lo em partes menores, trazer respostas mais rápidas para o cliente em termos de incrementos de software funcionando, assim como receber frequentemente seu *feedback*, favorecendo a adaptação à mudanças.

Com este processo, surge a necessidade de distribuir entre as iterações os requisitos a serem desenvolvidos em cada uma delas, porém, essa distribuição não é algo trivial. O custo de desenvolvimento por iteração é limitado devido ao tempo de duração do ciclo, além de que a empresa possui vários clientes demandando simultaneamente, portanto, faz-se necessário priorizar e escolher da melhor forma, conjuntos de requisitos que caibam nessas limitações, desenvolvendo em uma certa ordem o necessário para manter os clientes satisfeitos.

Alguns termos utilizados no processo da empresa estão definidos abaixo pois estão relacionados às adaptações realizadas para a adequação do presente estudo de caso ao Problema do Próximo Release.

- **Backlog:** É o conjunto de todos os requisitos que a empresa possui para serem desenvolvidos.
- **Sprint:** É cada um dos ciclos nos quais o desenvolvimento total é subdividido. Uma sprint é tida como a unidade básica de desenvolvimento do Scrum, geralmente com duração entre uma e quatro semanas, cujo objetivo é dividir o desenvolvimento e gerenciamento total em partes menores (iterações) ao final das quais são gerados incrementos dos produtos.
- **Time:** Equipe de desenvolvimento.
- **Time-box da sprint:** É a limitação de tempo da sprint. Cada sprint deve começar e terminar dentro de um time-box. O time-box estabelece um padrão de desenvolvimento e serve para ajudar os gerentes a extrair medidas que podem auxiliá-los no monitoramento e controle do projeto, como por exemplo, a quantidade média de pontos que a equipe consegue desenvolver dentro de uma sprint (medida que será utilizada para definir a limitação de custo do Problema do Próximo Release).
- **Histórias:** São equivalentes aos requisitos.

- **Tamanho:** É o custo ou complexidade de desenvolvimento de cada requisito, cuja unidade básica é o ponto do Planning Poker.

- **Planning Poker:** Técnica de estimativa do tamanho dos requisitos na qual, em vez de realizar estimativas em horas, a equipe estima uma pontuação para cada requisito, que leva em consideração a complexidade de implementação e é baseada em uma referência. Por exemplo, a equipe decidiu que a referência para estimativa será a implementação de um cadastro, que possui tamanho de 2 pontos. Os outros requisitos são estimados com base nesta referência, para mais se forem julgados mais complexos ou para menos, caso contrário. Cada membro da equipe sugere uma pontuação para o requisito e caso haja divergências, a equipe discute até que todos cheguem em um consenso.

Com base nessas definições, temos as seguintes relações entre o problema original e a abordagem utilizada na Empresa X:

Problema original	**Estudo de caso**
Equipe de desenvolvimento	Time
Release	Sprint
Conjunto de requisitos	Backlog
Requisito	História
Custo de implementação do requisito	Tamanho do requisito
Limitação de custo	Limitação de pontos por sprint, dada pelo seu time-box

Tabela 4.1: Associação dos elementos do problema original aos do estudo de caso.

Dessa forma, adequando o estudo de caso ao Problema do Próximo Relese, temos:

- Um backlog de histórias a serem desenvolvidas pelo time.

- Um conjunto de clientes que possuem diferentes valores de importância para a organização. Esses valores são baseados no percentual de contribuição destes clientes no faturamento da empresa.

- Uma limitação de pontos a serem desenvolvidos dada pelo time-box da sprint.

- Cada história tem um tamanho medido em pontos do Planning Poker.

- Cada cliente possui um subconjunto de histórias.

Originalmente, a Empresa X não mapeia a dependência entre as histórias na sua ferramenta de gestão, de onde os dados para os testes foram extraídos, porém, isso foi simulado na forma de um grafo G, chamado de grafo de precedência, cujas arestas $E(G)$ representam as relações de precedência, semelhante à formulação original do problema, detalhada no Capítulo 2, e utilizado para fins de testes com alguns dos algoritmos (a precedência foi testada apenas com as metaheurísticas).

O critério de seleção das histórias para a sprint não tem por objetivo atender todos os requisitos de um cliente como demonstrado na formulação original do problema, já que, numa mesma sprint, vários clientes precisam ser atendidos ao mesmo tempo dentro de um time-box pequeno. Dessa forma, a priorização ocorre não em função do cliente, mas sim da importância dos requisitos, que agrega tanto o valor do cliente quanto a prioridade que o cliente dá àquele requisito.

As demandas dos clientes da empresa também são independentes umas das outras, ou seja, não existem requisitos solicitados por mais de um cliente ao mesmo tempo, porém esta dependência foi simulada de maneira aleatória e tratada na forma de uma matriz de importância das histórias, que contempla a importância dada para cada história por cada cliente. Esta estratégia foi utilizada em [9] e pode ser definida da seguinte maneira:

Como uma história pode ser demandada por vários clientes ao mesmo tempo, a importância dessa história varia de cliente para cliente. Então, a importância de uma história $r_j \in R$, representa para o cliente i um dado valor v_{ij}. O maior valor v_{ij} representa a história de maior prioridade para o cliente i. Um valor $v_{ij} = 0$ significa que o cliente i não solicitou a história r_j. Todos esses valores v_{ij} de importância podem ser organizados sob a forma de uma matriz $m \times n$, onde m é a quantidade de linhas da matriz, que coincide com a quantidade de clientes e n é a quantidade de colunas, compatível com a quantidade de requisitos a serem desenvolvidos.

$$\begin{bmatrix} v_{11} & \dots & v_{1n} \\ \dots & \dots & \dots \\ v_{m1} & \dots & v_{mn} \end{bmatrix}_{m \times n}$$

A satisfação global s_j ou a adição de valor dada pela inclusão da história r_j na sprint é medida pela a soma ponderada do valores de importância atribuídos a história por todos os clientes e pode ser formalizado como

$$s_j = \sum_{i}^{m} c_i v_{ij}$$

Tendo em vista que neste caso o objetivo não é em função dos clientes, e sim da história, não precisa haver uma garantia de que todas as histórias de um cliente i devem ser implementadas. Para alguns testes em que não foi considerada a dependência entre as histórias, o problema pode ser formulado como o caso básico do Problema da Mochila, onde temos:

- Variável de decisão

  $$x_j = \begin{cases} 1 & \text{se o requisito j é implementado} \\ 0 & \text{caso contrário} \end{cases}$$

- Função objetivo

  $$max \sum_{j \in R} s_j x_j$$

- Restrição

  $$\sum_{j \in R} t_j x_j \leq T$$

onde s_j é a satisfação dada pela inclusão da história r_j na sprint, t_j é o tamanho da história r_j e T é a limitação dada pelo time-box da sprint.

Para os casos em que a precedência foi considerada, foi adicionada ao problema a seguinte condição nas suas restrições:

$$x_r \geq x_{r'}, \forall (r, r') \in E(G),$$

onde r e r' são duas histórias pertencentes a R, tais que r é pré-requisito de r', G é o grafo de precedência e $E(G)$ é o seu conjunto de arestas que representam as relações de precedência.

4.2 Implementação dos algoritmos e detalhamento das instâncias e dos testes

Para a implementação dos algoritmos foram utilizados códigos em Python e em Java disponíveis na internet e foram feitas algumas adaptações para adequá-los ao problema. Em

todos os algoritmos foram feitas adaptações para a entrada e saída de dados e também para tratar a matriz de importância dos requisitos dada pelos diferentes clientes.

Os algoritmos de Busca Exaustiva, Programação Dinâmica, Busca Gulosa e Branch and Bound estão disponíveis no mesmo repositório[1] . Nestes quatro casos, os algoritmos já estão implementados para o problema da mochila e só foram necessárias as adaptações gerais já citadas no parágrafo anterior.

Os algoritmos Simulated Annealing e Algoritmo Genético estão disponíveis em outro repositório[2]. Eles estão implementados de forma genérica e precisaram ser adaptados ao problema:

- No Algoritmo Genético estava especificado o algoritmo de seleção e os critérios de elitismo, além de uma classe básica para os indivíduos, que pôde ser modelada para o caso específico. Os detalhes da criação da população inicial de cromossomos precisou ser especificada para que a geração de cromossomos ficasse um pouco mais restrita, por meio de uma validação em relação à restrição de limite de custo de implementação dos requisitos. A função de fitness também foi modelada de acordo com o problema, sendo calculada com base no valor do cromossomo, incluindo uma punição para os cromossomos que não respeitassem as restrições do problema, tanto em relação à limitação dos custos de implementação dos requisitos quanto à sua relação de precedência.

- No Simulated Annealing estava projetado um esboço da função principal do algoritmo e as funções específicas para o problema foram especificadas numa classe "Solução", onde foram modeladas as funções para calcular o valor e o peso da solução. Além disso, foi inclusa na função principal a validação relacionada às restrições de limitação de custo de implementação dos requisitos e de precedência.

A validação dos algoritmos foi realizada por meio da aplicação destes em uma instância de um caso real já utilizado anteriormente por [9] e [15]. Esta instância trata-se de uma amostra de um projeto com 20 requisitos e 5 clientes, cuja limitação de custo de desenvolvimento é de 25. Cada requisito tem um custo associado que pode ser observado na Tabela 4.2. Cada cliente atribuiu uma prioridade para cada requisito utilizando um valor de importância que varia de 1 a 5. A matriz de importância dos requisitos para os clientes é mostrada na Tabela

[1] https://github.com/patrickherrmann/Knapsack/

[2] https://github.com/mlarocca/Algorithms

4.3. Por fim, os pesos dos clientes foram medidos sob o ponto de vista do gerente do projeto, que atribuiu um valor de importância que variou de 1 a 5 para cada cliente. Estes valores estão disponibilizados na Tabela 4.4.

Requisitos	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
Custos	1	4	2	3	4	7	10	2	1	3	2	5	8	2	1	4	10	4	8	4

Tabela 4.2: Custos dos requisitos do caso de validação.

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
c_1	4	2	1	2	5	5	2	4	4	4	2	3	4	2	4	4	4	1	3	2
c_2	4	4	2	2	4	5	1	4	4	5	2	3	2	4	4	2	3	2	3	1
c_3	5	3	3	3	4	5	2	4	4	4	2	4	1	5	4	1	2	3	3	2
c_4	4	5	2	3	3	4	2	4	2	3	5	2	3	2	4	3	5	4	3	2
c_5	5	4	2	4	5	4	2	4	5	2	4	5	3	4	4	1	1	2	4	1

Tabela 4.3: Matriz de importância dos requisitos para os clientes do caso de validação.

Clientes	c_1	c_2	c_3	c_4	c_5
Pesos	4	4	3	5	5

Tabela 4.4: Pesos de importância dos clientes do caso de validação.

Os testes foram realizados por meio da aplicação dos algoritmos em instâncias geradas aleatoriamente e na instância do estudo de caso da Empresa X. Os resultados são apresentados na Seção 4.3.

No estudo de caso, foi selecionada uma amostra do backlog da Empresa X, contendo 64 histórias e 4 clientes. O limite de pontos a serem trabalhados na sprint é de 82 e foi determinado pelo time-box da sprint que tem duração de duas semanas. O custo associado a cada requisito, que foi determinado pelo time utilizando pontos do Planning Poker, está disponível na Tabela 4.5. Como a Empresa X não possui dependência nas demandas dos seus clientes, foi gerada uma instância aleatória da matriz de importância dos requisitos para os clientes com valores de importância variando de 0 a 20. Como a Empresa X também não mapeia a precedência entre os requisitos, foi gerada uma instância aleatória do grafo de precedência contendo 15 arestas que indicam 15 relações de precedência. O peso dos clientes foi calculado

com base no percentual de contribuição do cliente para o faturamento da empresa e pode ser conferido na Tabela 4.6.

Requisitos	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}
Custos	2	4	2	2	4	4	2	16	6	6	4	4	2	6	6	2

Requisitos	r_{17}	r_{18}	r_{19}	r_{20}	r_{21}	r_{22}	r_{23}	r_{24}	r_{25}	r_{26}	r_{27}	r_{28}	r_{29}	r_{30}	r_{31}	r_{32}
Custos	10	10	6	10	16	4	4	4	2	6	6	6	6	2	6	10

Requisitos	r_{33}	r_{34}	r_{35}	r_{36}	r_{37}	r_{38}	r_{39}	r_{40}	r_{41}	r_{42}	r_{43}	r_{44}	r_{45}	r_{46}	r_{47}	r_{48}
Custos	6	1	4	6	2	1	1	6	4	4	2	16	6	10	2	1

Requisitos	r_{49}	r_{50}	r_{51}	r_{52}	r_{53}	r_{54}	r_{55}	r_{56}	r_{57}	r_{58}	r_{59}	r_{60}	r_{61}	r_{62}	r_{63}	r_{64}
Custos	10	16	1	2	2	6	16	4	16	2	4	4	4	16	10	6

Tabela 4.5: Custos dos requisitos da Empresa X.

Clientes	c_1	c_2	c_3	c_4
Pesos	50	15	15	20

Tabela 4.6: Pesos de importância dos clientes da Empresa X.

As instâncias aleatórias foram geradas por meio de uma função auxiliar, cuja entrada é a quantidade de requisitos N e a saída é um conjunto de requisitos representados por uma lista de N tuplas de dois elementos: o tamanho ou custo de implementação do requisito e o valor do requisito.

Cada algoritmo foi testado e validado utilizando as instâncias explicitadas acima para que as comparações pudessem ser realizadas. De acordo com a abordagem à qual o algoritmo pertence, diferentes análises foram realizadas.

No caso dos algoritmos exatos (Busca exaustiva, Programação Dinâmica e Branch and Bound), os testes foram realizados medindo o tempo que eles gastaram para chegar na solução ótima, executando por 20 vezes independentes os algoritmos e calculando a média dos tempos. Também, por meio de instâncias aleatórias, foi testada a viabilidade do uso destes algoritmos de acordo com o aumento do tempo de execução à medida que o tamanho

das instâncias era ampliado. Para este caso, os algoritmos também foram executados por 20 vezes e o limite de custo era sempre de 25% do custo total dos requisitos.

Na heurística Algoritmo Guloso foi avaliado o tempo de execução e a proximidade das suas soluções com a solução ótima. O algoritmo foi executado por 100 vezes para cada instância e as médias dos tempos de execução foram calculadas e registradas.

Já nas metaheurísticas Simulated annealing e Algoritmo genético, alguns parâmetros destes algoritmos foram configurados. Para cada configuração, os algoritmos foram executados por 100 vezes e foram calculadas as médias da satisfação do cliente e a sua proximidade com a solução ótima, custo de implementação e tempo de execução das soluções como medidas de tendência e seus desvios-padrão como medidas de dispersão. Estes algoritmos também foram testados com instâncias aleatórias maiores para que sua utilização fosse comparada com a dos outros algoritmos em termos de tamanho da instância.

Os parâmetros cujas configurações foram testadas no Simulated Annealing foram a quantidade de vezes em que era feito o resfriamento (CS), a fração de quanto a temperatura era resfriada por iteração (CF) e a quantidade de ciclos de perturbação ou mutação aos quais as soluções eram submetidas em cada iteração de resfriamento (ST). No Algoritmo Genético, os parâmetros configurados para os testes foram o tamanho da população inicial (P), a probabilidade de mutação (TM) e a probabilidade de recombinação (TC). Foram criadas funções auxiliares para facilitar os cálculos dos resultados das médias e desvios padrão.

4.3 Testes e comparação entre as metodologias de solução

4.3.1 Validação

Inicialmente, todos os algoritmos foram aplicados à instância de validação, a fim de averiguar se eles estavam funcionando como esperado. Apenas o algoritmo de Busca Sequencial não pôde ser validado pois ocorreu estouro de memória na sua execução. Os resultados podem ser analisados nas Tabelas 4.7, 4.8 e 4.9.

Algoritmos	Média de tempo de execução (ms)	Valor da melhor solução	Custo de implementação da melhor solução	% de proximidade da solução ótima
Programação dinâmica	3,097979	815	25	100%
Branch and Bound	1,624486	815	25	100%
Busca Gulosa	0,541436	815	25	100%

Tabela 4.7: Algoritmos exatos e Guloso na instância de validação.

		Satisfação		Custo de implementação		Tempo de execução (s)		
Parâmetros	Min-Max	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	% de proximidade da solução ótima
CS = 10; CF = 0,995; ST = 1000	669 - 785	726,94	22,71291263	24,28	0,825590698	1,524347437	0,611760519	89%
CS = 10; CF = 0,9995; ST = 1000	682 - 815	729,61	23,86247892	24,49	0,768049478	1,216965044	0,639801817	89%
CS = 100; CF = 0,95; ST = 1	213 - 760	496,88	115,2387331	22,99	2,076029865	0,030938142	0,008194116	61%
CS = 100; CF = 0,995; ST = 1	263 - 740	505,44	95,96408912	23,13	2,105492816	0,030349367	0,006157983	62%
CS = 100; CF = 0,995; ST = 10	475 - 747	656,87	40,71084745	23,8	1,319090596	0,235699807	0,054030958	81%
CS = 100; CF = 0,995; ST = 100	690 - 815	730,12	25,02010392	24,14	1,131547613	0,856236242	0,420321611	90%
CS = 100; CF = 0,995; ST = 1000	753 - 815	772,8	16,98116604	24,6	0,489897949	8,979994732	3,325294274	95%
CS = 100; CF = 0,9995; ST = 10	509 - 762	654,74	46,36477542	23,75	1,306713435	0,240104818	0,037531114	80%
CS = 100; CF = 0,9995; ST = 100	680 - 815	732,28	25,98271733	24,4	0,824621125	1,88409497	0,441041574	90%
CS = 100; CF = 0,9995; ST = 1000	731 - 815	773,74	17,03738243	24,73	0,63015871	12,7271239	5,65228793	95%
CS = 1000; CF = 0,9995; ST = 1	537 - 774	659,01	51,0966721	23,93	1,394668419	0,159939857	0,048637044	81%
CS = 1000; CF = 0,99995; ST = 1	549 - 815	664,57	45,11457747	24,07	1,032036821	0,24365709	0,032788644	82%
CS = 1000; CF = 0,999995; ST = 10	674 - 778	728,05	25,97590229	24,4	0,774596669	1,720422105	0,628274645	89%

Tabela 4.8: Aplicação do algoritmo Simulated Annealing na instância de validação.

		Satisfação		Custo de implementação		Tempo de execução (s)		
Parâmetros	Min-Max	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	% de proximidade da solução ótima
P=20;TM=0,2;TC=0,8	615 - 815	706,18	45,2776722	24,31	0,924067097	0,150058417	0,01863384	87%
P=100;TM=0,2;TC=0,8	731 - 815	784,29	25,4437792	24,72	0,633719181	1,170612066	0,536773674	96%
P=20;TM=0,02;TC=0,8	554 - 815	687,71	52,67319907	24,06	1,066020638	0,158907606	0,020921489	84%
P=100;TM=0,02;TC=0,8	707 - 815	779,46	27,4537502	24,68	0,785875308	2,112554624	0,36912058	96%
P=20;TM=0,2;TC=0,9	574 - 815	719,71	51,04924975	24,38	0,956869897	0,145481614	0,034996996	88%
P=100;TM=0,2;TC=0,9	714 - 815	786,29	26,43607195	24,62	0,78460181	1,506911911	0,52543072	96%
P=100;TM=0,02;TC=0,9	724 - 815	782,83	25,96384217	24,71	0,682568678	1,686554231	0,652030719	96%

Tabela 4.9: Aplicação do Algoritmo Genético na instância de validação.

Na Tabela 4.7 encontram-se os resultados dos testes realizados para a intância de validação com os algoritmos exatos e o Algoritmo Guloso. Foi considerada a média do tempo de execução do algoritmo para retornar a solução, o valor e o custo encontrado por eles na melhor solução e a proximidade com a solução ótima que em todos os casos foi de 100%.

Nas Tabelas 4.8 e 4.9 foram apresentados os resultados dos testes das metaheurísticas Simulated Annealing e Algoritmo Genético, respectivamente, para a instância de validação. Os significados dos parâmetros que foram configurados podem ser conferidos na Seção 4.2. Nas tabelas foram registrados os valores de satisfação mínimo e máximo encontrados, cal-

culadas as médias e desvios-padrão destes valores, dos custos e dos tempos de execução e a proximidade da média dos valores com a solução ótima.

Entre os algoritmos exatos, o Branch and Bound teve o melhor desempenho em tempo de execução. Entre as heurísticas e metaheurísticas, o algoritmo guloso alcançou a melhor solução em um tempo de execução menor.

4.3.2 Testes na instância da Empresa X

Em seguida, os algoritmos foram aplicados ao problema da Empresa X, como mostrado nas Tabelas 4.10, 4.11 e 4.12. O algoritmo de Busca Sequencial tornou-se inviável visto que a instância do problema era ainda maior que o do caso de validação, resultando também em estouro de memória.

Algoritmos	**Média de tempo de execução (ms)**	**Valor da melhor solução**	**Custo de satisfação da melhor solução**	**% de proximidade da solução ótima**
Programação dinâmica	49,603508	19355	81	100%
Branch and Bound	17,595416	19355	81	100%
Busca Gulosa	1,529982	19355	81	100%

Tabela 4.10: Algoritmos exatos e Guloso na instância da Empresa X.

		Satisfação		Custo de implementação		Tempo de execução (s)		
Parâmetros	Min-Max	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	% de proximidade da solução ótima
CS = 10; CF = 0,995; ST = 1000	4430 - 13830	9078,15	2114,365349	77,14	4,688325927	2,539633536	0,872882483	47%
CS = 10; CF = 0,9995; ST = 1000	4845 - 13200	9564,45	1877,666091	79,23	2,982800697	1,763256883	0,839182877	49%
CS = 100; CF = 0,995; ST = 100	4190 - 14370	9182,4	1869,613928	78,84	3,480574665	1,849811173	0,905584979	47%
CS = 100; CF = 0,9995; ST = 100	4510 - 14240	9111,4	2057,118869	78,81	3,684277405	1,827490444	0,901178411	47%
CS = 1000; CF = 0,999995; ST = 10	4740 - 15240	9496,25	2197,708395	79,26	3,276034188	1,364155564	0,595892737	49%

Tabela 4.11: Algoritmo Simulated Annealing na instância da Empresa X.

		Satisfação		Custo de implementação		Tempo de execução (s)		
Parâmetros	Min-Max	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	% de proximidade da solução ótima
P=100;TM=0,2;TC=0,8	12155 - 16180	13604,3	737,5876287	80,15	2,05121915	3,82846161	1,196907218	70%
P=100;TM=0,02;TC=0,8	11905 - 15365	13297,8	753,0874186	79,82	2,178898804	3,549345206	1,410722507	69%
P=100;TM=0,2;TC=0,9	11840 - 15585	13686,45	680,4936793	79,72	2,738904891	3,220264113	1,363603135	71%
P=100;TM=0,02;TC=0,9	11670 - 15430	13290,6	771,6787803	80,36	2,220450405	3,751552908	1,358033027	69%

Tabela 4.12: Algoritmo Genético na instância da Empresa X.

Na Tabela 4.10 encontram-se os resultados dos testes realizados para a intância da Empresa X com os algoritmos exatos e o Algoritmo Guloso. Foi considerada a média do tempo de

execução do algoritmo para retornar a solução, o valor e o custo encontrado por eles na melhor solução e a proximidade com a solução ótima que em todos os casos foi de 100%.

Nas Tabelas 4.11 e 4.12 foram coletados os resultados dos testes realizados para a instância da Empresa X com as metaheurísticas. Para estes casos foram registrados os valores de satisfação mínimo e máximo encontrados, calculadas as médias e desvios-padrão dos valores encontrados, dos custos e dos tempos de execução e a proximidade da média dos valores com a solução ótima. Os significados dos parâmetros que foram configurados podem ser conferidos na Seção 4.2.

Da mesma forma que na validação, o algoritmo Branch and Bound teve o melhor desempenho em tempo de execução entre os algoritmos exatos e entre as heurísticas e metaheurísticas, o Algoritmo Guloso também alcançou a melhor solução em um tempo de execução menor.

Testes das metaheurísticas considerando a precedência entre requisitos

Para a instância da Empresa X também foram realizados testes simulando a precedência entre as histórias por meio da geração aleatória de um grafo de precedência contendo 15 arestas (ou 15 relações de dependência). Os testes foram realizados apenas com as metaheurísticas e os resultados estão apresentados nas Tabelas 4.13 e 1.14.

		Satisfação		Custo de implementação		Tempo de execução (s)	
Parâmetros	Min-Max	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão
CS = 10; CF = 0,995; ST = 1000	205 - 11970	3337,96	2610,58486799	34,53	24,9836899045	1,7074933749	0,721936003901
CS = 10; CF = 0,9995; ST = 1000	265 - 10055	3172,18	2510,486936	31,30	24,9841857764	2,18577974988	1,11700698908
CS = 100; CF = 0,995; ST = 100	240 - 10400	2901,06	2155,72863267	27,78	21,0279234356	1,51961532743	0,723163027377
CS = 100; CF = 0,9995; ST = 100	205 - 9425	3407.70	2592,24708996	34,29	27,1165616491	1,65965708068	0,859317825528
CS = 1000; CF = 0,999995; ST = 10	260 - 10850	3198,27	2472,34720789	31,73	21,8400540098	1,10834119361	0,595892737

Tabela 4.13: SA na instância da Empresa X considerando precedência.

		Satisfação		Custo de implementação		Tempo de execução (s)	
Parâmetros	Min-Max	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão
P = 100; TM = 0,2; TC = 0,8	7595 - 12550	10048,0	972,388811124	79,95	2,99791594278	5,86909248177	2,09561965155
P = 100; TM = 0,02; TC = 0,8	6930 - 12750	9562,95	1223,17151189	79,71	3,83221867852	8,17464320451	3,56899881673
P = 100; TM = 0,2; TC = 0,9	8020 - 13435	10036,6	1046,06617382	79,6	2.66082693913	7,82424575514	0,723163027377
P = 100; TM = 0,02; TC = 0,9	7700 - 12290	9446,9	965,987003018	80,36	2,42289083535	11,875450517	4,56508204763

Tabela 4.14: AG na instância da Empresa X considerando precedência.

A metaheurística Algoritmo Genético obteve os melhores resultados em relação ao valor de

satisfação, porém com um tempo de execução maior que o Simulated Annealing. Como não se tem o conhecimento do valor da melhor solução para este caso de teste, não foi possível avaliar a proximidade dos algoritmos com a solução ótima. Os significados dos parâmetros que foram configurados podem ser conferidos na Seção 4.2.

4.3.3 Testes nas instâncias aleatórias

Por fim, foi realizado um teste do comportamento dos algoritmos de acordo com o aumento do tamanho das instâncias de entrada. Este aumento de tamanho se refere ao número de requisitos da empresa. O número de clientes foi igual para todas as instâncias (5 clientes). Os resultados desses testes encontram-se nas Tabelas 4.15, 4.16, 4.17, 4.18, 4.19, 4.20.

	Média dos tempos de execução (s)			
Número de requisitos	**Busca sequencial**	**Programação dinâmica**	**Branch and bound**	**Busca gulosa**
5	0,001293422	0,000365763	0,000805335	0,0003994
10	0,012245658	0,000709951	0,001010066	0,000464339
15	1,367207599	0,000920098	0,001608789	0,000431645
20	Estouro de memória	0,001116684	0,002904492	0,000453359
25		0,002453767	0,002975852	0,000679005
50		0,014807087	0,013215655	0,001349953
100		0,072261792	0,03816987	0,002834659
250		0,632207801	0,181860242	0,006521254
500		4,700317047	0,866298041	0,018318711
1000		Estouro de memória	2,438167956	0,026372155
2000			23,25836356	0,068006023
5000			Estouro de memória	0,165160334
10000				0,251540383
30000				0,600513823
50000				1,313385231
100000				4,040104321

Tabela 4.15: Algoritmos exatos e Guloso - tamanho das instâncias.

Número de requisitos	**% de proximidade da solução ótima**
5	100,00%
10	74,23%
15	97,18%
20	96,90%
25	100,00%
50	95,67%
100	99,20%
250	99,58%
500	99,63%
1000	99,94%
2000	100,00%

Tabela 4.16: Busca Gulosa - aproximação da solução ótima.

Parâmetros	Min-Max	Satisfação		Custo de implementação		Tempo de execução (s)		
		Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	% de proximidade da solução ótima
CS = 10; CF = 0,995; ST = 1000	399 - 936	672,03	89,91845806	227,43	7,613481464	2,430317207	1,283451541	45%
CS = 100; CF = 0,995; ST = 100	451 - 976	674,23	97,6362489	227,74	5,830300164	2,497152987	1,38568688	45%
CS = 1000; CF = 0,999995; ST = 10	428 - 880	667,31	105,1705943	228,32	5,291275839	3,143098838	1,212951479	44%

Tabela 4.17: Simulated Annealing na instância da tamanho 100.

Parâmetros	Min-Max	Satisfação		Custo de implementação		Tempo de execução(s)		
		Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	% de proximidade da solução ótima
CS = 5; CF = 0,995; ST = 100	5149 - 6117	5830	191,9838535	2240,5	109,9365726	4,981979607	0,721114727	20%
CS = 10; CF = 0,995; ST = 50	5149 - 6148	5850,4	58,56960378	2291,8	62,43364478	4,994634761	0,997182144	20%
CS = 5; CF = 0,99995; ST = 100	5432 - 6176	5856,7	195,2496095	2314,6	59,67277436	5,369093022	0,642335661	20%

Tabela 4.18: Simulated Annealing na instância da tamanho 2000.

Parâmetros	Min-Max	Satisfação		Custo de implementação		Tempo de execução (s)		
		Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	% de proximidade da solução ótima
P=100;TM=0,2;TC=0,8	777 - 934	843,95	37,74052861	227,55	6,19253583	7,124767685	0,419269376	56%
P=100;TM=0,02;TC=0,8	677 - 907	804,25	53,21923994	228,3	5,348831648	6,574364489	1,755607996	53%
P=100;TM=0,2;TC=0,9	754 - 931	834,95	45,92001198	226,6	8,230431337	6,005117248	0,841207824	55%

Tabela 4.19: Algoritmo Genético na instância da tamanho 100.

Parâmetros	Min-Max	Satisfação		Custo de implementação		Tempo de execução (s)		
		Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	Média Aritmética	Desvio Padrão	% de proximidade da solução ótima
P=100;TM=0,2;TC=0,8	10610 - 10912	10708,5	97,00644309	4874,1	34,07770532	59,75299591	9,703656423	37%
P=50;TM=0,2;TC=0,8	10202 - 10759	10512,2	164,8240274	4867	25,93838854	18,57190332	2,878528972	37%
P=20;TM=0,2;TC=0,8	10080 - 11071	10541,8	240,2027477	4873,1	36,16199663	4,478644772	0,251492719	37%
P=20;TM=0,02;TC=0,9	10125 - 10922	10510,9	209,4289617	4885,5	21,72671167	4,453224543	0,416855942	37%

Tabela 4.20: Algoritmo Genético na instância da tamanho 2000.

Na Tabela 4.15, são mostrados os resultados dos tempos de execução dos algoritmos exatos e do algoritmo de busca gulosa a partir do aumento do tamanho das instâncias. Na Tabela 4.16 é apresentada a proximidade das soluções encontradas pelo Algoritmo Guloso em relação à solução ótima.

Nas Tabelas 4.17 e 4.18 são apresentados os resultados das metaheurísticas aplicadas à instância de tamanho 100 e nas tabelas 4.19 e 4.20 são mostrados os resultados para a instância de tamanho 2000. Da mesma forma, foram registrados os valores de satisfação mínimo e máximo encontrados, calculadas as médias e desvios-padrão dos valores encontrados, dos custos e dos tempos de execução e a proximidade da média dos valores com a

solução ótima. Os significados dos parâmetros que foram configurados podem ser conferidos na Seção 4.2.

Com o aumento das instâncias foi possível notar que os algoritmos exatos vão se tornando inviáveis em tempo de execução. O Algoritmo Guloso apresentou resultados favoráveis nas instâncias testadas (até 2000 requisitos), tanto em tempo de execução quanto em qualidade das soluções, porém, foi notado que quanto mais esse tamanho aumenta, seu tempo também tende a aumentar. As metaheurísticas não apresentaram valores favoráveis em termos de qualidade das soluções mas na literatura é possível encontrar maneiras de melhorar esses resultados adaptando a implementação desses algoritmos. Pode-se utilizar, por exemplo, algumas técnicas como a hibridização. Isso torna essas soluções, ótimas candidatas a serem aplicadas ao problema.

Capítulo 5

Conclusão e trabalhos futuros

Este trabalho teve como objetivo estudar um problema da Engenharia de Software conhecido como Problema do Próximo Release. Ele foi modelado como um problema de otimização e diversos algoritmos de solução foram aplicados para resolvê-lo.

Devido à quantidade de fatores que influenciam a decisão do conjunto de requisitos ideal para compor o próximo release, este processo apresenta uma complexidade inerente que o torna adequado para resolução por meio de técnicas automatizadas na tentativa de se alcançar bons resultados. Foram aplicados algoritmos tanto de abordagem exata como heurísticas e metaheurísticas para buscar soluções para o problema. Dessa forma, o problema foi modelado como um problema de otimização a partir de uma formulação mono-objetiva proposta por [3] e foi resolvido com os algoritmos exatos de Busca Exaustiva, Programação Dinâmica e Branch and Bound, a heurística de Busca Gulosa e as metaheurísticas Simulated Annealing e Algoritmo Genético.

Alguns algoritmos exatos apresentaram soluções viáveis para as instâncias menores, que consideravam um número de requisitos pequeno, porém, à medida que esse número aumenta, estas soluções podem tornar-se extremamente lentas inviabilizando seu uso. As heurísticas e metaheurísticas podem ser uma saída para esses casos. Embora não é garantido que elas retornem a solução ótima, elas usam técnicas que tendem a aproximar seus resultados do valor ótimo. Ao realizar testes com instâncias maiores, os resultados mais satisfatórios foram obtidos com a heurística de busca gulosa, porém, é notado que seu tempo de execução também tende a aumentar, além de que, esta técnica não usa nenhuma estratégia de melhoria dos seus resultados, podendo retornar soluções desfavoráveis sem ter a possibilidade de

aperfeiçoá-las. As metaheurísticas Algoritmo Genético e Simulated Annealing apresentaram resultados insatisfatórios neste trabaho mas é possível identificar na literatura que melhorias podem ser aplicadas a esses algoritmos permitindo que eles possam ser ótimas sugestões para serem aplicadas ao problema.

Devido às adaptações que precisaram ser realizadas na instância da Empresa X, que resultaram na geração aleatória de algumas informações como a dependência entre os requisitos e a importância dos requisitos dada pelos clientes, não foi possível validar de maneira satisfatória os resultados deste trabalho com os resultados reais. Além disso, foi possível compreender que outras restrições e características intrínsecas ao gerenciamento do desenvolvimento de software como cronograma e custos, entre outros, precisam ser considerados para que se torne viável alcançar resultados mais próximos à realidade da empresa. A Empresa X, por ter uma demanda relativamente pequena, consegue realizar a seleção dos seus requisitos manualmente de maneira aceitável, porém, é notável que para situações em que o problema é maior, a aplicação da otimização pode ser uma alternativa bastante promissora se modelada de forma a adequar-se às necessidades da organização.

Trabalhos futuros incluem:

- Aplicação de outros algoritmos de otimização que não foram contemplados neste trabalho;
- Melhorias nas implementações das metaheurísticas utilizadas, com o intuito de obter resultados mais satisfatórios;
- Considerar a dependência entre requisitos também para os outros algoritmos a fim de validar os resultados encontrados;
- Adicionar ao problema outras restrições encontradas pelas empresas com o propósito de aproximar-se cada vez mais de cenários reais e tornar possível considerar o uso destas soluções num ambiente real;
- Estudar outros problemas da área de Engenharia de Software que podem ser modelados como problemas de otimização.

Referências Bibliográficas

[1] Metaheuristics network. `http://http://www.metaheuristics.net/`. Acessado em 18/01/2015.

[2] Fabio Felippe de Andrade. *O método de melhorias PDCA.* PhD thesis, Universidade de São Paulo, 2003.

[3] Anthony J. Bagnall, Victor J. Rayward-Smith, and Ian M Whittley. The next release problem. *Information and software technology*, 43(14):883–890, 2001.

[4] Xinye Cai, Ou Wei, and Zhiqiu Huang. Evolutionary approaches for multi-objective next release problem. *Computing and Informatics*, 31(4):847–875, 2012.

[5] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

[6] George B Dantzig. Discrete-variable extremum problems. *Operations research*, 5(2):266–288, 1957.

[7] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[8] José del Sagrado, Isabel M ÁAguila, and Francisco J Orellana. Requirements interaction in the next release problem. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 241–242. ACM, 2011.

[9] Jose del Sagrado, Isabel Maria del Aguila, and Francisco Javier Orellana. Ant colony optimization for the next release problem: A comparative study. In *Search Based Software Engineering (SSBSE), 2010 Second International Symposium on*, pages 67–76. IEEE, 2010.

[10] William W Doerr. What–if analysis. *Risk assessment and risk management for the chemical process industry*, 1991.

[11] Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[12] Juan J Durillo, YuanYuan Zhang, Enrique Alba, and Antonio J Nebro. A study of the multi-objective next release problem. In *Search Based Software Engineering, 2009 1st International Symposium on*, pages 49–58. IEEE, 2009.

[13] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[14] David E Goldberg. *Genetic algorithms*. Pearson Education India, 2006.

[15] Des Greer and Günther Ruhe. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, 46(4):243–253, 2004.

[16] Mark Harman and Bryan F Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833–839, 2001.

[17] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. *Department of Computer Science, Kings College London, Tech. Rep. TR-09-03*, 2009.

[18] Maya Hristakeva and Dipti Shrestha. Different approaches to solve the 0/1 knapsack problem.

[19] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simmulated annealing. *science*, 220(4598):671–680, 1983.

[20] Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[21] Gabriela Rosa Machado Linhares, FG Freitas, RAF Carmo, CL Maia, and JT Souza. Aplicaçao do algoritmo grasp reativo para o problema do próximo release. *XLII Simpósio Brasileiro de Pesquisa Operacional (SBPO)*, 2010.

[22] Antonio J Nebro, Juan J Durillo, Francisco Luna, Bernabé Dorronsoro, and Enrique Alba. Mocell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, 24(7):726–746, 2009.

[23] Gunther Ruhe and Moshood Omolade Saliu. The art and science of software release planning. *Software, IEEE*, 22(6):47–53, 2005.

[24] Günther Ruhe and An Ngo The. Hybrid intelligence in software release planning. *International Journal of Hybrid Intelligent Systems*, 1(1):99–110, 2004.

[25] Ken Schwaber and Jeff Sutherland. The scrum guide. *Scrum. org, October*, 2011.

[26] Marjan van den Akker, Sjaak Brinkkemper, Guido Diepen, and Johan Versendaal. Software product release planning through optimization and what-if analysis. *Information and Software Technology*, 50(1):101–111, 2008.

[27] Peter JM Van Laarhoven and Emile HL Aarts. *Simulated annealing*. Springer, 1987.

[28] Yuanyuan Zhang, Mark Harman, and S Afshin Mansouri. The multi-objective next release problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1129–1137. ACM, 2007.