CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Diretoria de Pesquisa e Pós-Graduação

Programa de Pós-Graduação em Modelagem Matemática e Computacional

Desenvolvimento de Cronogramas de Projetos de Software Utilizando Otimização Heurística

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional, como parte dos requisitos exigidos para a obtenção do título de Mestre em Modelagem Matemática e Computacional.

Aluno : Sophia Nóbrega

Orientador : Prof. Dr. Sérgio Ricardo de Souza

Co-Orientador : Prof. Dr. Marcone Jamilson Freitas Souza

Belo Horizonte - MG
Agosto de 2013

Nóbrega, Sophia
M931a DESENVOLVIMENTO DE CRONOGRAMAS DE PROJETOS DE SOFTWARE UTILIZANDO OTIMIZAÇÃO HEURÍSTICA / Sophia Nóbrega. – Belo Horizonte, 2013.
91p.

Dissertação (Mestrado) – Centro Federal de Educação Tecnológica de Minas Gerais
Programa de Pós-Graduação em Modelagem Matemática e Computacional
Orientador: Prof. Dr. Sérgio Ricardo de Souza
Co-orientador: Prof. Dr. Marcone Jamilson Freitas Souza

1. Pesquisa Operacional - Teses. 2. MOVNS - Teses. 3. Metaheurísticas. 4. Computação.
I. Souza, Sérgio Ricardo de. II. Souza, Marcone Jamilson Freitas.
III. Centro Federal de Educação Tecnológica de Minas Gerais.
IV. Título.

CDD: 006.3

Dedico aos meus pais e aos meus irmãos, que são as pessoas mais importantes na minha vida:
Jorge, Shirley, Levi e Diana.

Agradecimentos

A conclusão deste trabalho representa uma grande conquista para mim, e foi possível apenas com a ajuda daqueles que fizeram parte de cada momento que vivi nos últimos anos. Por isso, agradeço de coração a todos vocês.

Agradeço a Deus, pelas bençãos concedidas e por ter me guardado até aqui.

Aos meus pais Jorge e Shirley pelo amor, compreensão e confiança dedicados à mim. Aos meus irmãos, Levi e Diana, pelos momentos de apoio, compreensão e diversão. Agradeço a toda a minha família pelo apoio e o incentivo durante todo o meu trabalho, não me deixando desistir mesmo com todos os problemas que vivi durante o curso.

Aos professores Marcone e Sérgio pela excelente orientação para o desenvolvimento deste trabalho, pela paciência, compreensão, confiança, amizade, incentivo, e principalmente pelos ensinamentos de vida.

Aos amigos Carolina, Francielly, Giseli e Taisa por todo o apoio e incentivo.

Aos colegas de mestrado Juliana, Nilmar, José Maurício, e todos os demais que convivi durante o curso.

As empresas que me permitiram acesso aos dados de seus projetos para construção das instâncias de testes que foram utilizadas para validar o presente trabalho.

Ao CEFET e ao programa de pós-graduação em Modelagem Matemática e Computacional pela oportunidade de me qualificar.

Aos professores do CEFET que contribuíram muito para a minha formação, transmitindo valiosos conhecimentos, em cada disciplina cursada.

À todos, o meu muitíssimo obrigada.

"A persistência é o menor caminho do êxito."
Charles Chaplin
"A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original."
Albert Einsten

Resumo

O foco desse trabalho é o Problema de Desenvolvimento de Cronogramas (PDC) de Projetos de Software. Nesse estudo são considerados diversos aspectos importantes, como a disponibilidade de recursos e suas habilidades; as tarefas do projeto e suas interdependências; os prazos; os custos; as estimações de tamanho das tarefas; a produtividade e a experiência das equipes. Somente os recursos humanos serão envolvidos, já que o desenvolvimento de projetos de software podem ser considerados atividades essencialmente intelectual e social. Muitos pesquisadores têm dedicado seus esforços aplicando técnicas de otimização em Engenharia de Software (SBSE - Search-Based Software Engineering) para resolver esse problema. Para resolver o PDC é proposta uma abordagem baseada na metaheurística MOVNS (MultiObjective Variable Neighborhood Search). Os resultados obtidos são comparados com um algoritmo NSGA-II existente na literatura. Os experimentos computacionais executados mostram que o algoritmo MOVNS proposto é estatisticamente melhor que o algoritmo NSGA-II em relação às métricas de cardinalidade, *epsilon* e hipervolume.

Palavras-chave: *Search Based Software Engineering*; Desenvolvimento de Cronograma de Projetos de Software; Recursos humanos; Otimização multiobjetivo; Métricas de avaliação.

Abstract

The focus of this work is the Problem of Development Schedules (PDS) Software Project. In this study several important aspects are considered, such as the availability of resources and skills, project tasks and their interdependencies, deadlines, costs, estimating size of the tasks, the productivity and experience of the teams. Only human resources are involved, since the development of software project can be considered an essentially intellectual and social activity. Many researchers have driven efforts to apply Search-Based Software Engineering (SBSE) techniques to solve this problem. This work presents an approach based on the MOVNS (MultiObjective Variable Neighborhood Search) metaheuristic for solving the PDS. The obtained results are compared against the results from a literature algorithm NSGA-II. Computational experiments done show that proposed MOVNS is statistically better than NSGA-II in relation to the cardinality, epsilon and hypervolume metrics.

Keywords: Search Based Software Engineering; Human Resources; Multiobjective Optimization; Metaheuristics; Evaluation Metrics.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Lista de Siglas

ACS - do inglês: Ant Colony System

AE - Algoritmo Evolucionário

AG - Algoritmos Genéticos

AM - Algoritmos Meméticos

BFPUG - do inglês: Brazilian Function Point Users Group

CMP - do inglês: Critical Method Path

FF - Final-Final

FI - Final-Início

GRASP - do inglês: Greedy Randomized Adaptative Search Procedure

IF - Início-Final

II - Início-Início

ILS - do inglês: Iterated Local Search

HC - do inglês: Hill Climbing

LRC - Lista Restrita de Candidatos

LS - do inglês: Latest Starting Time

MOVNS - do inglês: MultiObjective Variable Neighborhood Search

NSGA - do inglês: Nondominated Sorting Genetic Algorithm

PDC - Problema de Desenvolvimento de Cronograma

PF - Pontos de Função

PMBOK - do inglês: Project Management Body of Knowledge

PSO - do inglês: Particle Swarm Optimization

RAP - do inglês: Resources Allocation Problem

RCPSP - do inglês: Resource-Constrained Project Scheduling Problem

SA - do inglês: Simulated Annealing

SBES - Simpósio Brasileiro de Engenharia de Software

SBSE - do inglês: Search Based Software Engineering

SEBASE - do inglês: Software Engineering by Automated Search

SPS - Simulador de Projeto de Software

VNS - do inglês: Variable Neighborhood Search

WOES - Workshop de Otimização em Engenharia de Software

Capítulo 1

Introdução

1.1 Introdução Geral

O Desenvolvimento de Cronograma de um Projeto é uma atividade crucial na prática da Engenharia de Software, já que o orçamento total e os recursos humanos disponíveis devem ser gerenciados eficientemente para se obter um projeto de sucesso. Empresas com uma maior capacidade de controlar pessoas e o processo de desenvolvimento conseguem alocar eficientemente os recursos disponíveis para executar as tarefas demandadas pelo projeto, sempre satisfazendo uma variedade de restrições.

Dentro desse contexto, a atividade de alocação de recursos e a atividade de escalonamento de tarefas com restrições de recursos são cruciais para o processo de desenvolvimento do cronograma.

A alocação de recursos e o escalonamento de tarefas com restrição de recursos são atividades que possuem diversos aspectos que precisam ser considerados, como a disponibilidade de recursos; os prazos; os custos; as interdependências entre as tarefas; os replanejamentos; as estimações de custos; as estimações de tamanho de tarefas; e o planejamento multiprojeto, dentre outras questões. No presente estudo, somente serão considerados os recursos humanos envolvidos, já que o desenvolvimento de projetos de software podem ser considerados atividades essencialmente intelectual e social, conforme afirma Colares (2010).

Para auxiliar os gerentes de projeto na atividade de Desenvolvimento de Cronograma, existem diversas ferramentas de suporte, como *MS Project* (Microsoft, 2011), *Gnome Planner* (Project, 2012) e *OpenProj* (Inc., 2011), todas elas bem conhecidas e muito utilizadas pelos gerentes. Além dessas ferramentas, existem também manuais de práticas de projeto, como, por exemplo, o *PMBOK* (PMI, 2008), que é o mais difundido guia de boas práticas em gerência de projetos; e o COCOMO-II (Boehm, 2000), que é um manual de métodos de estimação para projetos de software. Apesar de existir todo esse suporte ao gerente de projeto, a obtenção de um cronograma de projeto de qualidade é responsabilidade do gerente, que ainda precisa contar com seu conhecimento, experiências e intuição.

Visando superar as questões acima citadas, a presente dissertação aborda o Problema de Desenvolvimento de Cronogramas de Projeto utilizando técnicas de otimização computacional, com o objetivo de superar as atuais limitações através de uma abordagem automatizada.

Como o custo e a duração de um projeto de software são duas variáveis críti-

cas para o sucesso do projeto, essa dissertação de mestrado adota uma abordagem baseada em otimização multiobjetivo, utilizando a metaheurística MOVNS *(MultiObjective Variable Neighborhood Search)* como algoritmo base desse projeto. A metaheurística MOVNS é baseada em um princípio simples: um processo de busca local realizada com mudanças sistemáticas de estruturas de vizinhanças. Aplicações do MOVNS em problemas de sequenciamento foram estudadas por Geiger (2008) e Arroyo et al. (2011), obtendo resultados satisfatórios.

O uso de técnicas de otimização para resolver problemas complexos da Engenharia de Software é uma área emergente de pesquisa denominada SBSE *(Search Based Software Engineering)* Harman et al. (2009). O principal objetivo do SBSE é oferecer mecanismos de apoio ao engenheiro de software para resolver problemas inerentes da Engenharia de Software. Baseado nesses conceitos, a abordagem proposta tem como objetivo apoiar e guiar o gerente na atividade de desenvolvimento de cronogramas de projetos de software.

Dentro desse contexto, o presente trabalho apresenta importantes contribuições. Segundo o conhecimento da autora, nesta dissertação é apresentada a primeira abordagem baseada em otimização multiobjetivo usando a metaheurística MOVNS *(MultiObjective Variable Neighborhood Search)* para resolver o PDC. A segunda contribuição é a adaptação para o PDC de um conjunto de 9 estruturas de vizinhança, com o objetivo de explorar todo o espaço de busca para o problema.

Outra importante contribuição é a realização de uma análise estatística da abordagem proposta, comparando os resultados obtidos com a implementação do algoritmo NSGA-II proposta por Colares (2010). As comparações são feitas com o algoritmo NSGA-II, pois grande parte das pesquisas em SBSE, como em Alba e Chicano (2007); Antoniol et al. (2005); Di Penta et al. (2011); Barreto et al. (2008); Colares (2010), são feitas utilizando algoritmos genéticos, seja na forma mono-objetiva, seja na forma multiobjetiva.

1.2 Justificativas

O interesse no estudo do Problema de Desenvolvimento de Cronogramas (PDC) ocorre por dois aspectos principais. O primeiro é o interesse teórico no PDC, que é um problema NP-Difícil, já que trata a junção do Problema de Alocação de Recursos (RAP - *Resources Allocation Problem*) com o Problema de Escalonamento de Tarefas com Restrições de Recursos (RCPSP - *Resource-Constrained Project Scheduling Problem*). O RCPSP, estudado por Kolish e Hartmann (2006); Hartmann e Kolish (2000); Viana e de Sousa (2000), dentre outros; e o RAP, estudado por Lai e Li (1999); Datta et al. (2008), dentre outros, são problemas bem conhecidos e frequentemente estudados, ambos pertencentes à classe dos problemas NP-Difícil. O segundo aspecto é o interesse prático da abordagem, uma vez que o desenvolvimento de cronogramas é uma atividade complexa, que tem impacto direto no sucesso de um projeto de desenvolvimento de software.

Observamos que na grande maioria dos estudos encontrados na literatura o PDC é tratado por meio de algoritmos baseados em busca populacional, sendo por meio de uma abordagem mono-objetivo ou multiobjetivo. Por isso, optou-se, neste trabalho, por desenvolver e comparar o desempenho de duas classes de algoritmos. Os

algoritmos baseados em busca local GRASP e MOVNS, e o algoritmo populacional NSGA-II foram os algoritmos utilizados nas comparações. Apontando o uso dessa abordagem como uma contribuição científica de interesse.

Não foram encontradas, na literatura, referências ao uso dos algoritmos GRASP e MOVNS para resolver o PDC, apontando o uso dessa abordagem como uma contribuição científica de interesse. Observamos também, que na grande maioria dos estudos encontrados na literatura o PDC é tratado por meio de algoritmos baseados em busca populacional, sendo por meio de uma abordagem mono-objetivo ou multiobjetivo. Por isso, optou-se, neste trabalho, por desenvolver e comparar o desempenho dessas duas classes de algoritmos.

1.3 Objetivos

1.3.1 Objetivo Geral

Esta dissertação tem, como objetivo geral, propor um algoritmo multiobjetivo eficiente para resolver o Problema de Desenvolvimento de Cronogramas de um projeto de software, tendo, como objetivos, a minimização de custos e a minimização do *makespan* do projeto.

1.3.2 Objetivos Específicos

Os objetivos específicos são:

- Conhecer a literatura acerca do Problema de Desenvolvimento de Cronogramas de um projeto de software e do uso de metaheurísticas para a resolução do mesmo;
- Implementar o algoritmo multiobjetivo baseado em busca local GRASP-MOVNS e o algoritmo baseado em busca populacional NSGA-II para resolver o problema tratado;
- Propor um conjunto de instâncias para o problema abordado;
- Realizar experimentos com os algoritmos implementados;
- Comparar os resultados obtidos por cada algoritmo, segundo as métricas de cardinalidade, hipervolume e epsilon;
- Publicar os resultados da pesquisa em eventos e periódicos.

1.4 Organização do Trabalho

O restante desta dissertação é organizado da seguinte maneira. No Capítulo 2 são descritas as características do problema abordado e apresentada a modelagem matemática para o PDC. No Capítulo 3 é apresentada a revisão bibliográfica relacionada

ao PDC. No Capítulo 4 é mostrada uma revisão sobre métodos heurísticos e os algoritmos implementados. No Capítulo 5 é apresentada a metodologia utilizada para o desenvolvimento do trabalho. No Capítulo 6 são postos e analisados os resultados dos testes realizados. O Capítulo 7 conclui o trabalho e aponta sugestões para trabalhos futuros.

Capítulo 2

Caracterização do Problema

Este capítulo apresenta o Problema de Desenvolvimento de Cronogramas. A Seção 2.1 apresenta uma sucinta contextualização do PDC. A Seção 2.2 apresenta a formulação para o PDC, levando em consideração os diversos aspectos presentes no cotidiano de projetos de desenvolvimento de software. A Seção 2.3 apresenta a modelagem matemática proposta para o PDC.

2.1 Contextualização

Paula Filho (2009) caracteriza o desenvolvimento de cronogramas de um projeto de software como um processo que define as relações de precedência entre as tarefas específicas de Engenharia de Software e a alocação dos recursos humanos necessários para a execução das respectivas tarefas.

No PMBOK (PMI, 2008), importante guia de boas práticas em gerência de projetos, a atividade de desenvolver o cronograma é definida como um processo de análise do sequenciamento das atividades, suas durações, os recursos necessários à sua execução e restrições existentes, visando criar o cronograma do projeto.

A Figura 2.1 apresenta uma visão geral do processo de desenvolvimento de cronogramas, ilustrando as técnicas e ferramentas de suporte ao processo (PMI, 2008). As técnicas sugeridas pelo PMBOK abrangem o gerenciamento de qualquer tipo de projeto; portanto, muitas delas não consideram particularidades dos projetos de software.

No PMBOK existe uma área de conhecimento denominada Gerenciamento do Tempo do Projeto, que engloba diversos processos necessários para gerenciar o término pontual do projeto. O processo de desenvolvimento do cronograma faz parte dessa área de conhecimento (PMI, 2008). A Figura 2.2 mostra o diagrama de fluxo de dados do processo para desenvolver o cronograma, no qual as linhas em negrito representam as relações dentro da área de conhecimento de Gerenciamento do Tempo, e a linhas claras representam as relações entre diferentes áreas.

Além dos fatores supracitados, a proposta descrita nesse Capítulo considera vários outros fatores que estão presentes no cotidiano dos gerentes de projeto de software.

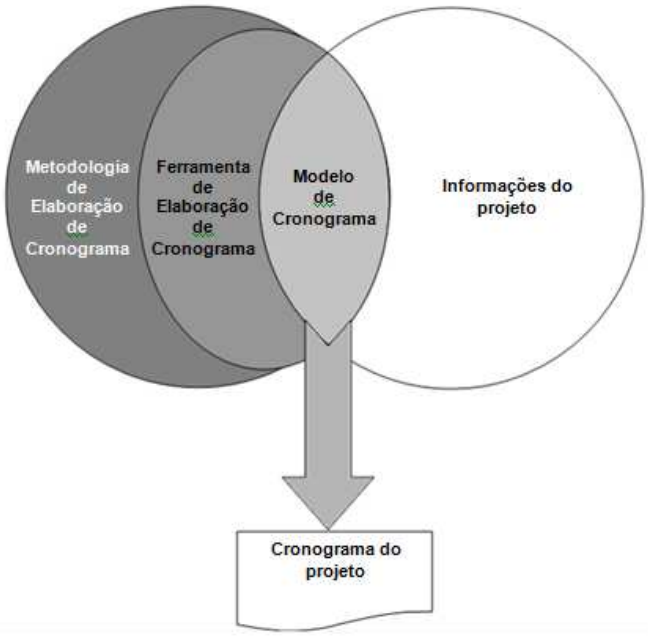

Figura 2.1: Resumo do desenvolvimento de cronogramas. Fonte: PMI (2008)

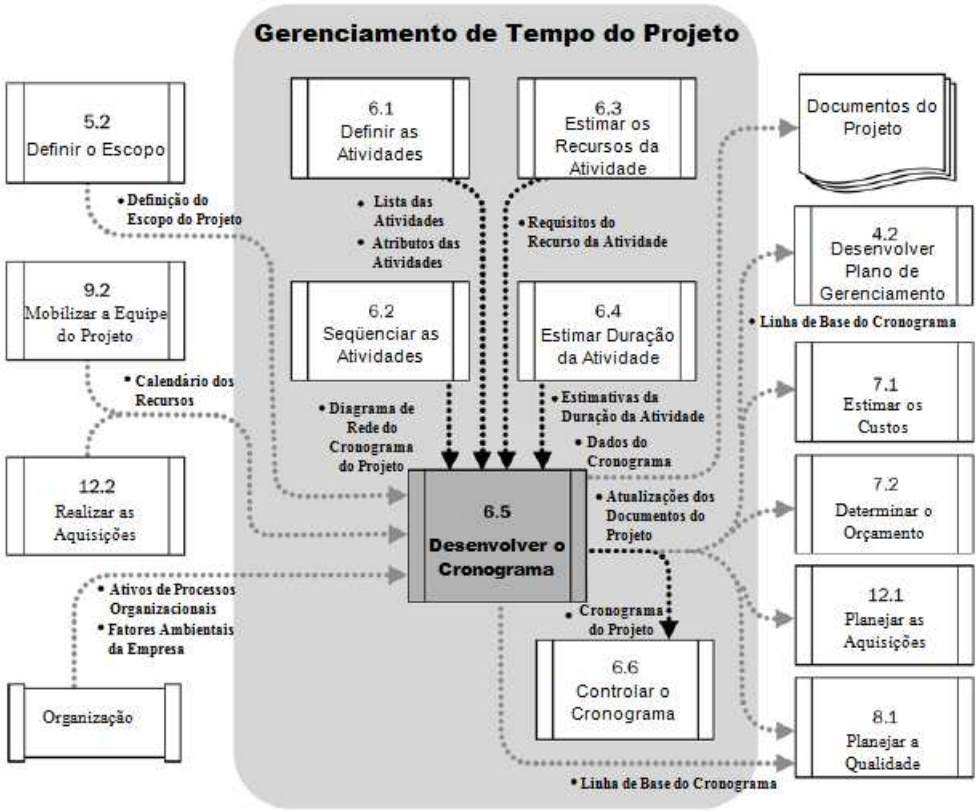

Figura 2.2: Diagrama de fluxo de dados do processo Desenvolver Cronograma. Fonte: PMI (2008)

2.2 Caracterização do Problema

Os aspectos importantes a serem considerados na formulação do Problema de Desenvolvimento de Cronogramas de Projeto de Software (PDC) são, segundo Colares (2010):

(i) **Tarefas:** $J = \{j_1, j_2, ..., j_{|J|}\}$ representa o conjunto das tarefas a serem executadas. Na formulação proposta, são consideradas como tarefas quaisquer atividades que precisem ser executadas por recursos humanos. Cada tarefa possui os seguintes atributos: esforço estimado, nível de importância e, se existirem, datas de início e término das tarefas.

(ii) **Recursos Humanos:** $R = \{r_1, r_2, ..., r_{|R|}\}$ representa o conjunto dos recursos disponíveis para execução das tarefas. Os recursos são divididos em dois tipos: empregado e contratado. O recurso tipo empregado possui os seguintes atributos: salário, dedicação diária, custo de hora extra e tempo máximo de hora extra. O recurso tipo contratado recebe por hora trabalhada e possui, como atributos: custo da hora trabalhada e máxima dedicação diária. Ambos os tipos possuem o atributo disponibilidade, que representa o calendário de cada recurso.

(iii) **Habilidades:** $H = \{h_1, h_2, ..., h_{|H|}\}$ representa o conjunto de habilidades que um recurso humano possui, ou que uma determinada tarefa requer para sua execução.

(iv) **Tarefas** $\times$ **Habilidades:** cada tarefa possui uma lista de habilidades como requisitos necessários para sua execução.

(v) **Recursos** $\times$ **Habilidades:** cada recurso possui uma lista de habilidades com seu respectivo nível de proficiência.

(vi) **Recursos** $\times$ **Tarefas:** cada recurso pode executar uma ou mais tarefas, e possui um nível de experiência em cada uma delas.

(vii) **Alocação de equipes:** como nem sempre é possível quebrar as tarefas de um projeto pequenas o suficiente para que um único recurso possa executá-la, é necessário atribuir um conjunto de recursos para executa uma determinada tarefa. Essa atribuição de vários recursos para executar uma tarefa define a alocação das equipes. A alocação de um recurso a determinada tarefa é representada por um valor percentual de sua dedicação diária. A atribuição de 0% indica que o recurso não está alocado à tarefa, e a atribuição de percentuais superiores a 100% indicam o uso de hora extra. A atribuição de -1 indica que o recurso não possui habilidades necessárias para executar a tarefa.

(viii) **Custos:** a alocação de equipes gera custos, e é desejável que seja feita de forma a minimizá-los.

(xi) **Interdependências entre tarefas:** descrita pelo PMI (2008) como "sequenciamento das tarefas". A abordagem proposta utiliza os quatro conceitos de vínculos entre tarefas utilizados pela maioria das ferramentas de gerência de projeto:

Início-Início (II): A tarefa dependente (B) não pode começar até que a tarefa da qual ela depende (A) seja iniciada. A tarefa dependente pode ser iniciada a qualquer momento depois que a tarefa da qual ela depende for iniciada. O tipo de vínculo II não exige que as duas tarefas sejam iniciadas ao mesmo tempo. A Figura 2.3 ilustra o vínculo II.

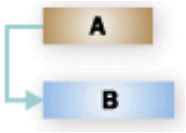

Figura 2.3: Início-Início (II)

Início-Final (IF) A tarefa dependente (B) não pode ser concluída até que a tarefa da qual ela depende (A) seja iniciada. A tarefa dependente pode ser concluída a qualquer momento depois que a tarefa da qual ela depende for iniciada. O tipo de vínculo IF não exige que a tarefa dependente seja concluída ao mesmo tempo em que a tarefa da qual ela depende é iniciada. A Figura 2.4 ilustra o vínculo IF.

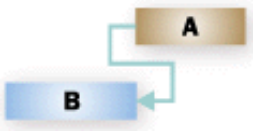

Figura 2.4: Início-Final (IF)

Final-Início (FI) A tarefa dependente (B) não pode começar até que a tarefa da qual ela depende (A) seja concluída. A Figura 2.5 ilustra o vínculo FI.

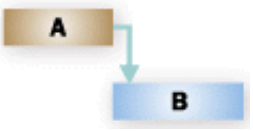

Figura 2.5: Final-Início (FI)

Final-Final (FF) A tarefa dependente (B) não pode ser concluída até que a tarefa da qual ela depende (A) seja concluída. A tarefa dependente pode ser concluída a qualquer momento depois que a tarefa da qual ela depende for concluída. O tipo de vínculo FF não exige que as duas tarefas sejam concluídas ao mesmo tempo. A Figura 2.6 ilustra o vínculo FF.

(x) **Cronograma:** estimado o tempo de execução de cada tarefa, é possível definir o cronograma do projeto. Para gerar o cronograma é necessário calcular para cada tarefa: sua duração $t_j^{duracao}$, e em seguida calcular o tempo de início e de término de cada tarefa considerando suas interdependências e eventuais restrições do tipo II, IF, FI, FF.

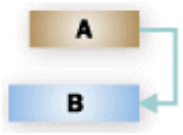

Figura 2.6: Final-Final (FF)

(xi) **Produtividade:** as expressões (2.1) e (2.2), segundo Colares (2010), representam, respectivamente, o cálculo da produtividade de uma equipe e o cálculo de duração de uma tarefa:

$$prod_{r,j} = x_{r,j}.r^{dedicacao}.\left(\prod_{h\in(H^r \bigcap H^j)} r^{proef}(\ h)\right).r^{exp}(\ j) \tag{2.1}$$

$$t_j^{duracao} = \frac{j^{esforco}}{\sum_{r\in R} prod_{r,j}},\quad \forall j \in J \tag{2.2}$$

As variáveis utilizadas nestas expressões são definidas como:

- x_{rj}: proporção de dedicação do recurso r para executar a tarefa j. É uma variável de decisão do problema.
- $r^{dedicacao}$: dedicação diária em horas do recurso r.
- H^r: conjunto de habilidades que o recurso r possui.
- H^j: conjunto de habilidades necessárias para a execução da tarefa j.
- $r^{proef}(h)$: fator de ajuste de proficiência do recurso r na habilidade h. Seus possíveis valores estão representados na Tabela 2.1.
- $r^{exp}(j)$: fator de ajuste de experiência do recurso r na tarefa j. Seus possíveis valores estão representados na Tabela 2.2.
- $j^{esforco}$: esforço da tarefa j em Pontos de Função (PF). Pontos de Função é uma medida padronizada e normalizada do tamanho funcional dos requisitos lógicos de um software (BFPUG, 2013).
- $t_j^{duracao}$: tempo de duração da tarefa j em dias.

Na Figura 2.7, tem-se a representação de uma matriz de produtividade $prod_{r,j}$.

	j_1	j_2	j_3	j_4	j_5	j_6
r_1	0	0,75	0	0	1	0
r_2	0,65	0,69	0	0	0,35	0
r_3	1	2,15	0	0	0	0
r_4	0	0	1,23	3,98	2,17	0
r_5	0	0	4,15	2,09	0	1

Figura 2.7: Matriz de produtividade $prod_{r,j}$

Tabela 2.1: Fatores de ajuste de proficiência. Fonte: (Boehm, 2000)

	very low	*low*	*normal*	*high*	*very high*
Habilidades	15º perc.	35º perc.	55º perc.	75º perc.	90º perc.
Análise	0,70	0,84	1,00	1,18	1,41
Implementação	0,75	0,87	1,00	1,14	1,32
Outros	0,84	0,92	1,00	1,10	1,18

Tabela 2.2: Fatores de ajuste de experiência. Fonte: (Boehm, 2000)

	extra low[3]	*very low*	*low*	*normal*	*high*	*very high*
Descrição	=0	<=2meses	<=6meses	<=1ano	<=3anos	>3anos
Valor	0,63	0,82	0,91	1,00	1,14	1,23

Observe que o cálculo de duração para uma tarefa do projeto é feito pela somatória dos valores de uma coluna, representando o esforço de produtividade associada à tarefa j_3. No exemplo posto, segundo a expressão (2.2), tem-se que:

$$\begin{aligned} t_3^{duracao} &= \frac{j_3^{esforco}}{\sum_{r=5}^{r=1} prod_{r,3}} \\ &= \frac{j_3^{esforco}}{0+0+0+1,23+4,15} \\ &= \frac{j_3^{esforco}}{5,38} \end{aligned}$$

Os valores utilizados como fatores de ajustes para habilidade e experiência são apresentados, respectivamente, nas Tabelas 2.1 e 2.2. Esses valores foram utilizados como propostos em Colares (2010) e foram originalmente baseados nos multiplicadores de esforço contidos no manual do método COCOMO-II (Boehm, 2000). Também com base neste mesmo manual, as habilidades são classificadas em níveis, de acordo com a posição relativa do recurso em relação ao mercado, enquanto a experiência é classificada pelo acúmulo de tempo de trabalho. Além disso, as habilidades são divididas em três grupos, de acordo com seu tipo:

- Análise: inclui os tipos relativos à análise, ao desenho e aos atributos psicológicos, como comunicação e cooperação;
- Implementação: compreende habilidades de desenvolvimento e programação;
- Outros: relativos aos demais tipos, como habilidades em bancos de dados, sistemas operacionais, redes e etc.

Diante das questões citadas anteriormente, os objetivos a serem satisfeitos pelo Problema de Desenvolvimento de Cronogramas (PDC) são:

(i) Minimizar o tempo total do projeto, ou *makespan*;

(ii) Minimizar o custo total.

Ao mesmo tempo, este Problema de Desenvolvimento de Cronograma deve satisfazer, em suas soluções, as seguintes restrições:

(i) conjunto de habilidades requeridas por cada tarefa;

(ii) tempo máximo de dedicação de cada recurso;

(iii) interdependência entre as tarefas;

(iv) disponibilidade de cada recurso;

(v) prazo para término do projeto, se existir; e

(vi) limite de custos, se existir.

2.3 Formulação Matemática

Nesta seção é apresentada a formulação matemática definida por Colares (2010) para resolver o Problema de Desenvolvimento de Cronogramas. Em seguida, na Subseção 2.3.2, são descritas as adaptações propostas no modelo matemático de Colares (2010), com o fim de melhor adequar o modelo aos propósitos do presente projeto. Na Subseção 2.3.3 é apresentada a proposta adaptada da formulação matemática.

2.3.1 Formulação Matemática Obtida na Literatura

Nessa Subseção é apresentada a formulação matemática proposta por Colares (2010) para resolver o Problema de Desenvolvimento de Cronogramas.

Estruturas de Dados

A estrutura de dados proposta compreende os seguintes itens:

(a) $I = \{1, \ldots, i, \ldots, |I|\}$: conjunto de tarefas a serem executadas;

(b) $J = \{1, \ldots, j, \ldots, |J|\}$: conjunto de recursos humanos;

(c) $T = \{1, \ldots, t, \ldots, |T|\}$: conjunto de períodos de tempo;

(d) $J^i = \{j_1, j_2, \ldots, j_{|J|}\}$: conjunto de recursos humanos que podem executar a tarefa i;

(e) $D^i = \{i_1, i_2, \ldots, i_{|I|}\}$: conjunto de dependências da tarefa i, isto é, tarefas que precisam ser executadas antes da tarefa i.

Dados

Os dados necessários à solução do problema são:

(a) t_{ij}: tempo de execução da tarefa i pela equipe j, ou seja, sua duração;

(b) c_{ij}: custo de desenvolvimento da tarefa i pela equipe j;

(c) c_j^{he}: custo de hora-extra para equipe j;

(c) d_i: *deadline* de entrega da tarefa i;

(d) w_i: custo pelo atraso da tarefa i.

Variáveis de Decisão

As variáveis de decisão definidas para este modelo são:

(a) x_{ij}^t: variável de decisão que define a alocação da tarefa i ao recurso j no período de tempo t, de modo que:

$$x_{ij}^t = \begin{cases} 1, & \text{se a tarefa } i \text{ é alocada ao recurso } j \text{ no período de tempo } t; \\ 0, & \text{caso contrário.} \end{cases}$$

(b) a_i: atraso na entrega da tarefa i;

(c) he_j: quantidade de horas-extra da equipe j;

(d) s_j: término das tarefas da equipe j;

(e) s: *makespan*, ou seja, tempo de término de todas as tarefas.

Objetivos

Os objetivos tratados em Colares (2010) são:

$$\min \begin{cases} A = \sum_{i \in I} (w_i a_i) \\ C = \left[\sum_{i \in I} \sum_{j \in J^i} \sum_{t=1}^{|T|-t_{ij}} (c_{ij} x_{ij}^t) \right] + \left[\sum_{j \in J} c_j^{he} he_j \right] \\ S = s \end{cases}$$

O objetivo A indica a minimização do atraso total de entrega das tarefas do projeto, ponderado por um fator de custo w_i associado a cada tarefa i. O objetivo C implica na minimização do custo total do projeto; e o objetivo S é a função de minimização do *makespan* (*makespan* representa a duração total do cronograma do projeto, isto é, quando todas as tarefas estiverem finalizadas).

2.3.2 Adaptações Propostas

Para melhor adequar o modelo aos propósitos da presente dissertação, foram feitas as seguintes alterações:

- A nomenclatura dos conjuntos foi alterada, para tornar mais intuitiva a associação dos conjuntos aos dados armazenados;
- Foram criados os conjuntos H, J^h e R^h, para permitir o agrupamento das tarefas e recursos humanos por habilidades;
- O conjunto J^i foi removido, pois o conjunto J^h proposto tem função semelhante;
- A função objetivo A, que minimiza o atraso das tarefas, foi excluída, pois não foi possível coletar dados de projetos reais que permitissem o cálculo do atraso das tarefas.

As alterações realizadas têm, como principal objetivo, melhorar o desempenho do algoritmo proposto, que irá trabalhar somente com soluções válidas para o problema estudado.

2.3.3 Formulação Proposta

É apresentada, a seguir, a formulação matemática para o Problema de Desenvolvimento de Cronogramas resultante da aplicação das alterações propostas na Subseção 2.3.2.

Estrutura de Dados

Sejam, então, os seguintes dados a serem utilizados para a determinação da solução do problema:

(a) $J = \{1, ..., j, ..., |J|\}$: conjunto de tarefas a serem executadas;

(b) $R = \{1, ..., r, ..., |R|\}$: conjunto de recursos humanos disponíveis;

(c) $H = \{1, ..., h, ..., |H|\}$: conjunto de habilidades relacionadas ao projeto de software desenvolvido.

(d) $T = \{1, ..., t\}$: conjunto de períodos de tempo;

(e) $J^h = \{j_1, j_2, ...\}$: conjunto de tarefas que precisam da habilidade h para serem executadas;

(f) $R^h = \{r_1, r_2, ...\}$: conjunto de recursos humanos que possuem a habilidade h;

(g) $R^j = \{r_1, r_2, ...\}$: conjunto de recursos humanos que podem executar a tarefa j;

(h) $D^j = \{j_1, j_2, ...\}$: conjunto de dependências da tarefa j, isto é, tarefas que precisam ser executadas antes da tarefa j.

Dados

Os dados necessários à solução do problema são:

- $t_j^{duracao}$: tempo de duração da tarefa j;
- c_{rj}: custo de desenvolvimento da tarefa j pelo recurso r;
- c_r^{he}: custo de hora extra do recurso r.

Variáveis de Decisão

As variáveis de decisão do problema são definidas como:

(a) x_{rj}^t: variável de decisão que define a alocação da tarefa j ao recurso r no período t, de modo que:

$$x_{rj}^t = \begin{cases} 1, & \text{se a tarefa } j \text{ é alocada ao recurso } r \text{ no período de tempo } t; \\ 0, & \text{caso contrário.} \end{cases}$$

(b) he_r: quantidade de horas-extra do recurso r;

(c) s_r: instante de término das tarefas do recurso r;

(d) s: *makespan* (tempo de término de todas as tarefas).

Objetivos

O problema em análise consiste em minimizar duas funções objetivos, quais sejam:

- C: a função de minimização do custo total do projeto;
- S: a função de minimização do *makespan*.

As duas funções descritas acima são representadas matematicamente na seguinte forma:

$$\min \begin{cases} C = \left[\sum_{j\in J}\sum_{r\in R^j}\sum_{t=1}^{|T|-t_{rj}} \left(c_{rj}x_{rj}^t\right)\right] + \left[\sum_{r\in R}\left(c_r^{he}he_r\right)\right] \\ S = s \end{cases}$$

Restrições

As restrições associadas ao PDC são:

- Todas as tarefas devem ser executadas:

$$\sum_{r\in R}\sum_{t=1}^{|T|-t_{rj}} x_{rj}^t = 1, \quad \forall j \in J \tag{2.3}$$

- Um recurso pode executar no máximo uma tarefa no período t:

$$\sum_{j \in J} x_{rj}^{t} \leq 1, \quad \forall r \in R; \quad \forall t \in T \tag{2.4}$$

- Término das tarefas executadas pelo recurso r:

$$s_r \geq (t + t_{rj}) x_{rj}^{t}, \quad \forall j \in J; \ \forall r \in R; \ \forall t \ : \ 0 \leq t \leq (|T| - t_{rj}) \tag{2.5}$$

- *Makespan* (término de todas as tarefas):

$$s \geq s_r, \quad \forall r \in R \tag{2.6}$$

- Interdependências entre as tarefas (tarefa j' é pré-requisito para tarefa j):

$$x_{rj}^{t} \leq x_{r'j'}^{t+t_{rj}}, \quad \forall j \in J; \ \forall j' \in D^{j}; \ \forall r \in R^{j}; \ \forall r' \in R^{j'}; \ \forall t \ : \ 0 \leq t \leq t_{rj} \tag{2.7}$$

- Integralidade e não negatividade:

$$x_{rj}^{t} \in \{0, 1\}; s_r \geq 0; \forall j \in J; \forall r \in R; \forall t \ : \ 0 \leq t \leq (|T| - t_{rj}) \tag{2.8}$$

2.3.4 Visão Computacional

Nesta seção é apresentada uma visão do modelo computacional proposto para representar os principais elementos apresentados na formulação matemática do Problema de Desenvolvimento de Cronogramas.

A Figura 2.8 apresenta o diagrama de classes proposto para representar as principais classes de entidade identificadas na formulação do PDC: Recursos, Tarefas e Habilidades.

Para representar as interdependências entre as tarefas, foi utilizada uma estrutura de Grafo. O Grafo de Rede do Projeto GRP é um grafo acíclico direcionado, denominado $GRP(V, A)$, tendo o conjunto de vértices $V = \{j_1, j_2, ..., j_J\}$ e o conjunto de arcos A. O conjunto de arcos A representa a interdependência das tarefas, de modo que $(j_i, j_k) \in A$ se a tarefa j_i deva ser finalizada antes que a tarefa j_k possa ser iniciada. Cada vértice $j_i \in V$ representa uma tarefa do projeto, tendo como identificadores o índice j e o tempo de duração $t_j^{duracao}$ da tarefa j. A Figura 2.9 mostra um exemplo de um GRP para o PDC. Observe a relação de precedência entre as seis tarefas existentes, com todas as tarefas com vínculo do tipo Final-Inicio (FI).

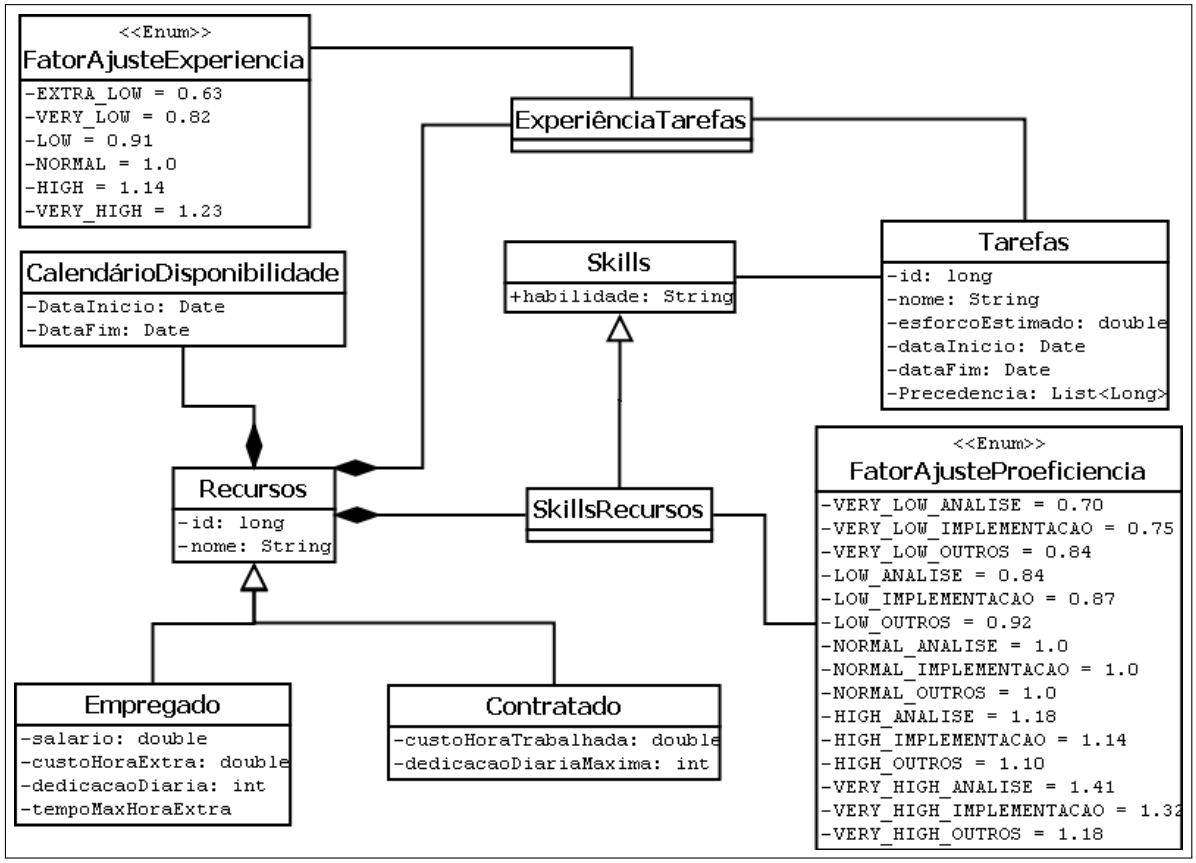

Figura 2.8: Diagrama de Classe para o PDC.

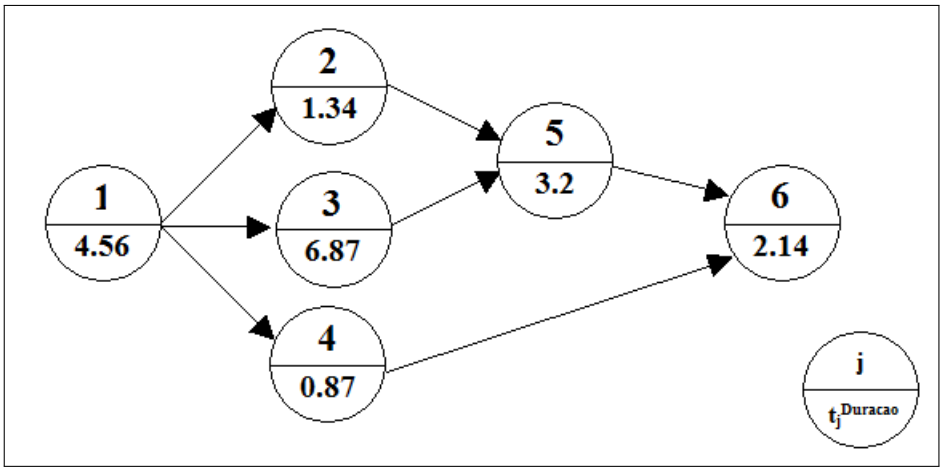

Figura 2.9: Exemplo de Grafo da Rede do Projeto (GRP).

Capítulo 3

Revisão Bibliográfica

Este capítulo apresenta uma revisão bibliográfica referente ao Problema de Desenvolvimento de Cronograma. A Seção 3.1 aborda a utilização de otimização para resolver problemas de Engenharia de Software. Na Seção 3.2 são apresentados trabalhos que aplicam otimização nas diversas áreas da Engenharia de Software. Na Seção 3.3 é apresentada uma revisão sobre a aplicação de otimização em planejamento de projetos de software.

3.1 Histórico

A utilização de métodos de otimização para resolver problemas de Engenharia de Software é uma área de pesquisa recente, denominada *Search-Based Software Engineering* (SBSE). Suas origens podem ser rastreadas até os primeiros trabalhos publicados na década de 1970.

O primeiro registro de pesquisa utilizando otimização na Engenharia de Software data de 1976. Miller e Spooner (1976) publicaram um estudo em que fazem uso de métodos de maximização numérica para geração de dados de teste. Já em 1992, Xanthakis et al. (1992) publicaram, pela primeira vez, um trabalho utilizando uma metaheurística para resolver um problema de Engenharia de Software. Nesse trabalho, foi utilizado um algoritmo genético para geração de casos de testes.

O termo *Search-Based Software Engineering* (SBSE) surgiu em 2001, no primeiro *survey* sobre o assunto publicado por Harman e Jones (2001). Como pode ser visto na Figura 3.1, na década de 90 o volume de pesquisas nessa área era pequeno e esporádico. Somente a partir de 2001 pode-se observar um crescimento considerável de pesquisas em SBSE.

Pesquisas em SBSE estão emergindo também no cenário brasileiro de Engenharia de Software. A comunidade brasileira de pesquisas em SBSE promoveu em 2010 o primeiro evento nessa área, denominado WOES (*Workshop* de Otimização em Engenharia de Software), realizado como atividade paralela do SBES (Simpósio Brasileiro de Engenharia de Software). Vergilio et al. (2011) apresentam uma análise dos trabalhos produzidos pela comunidade brasileira de pesquisa em SBSE publicados no SBES. Na Figura 3.2 pode-se observar o crescimento das publicações nacionais em SBSE

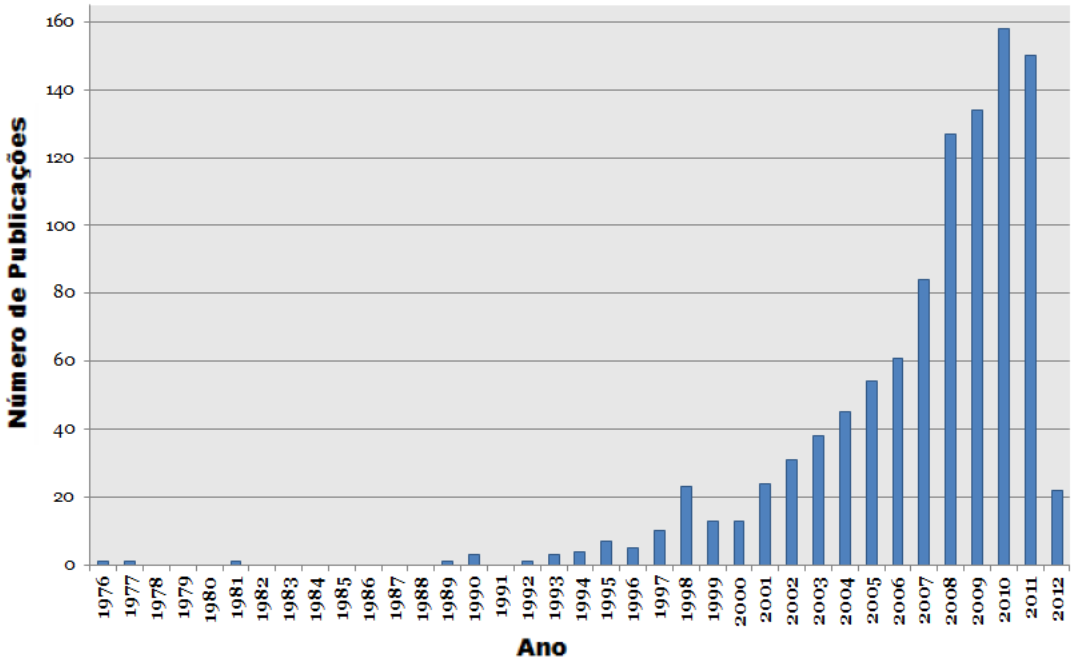

Figura 3.1: Número de publicações em SBSE por ano no mundo. Fonte: SEBASE (2012)

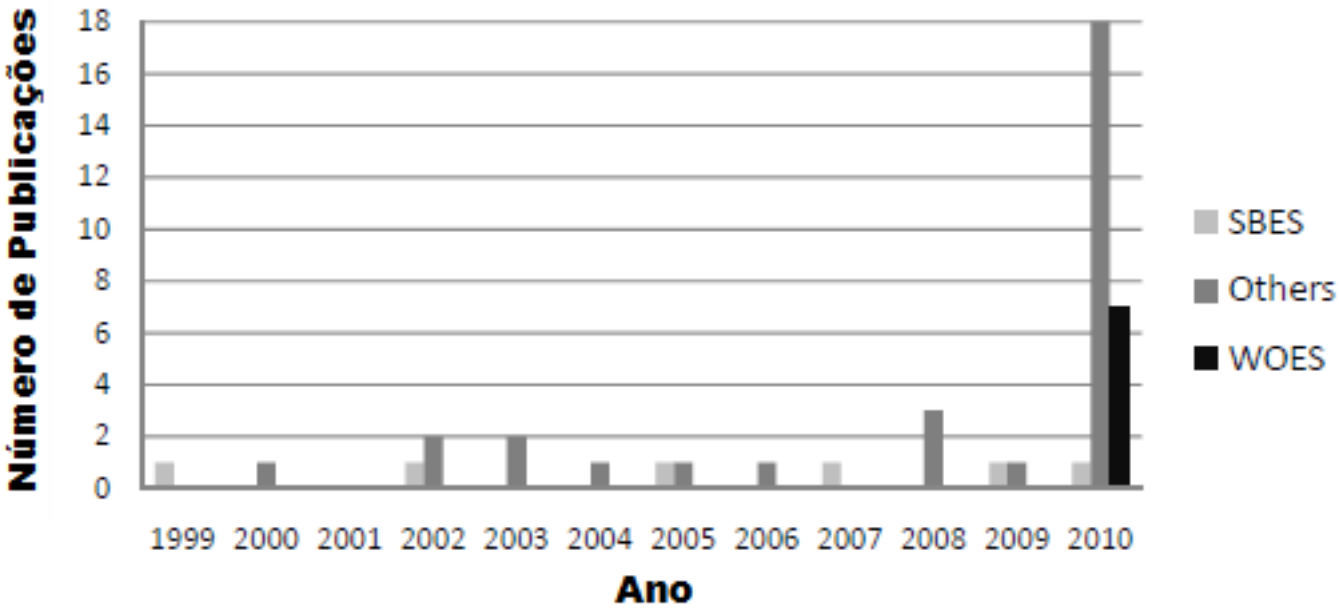

Figura 3.2: Crescimento Quantitativo de publicações em SBSE no Brasil. Fonte: Vergilio et al. (2011)

3.2 Aplicações de Otimização na Engenharia de Software

Harman et al. (2009) publicaram uma detalhada análise e revisão sobre tendências, técnicas e aplicações de otimização na Engenharia de Software. Um dos resultados desse estudo pode ser visto na Figura 3.3, que apresenta o percentual de publicações de SBSE nas principais áreas da Engenharia de Software.

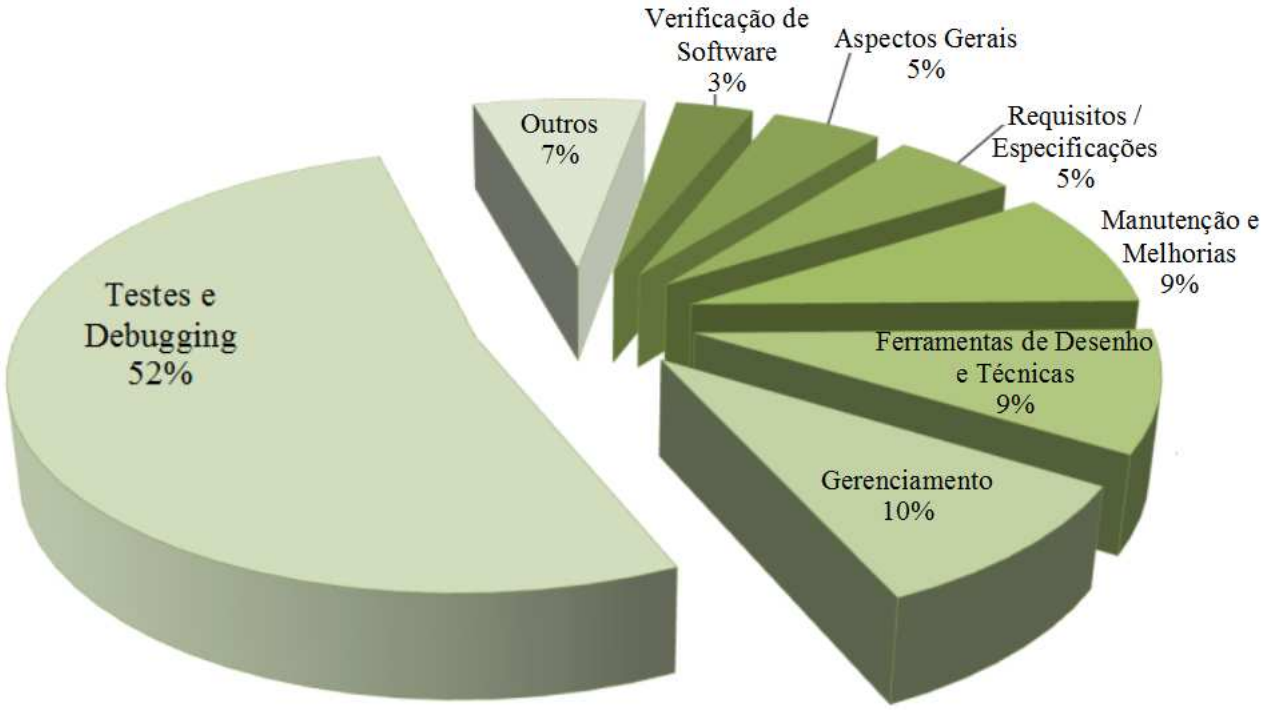

Figura 3.3: Percentual de publicações de SBSE nas principais áreas da Engenharia de Software. Fonte: SEBASE (2012)

Como pode ser visto na Figura 3.3, a área de *Testes e Debugging* concentra o maior número de pesquisas em SBSE. A aplicação de metaheurísticas nessa área tenta solucionar os seguintes problemas:

- priorização de casos de teste; e
- geração de dados para teste.

Este último problema é o que apresenta maior número de publicações, como os trabalhos de Michael et al. (2001), Hermadi e Ahmed (2003), Khor e Grogono (2004) e Harman (2007). O Problema de Priorização de Casos de Testes, que também é bastante estudado, é mais comum em testes de regressão, como visto nos trabalhos de Li et al. (2007) e Maia et al. (2008).

A Engenharia de Requisitos pertence à área de *Requisitos e Especificações* e, como pode ser visto na Figura 3.3, possui um percentual de apenas 5% das publicações em SBSE. Na área de *Engenharia de Requisitos*, um problema bastante difundido trata da seleção do conjunto de requisitos que devem ser incluídos na próxima iteração do

desenvolvimento do projeto. Esse problema é conhecido em SBSE como Problema do Próximo *Release*, segundo Bagnall et al. (2001) e e Zhang et al. (2007).

A área de *Estimativa de Software* é uma importante atividade que ocorre ainda na fase de planejamento do projeto. Em SBSE, são estudados problemas relacionados a estimativas de tamanho em, por exemplo, Dolado (2000); e estimativas de custo de software, em, por exemplo, Dolado (2001) e Aguilar-Ruiz et al. (2001). A Estimativa de Software pertence a área de *Gerenciamento*, que representa 10% do percentual das publicações em SBSE, conforme a Figura 3.3.

A área de *Desenho de Software* é a parte da Engenharia de Software que se encarrega de transformar os resultados da análise de requisitos em um documento ou conjunto de documentos capazes de serem interpretados diretamente pelo programador. Pesquisas de otimização nessa área são muito usadas como meio de desenvolver novas técnicas de projeto de software. Simons e Parmee (2008a) e Simons e Parmee (2008b) propuseram um algoritmo genético multiobjetivo para tratar coesão e acoplamento entre classes, dentro de padrões de projeto orientado a objetos. O'Keeffe e Cinnéide (2004) converteram o padrão de projeto orientado a objetos em um problema de otimização, utilizando a metaheurística *Simulated Annealing* para solucioná-lo. Para avaliar a qualidade do projeto, O'Keeffe e Cinnéide (2004) utilizaram um conjunto de métricas. A área de *Desenho de Software* pertence a área de *Ferramentas de Desenho e Técnicas* e, como pode ser visto na Figura 3.3, possui um percentual de 9% das publicações em SBSE.

A área de *Manutenção de Software* trata as alterações no software, a fim de corrigir defeitos e não conformidades encontrados durante sua experiência de uso, bem como a adição de novas funcionalidades para melhorar a usabilidade e funcionalidade do software. Harman (2006) apresenta formas de tratar problemas de reengenharia e manutenção de software do ponto de vista de otimização. A *Manutenção de Software* pertence à área de *Manutenção e Melhorias*, que representa 9% do percentual das publicações em SBSE, conforme Figura 3.3.

A área de *Planejamento de Projeto*, tema abordado nessa dissertação, está relacionada à gestão de diversas tarefas complexas durante todo o ciclo de vida do projeto de software, tentando otimizar os processos de produção de software e os produtos por ele produzidos. O *Planejamento de Projeto* pertence a área de *Gerenciamento* e, como pode ser visto na Figura 3.3, possui um percentual de 10% das publicações em SBSE, ou seja, o segundo maior percentual de publicações em SBSE.

A Seção 3.3 apresenta, de forma detalhada, o uso de SBSE em planejamento de projetos.

3.3 SBSE em Planejamento de Projetos

Chang et al. (1994) apresentaram o primeiro artigo usando SBSE para tratar problemas de gerenciamento de projetos. No referido artigo foi introduzido o SPMNet, uma abordagem para resolver problemas de alocação de recursos. O SPMNet utiliza uma técnica automática, baseada em algoritmos genéticos (AG), para determinar a alocação ótima dos recursos e calcular o tempo total e o custo de um projeto. O artigo se limita a apresentar a formulação do SPMNet, não apresentando validações ou testes.

Em 1998, Chang et al. (1998) publicaram um artigo propondo melhorias no algoritmo genético utilizado no SPMNet, reduzindo drasticamente a complexidade do espaço de busca, fornecendo soluções ótimas ou muito próximas do ótimo para a alocação de recursos e escalonamento de tarefas do projeto. A principal limitação do SPMNet é a elevada complexidade da modelagem apresentada.

Aguilar-Ruiz et al. (2001) propõem o uso de um Simulador de Projeto de Software (SPS) baseado em algoritmo evolucionário, que gera um conjunto de regras de gerenciamento de projeto. O uso de simuladores de projetos ajuda a estimar projetos de software e produzir, automaticamente, regras de gerenciamento, que podem ser aplicadas: antes do início do projeto, para definir as políticas gerenciais adequadas; durante a execução dos projetos, auxiliando as tomadas de decisão imediatas; ou depois de concluir o projeto, fazendo uma análise final. O uso de regras de gerenciamento permite obter avaliações qualitativas para diversas variáveis de interesse de um projeto de software, como tempo, esforço ou qualidade. Esse estudo trata o problema de gerenciamento de projeto com o foco nas políticas gerenciais que auxiliam as tomadas de decisões.

Chang et al. (2001) propõem novas melhorias em seu algoritmo genético, proposto inicialmente em Chang et al. (1994), para resolver o problema de alocação de recursos e escalonamento de tarefas. Chang et al. (2008) introduziram um modelo baseado no conceito de linha de tempo, dividindo o problema principal em subproblemas menores, aos quais aplicam algoritmos genéticos, com o objetivo de minimizar o custo do projeto. A formulação apresentada é bem completa, tratando fatores como curva de aprendizado e treinamentos. Em projetos de larga escala, situação em que um gerente poderia se beneficiar do uso de SBSE, a dimensão temporal pode tornar o problema inviável. Para testar a abordagem proposta, foi utilizado um projeto fictício de tamanho médio.

Antoniol et al. (2004a), Antoniol et al. (2004b), Antoniol et al. (2005) utilizaram algoritmos de busca em problemas de gerenciamento de projetos. Foram aplicados algoritmos genéticos (AG), *Simulated Annealing* (SA), Busca Randômica e *Hill Climbing* (HC) para a solução do Problema de Alocação de Pessoas. Também consideram problemas de retrabalho, erros de estimativas e abandono de projetos, aspectos importantes e comuns em projetos de Engenharia de Software. Estes trabalhos apresentam algumas limitações, como o foco em projetos de manutenção e não de desenvolvimento, além de desconsiderar aspectos importantes, como experiências e habilidades individuais e as interdependências entre as tarefas. Para validar a pesquisa, foram utilizados dados reais de uma grande empresa de projetos de manutenção. Os autores concluíram que o uso de SBSE pode reduzir em até 50% o tempo de duração de um projeto.

Alba e Chicano (2007) trataram o Problema de Desenvolvimento de Cronogramas de projetos utilizando um algoritmo genético tradicional, que foi validado utilizando dados fictícios obtidos a partir de um gerador automático de projetos. Apesar de utilizar um algoritmo genético tradicional, os autores tratam dois objetivos: minimizar o tempo e o custo do projeto, que são combinados em uma única função objetivo, usando pesos diferentes para cada um dos objetivos. A formulação considera aspectos importantes, como habilidades dos recursos e ordem de precedências entre as tarefas. O trabalho apresenta algumas limitações em sua formulação, como a restrita participação do gerente, que se limita a parametrizar os dados do algoritmo, e a

impossibilidade de planejamento de multiprojetos.

Gueorguiev et al. (2009) apresentam uma formulação multiobjetivo baseada no algoritmo SPEA II, na qual a robustez e o tempo de conclusão do projeto são tratados como dois objetivos concorrentes, para resolver o problema de planejamento de projeto de software. O artigo introduz dois modelos diferentes de robustez, com relação a incertezas sobre as estimativas de custo de pacotes de trabalho, e incertezas relacionadas à correta contabilização de todos os pacotes de trabalho. O algoritmo proposto foi testado em quatro projetos reais de larga escala, e os resultados indicam o sucesso da abordagem proposta.

Colares (2010) utilizou um algoritmo genético multiobjetivo para resolver o problema de alocação de equipes e desenvolvimento de cronogramas. A função de aptidão aplicada busca minimizar o tempo total do projeto, o custo total, o atraso nas tarefas e as horas extras. A abordagem proposta é bem completa, tratando aspectos como habilidades dos recursos, curva de aprendizado e *overhead* de comunicação. Outro ponto positivo é a maior interação com o gerente de projetos, que pode incluir, na população inicial do algoritmo, soluções construídas por ele. Pode-se observar também algumas limitações no estudo, como o não tratamento de mudanças de pessoal, de alterações no cronograma e de erros de estimativas. O algoritmo foi avaliado utilizando dados reais de projetos de médio porte de uma empresa de desenvolvimento de software. Os resultados dos testes mostram a eficácia da abordagem, que obteve resultados comparáveis e melhores que os propostos por especialistas da área.

Rocha et al. (2011) utilizam o conceito "Data de Liberação", armazenando as datas de conclusão das atividades ou aumento de disponibilidade dos recursos. Com o uso desse conceito, junto com uma técnica para definição da primeira data de programação de uma atividade, os autores conseguiram diminuir o custo de processamento e permitir a utilização do minuto como unidade de tempo, tornando a modelagem do problema mais flexível. A abordagem utilizou um algoritmo genético multiobjetivo, com três funções objetivo que buscam minimizar a duração do projeto; minimizar o custo relativo a horas extras; e maximizar a qualidade das equipes alocadas, através de um somatório das habilidades dos recursos alocados para executar as tarefas do projeto. A abordagem foi testada em um projeto real e seus resultados comparados com soluções geradas por um gerente. O algoritmo proposto conseguiu gerar pelo menos um resultado superior ao planejado pelo gerente. Apesar dos pontos positivos, o trabalho apresenta algumas limitações em sua modelagem, não tratando o replanejamento do projeto; a produtividade individual; e as curvas de aprendizado da equipe.

Di Penta et al. (2011) apresentam duas modelagens para o problema de agendamento e alocação de equipes em projetos de software. A primeira modelagem é mono-objetivo e busca minimizar a duração do projeto. A segunda é multiobjetivo e busca minimizar a duração do projeto e reduzir a fragmentação do cronograma. A abordagem mono objetivo foi implementada, utilizando, para a solução, algoritmos genéticos, *Simulated Annealing* e *Hill Climbing*. Para validar a abordagem, foram realizados estudos empíricos utilizando dados reais de dois projetos. Os resultados mostram que tanto o *Simulated Annealing* quanto os algoritmos genéticos conseguem gerar soluções que podem ser valiosas para os gerentes de projeto no processo de tomada de decisão. A abordagem multiobjetivo foi modelada, mas não foi implementada e testada.

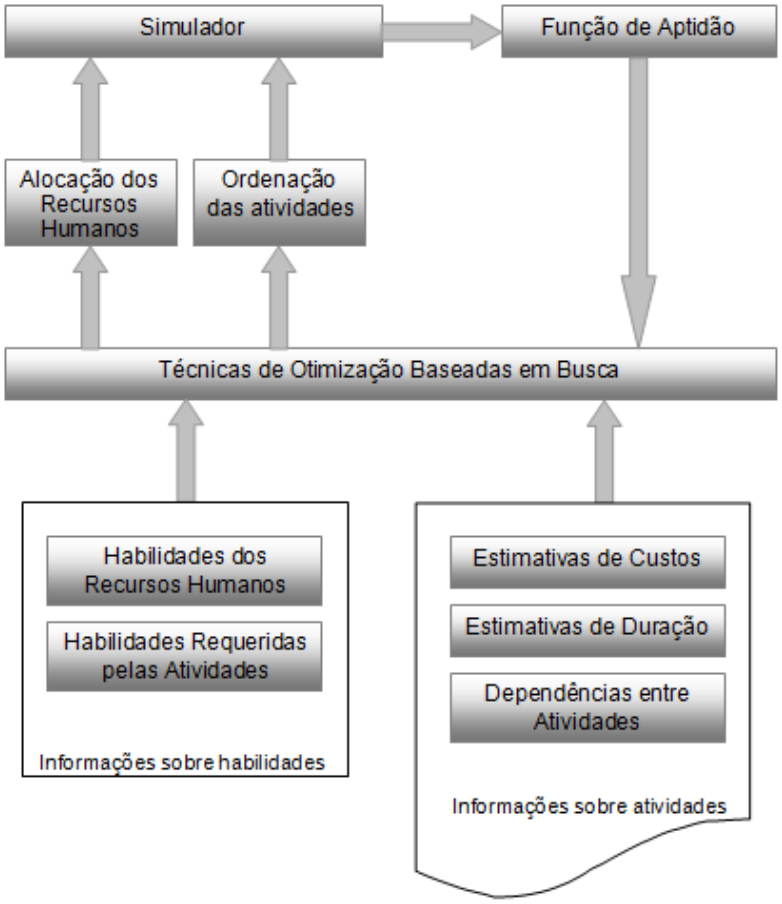

Figura 3.4: Esquema genérico de busca em gerenciamento de projeto. Fonte: Harman et al. (2009)

Barreto et al. (2008) propõem uma abordagem para o problema de alocação de recursos humanos em projetos de desenvolvimento de software. Os autores consideram características importantes de recursos humanos, como habilidades individuais, experiência, conhecimento, seu papel na organização e seu papel no projeto. Para realizar a otimização, é utilizado o algoritmo de otimização combinatória *Branch and Bound*. Uma das principais limitações dessa abordagem é a necessidade de se quebrar as tarefas pequenas o suficiente para que possam ser executadas por um único desenvolvedor, o que nem sempre é possível em um projeto real.

Minku et al. (2012) apresentam a primeira análise de tempo de execução para o Problema de Desenvolvimento de Cronogramas, definindo quais características podem transformar o Desenvolvimento de Cronogramas em um problema fácil ou difícil. Baseado nessa análise, é feita uma avaliação de desempenho dos Algoritmos Genéticos (AG) e é desenvolvido um novo Algoritmo Evolucionário (AE), incluindo a normalização da dedicação dos funcionários para diferentes tarefas e eliminando o problema de exceder a dedicação máxima diária. O AE criado para resolver o PDC é comparado com o estado da arte em Algoritmos Genéticos (AG) para a solução desse problema, em Alba e Chicano (2007)

Outros estudos de SBSE relacionados ao assunto merecem ser mencionados. Di

Penta et al. (2007) apresentam um estudo sobre os efeitos do *overhead* de comunicação em planejamento de projetos de software. Colares et al. (2009) apresentam uma abordagem multiobjetivo, que tem, como objetivos, maximizar a satisfação das partes interessadas no projeto e minimizar os riscos em planejamentos da próxima *release*. Britto et al. (2012) apresentam uma abordagem híbrida baseada na metaheurística multiobjetivo NSGAII e Sistemas de Inferência Fuzzy para resolver o problema de alocação de equipes em projetos ágeis.

A Figura 3.4 apresenta um esquema genérico de busca em gerenciamento de projeto. Essa abordagem é guiada pela simulação, que captura, de forma abstrata, o desenvolvimento de um projeto de acordo com um determinado plano. O plano do projeto é avaliado usando simulação. Normalmente, a simulação é um simples enfileiramento, que pode calcular deterministicamente propriedades do projeto (como, por exemplo, tempo de conclusão), com base em um plano. O plano envolve dois aspectos: pessoas e tarefas. As tarefas, também chamadas pacotes de trabalho, têm que ser finalizadas por uma equipe. Podem existir dependências entre tarefas, o que significa que uma não pode começar antes de outra terminar. Tarefas também podem exigir determinadas habilidades para sua execução, habilidades que alguns recursos humanos possuem, e outros não (Harman et al., 2009).

3.4 Visão Geral das Publicações em SBSE

A Tabela 3.1 apresenta um resumo da revisão bibliográfica descrita nas Seções 3.2 e 3.3, caracterizando os trabalhos citados de acordo com a área de Engenharia de Software e o tipo de problema abordado.

As Tabelas 3.2 e 3.3 apresentam um detalhamento das publicações realizadas na área de Gerenciamento de Projeto. Os trabalhos são classificados de acordo com o problema abordado, sua função objetivo (ou função de avaliação) e as técnicas de otimização utilizadas.

Harman et al. (2012) publicaram um importante artigo tutorial, que serve como um guia útil para leitores de publicações em SBSE. O principal objetivo do artigo é deixar o leitor apto a iniciar pesquisas em SBSE. O tutorial apresenta um guia passo a passo para desenvolver a formulação necessária, implementar, realizar experimentos, e coletar os resultados requeridos para realização de pesquisas.

Mais trabalhos publicados em SBSE podem ser encontrados no repositório do SEBASE (*Software Engineering by Automated Search*), acessado em `http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/`, mantido por Yuanyuan Zhang, do Centre for Research on Evolution, Search And Testing (CREST) do Department of Computer Science na University College London (UCL).

Tabela 3.1: Classificação das Publicações em SBSE

Áreas da Engenharia de Software	**Problema Abordado**	**Autores**
Testes de Software	Priorização de Casos de Teste	Li et al. (2007) Maia et al. (2008)
	Geração de Dados para Teste	Michael et al. (2001) Hermadi e Ahmed (2003) Khor e Grogono (2004) Harman (2007)
Engenharia de Requisitos	Próximo *Release*	Bagnall et al. (2001) Zhang et al. (2007) Colares et al. (2009)
Estimativa de Software	Estimativas de Tamanho	Dolado (2000)
	Estimativas de Custo	Dolado (2001) Aguilar-Ruiz et al. (2001)
Projeto de Software	Coesão e Acoplamento de Classes	Simons e Parmee (2008a) Simons e Parmee (2008b)
	Padrão Orientado a Objetos	O'Keeffe e Cinnéide (2004)
Manutenção de Software	Reengenharia e Manutenção	Harman (2006)
Gerenciamento de Projeto	Escalonamento de Tarefas e Alocação de Recursos Humanos	Chang et al. (1994) Chang et al. (1998) Chang et al. (2001) Chang et al. (2008) Alba e Chicano (2007) Gueorguiev et al. (2009) Colares (2010) Rocha et al. (2011) Di Penta et al. (2011) Minku et al. (2012)
	Alocação de Recursos Humanos	Barreto et al. (2008)
	Overhead de Comunicação	Di Penta et al. (2007)
	Planejamento de Projetos de Manutenção	Antoniol et al. (2004a) Antoniol et al. (2004b) Antoniol et al. (2005)
	Simulação de Projeto de Software	Aguilar-Ruiz et al. (2001)

Tabela 3.2: Detalhamento das Publicações na Área de Gerenciamento de Projetos - Parte 1

Problema Abordado	**Função Objetivo**	**Técnica de Otimização**	**Autores**
Escalonamento de Tarefas e Alocação de Recursos Humanos	Determinar a alocação dos recursos	AG Mono Objetivo	Chang et al. (1994)
	Minimizar o custo total e o tempo	AG Mono Objetivo	Chang et al. (1998) Alba e Chicano (2007)
	Minimizar o custo, a duração e a sobrecarga de trabalho	AG Mono Objetivo	Chang et al. (2001) Chang et al. (2008)
	Minimizar o tempo total, a diferença de tempo ao incluir uma nova tarefa, e a diferença de tempo ao aumentar a duração de uma tarefa	SPEA II Multi Objetivo	Gueorguiev et al. (2009)
	Minimizar o custo total, o atraso das tarefas e o tempo total do projeto	NSGA II Multi Objetivo	Colares (2010)
	Minimizar a duração do do projeto e o custo das horas-extras, e Maximizar a qualidade das equipes alocadas	NSGA II Multi Objetivo	Rocha et al. (2011)
	Minimizar o custo total e o tempo	NSGA II e Fuzzy Multi Objetivo	Minku et al. (2012)

Tabela 3.3: Detalhamento das Publicações na Área de Gerenciamento de Projetos - Parte 2

Problema Abordado	**Função Objetivo**	**Técnica de Otimização**	**Autores**
Escalonamento de Tarefas e Alocação de Recursos Humanos	Minimizar a Duração do Projeto	AG, HC e SA Mono Objetivo	Di Penta et al. (2011)
	Minimizar a Duração do Projeto e Reduzir a fragmentação do cronograma	NSGA II Multi Objetivo	Di Penta et al. (2011)
Alocação de Recursos Humanos	Maximizar o valor agregado para o projeto	*Branch and Bound*	Barreto et al. (2008)
		Otimização Combinatória	
Overhead de Comunicação	Maximizar o tempo de uso de recurso durante o ciclo de vida do projeto	AG Mono Objetivo	Di Penta et al. (2007)
Planejamento de Projetos de Manutenção	Minimizar a duração do Projeto	AG e Teoria de Simulação de Filas Mono Objetivo	Antoniol et al. (2004a)
	Minimizar a duração do Projeto	AG, HC e SA Mono Objetivo	Antoniol et al. (2004b)
	Minimizar a duração do Projeto	AG, HC, SA e *Random Search* Mono Objetivo	Antoniol et al. (2005)
Simulação de Projeto de Software	Maximizar o percentual de classificação e compreensão das regras	Algoritmo Evolucionário Mono Objetivo	Aguilar-Ruiz et al. (2001)

Capítulo 4

Heurísticas

Este capítulo apresenta uma introdução sobre os principais métodos heurísticos da literatura para a resolução de problemas de otimização combinatória, com ênfase nos algoritmos de busca local, foco deste trabalho. Além disso, apresenta uma breve revisão sobre algoritmos multiobjetivo.

4.1 Introdução às Heurísticas

Diversos problemas práticos podem ser modelados como problemas de otimização matemática, definindo-se uma função objetivo que se deseja otimizar, sujeita a um conjunto de restrições. Em geral pode-se definir este problema da seguinte maneira:

$$\begin{array}{ll} \min & f(x) \\ s.a. & x \in S \end{array}$$

em que x representa uma proposta de solução, $f(x)$ a função objetivo e S o espaço de soluções factíveis para o problema de interesse.

A Figura 4.1 representa uma solução hipotética para um problema de otimização, sendo representados os ótimos locais e o ótimo global da solução. Neste problema, deseja-se minimizar a função $f(x)$. Na Figura 4.1, o ótimo global é o menor valor assumido pela função objetivo, ou seja, a solução ótima do problema. Já os ótimos locais são os pontos de mínimo encontrados em alguma região do espaço de busca de soluções.

Em casos reais, grande parte destes problemas não pode ser resolvido utilizando métodos exatos de otimização. Mesmo com o atual avanço tecnológico, processadores cada vez mais rápidos e técnicas de processamento paralelo, muitos destes problemas possuem alta complexidade matemática e apresentam o fenômeno da explosão combinatorial, ou seja, a quantidade de soluções possíveis é tão grande que se torna inviável a exploração de todas elas. Estes problemas são geralmente classificados como NP-Difíceis, o que significa que não existe um algoritmo que seja capaz de obter a solução ótima em tempo polinomial.

Para tentar resolver esse tipo de situação, ao longo do tempo foram desenvolvidos diversos métodos que buscam encontrar soluções ótimas ou de boa qualidade com o menor esforço computacional possível, sem a necessidade de se realizar buscas exaustivas dentro do espaço de soluções factíveis. Esses métodos são heurísticas,

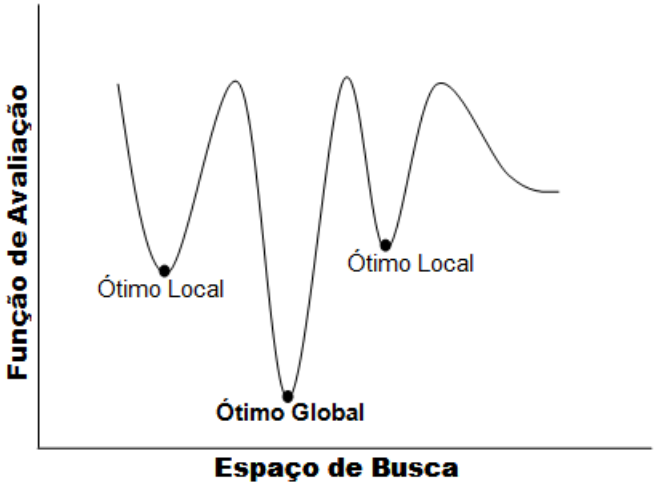

Figura 4.1: Representação hipotética de um problema de otimização.

que têm como objetivo encontrar soluções para problemas combinatoriais complexos com esforços computacionais razoáveis.

As heurísticas podem ser classificadas em heurísticas construtivas, heurísticas de refinamento e metaheurísticas.

4.1.1 Heurísticas construtivas

Esses algoritmos constroem uma solução viável para um dado problema de forma incremental, de acordo com a função de avaliação escolhida para o problema em questão. O elemento escolhido em cada passo é, em geral, o melhor candidato de acordo com algum critério. A execução do método só se encerra quando todos os elementos candidatos forem analisados (Lopes et al., 2013). O Algoritmo 1 apresenta o pseudocódigo geral das heurísticas construtivas.

Nas heurísticas clássicas, geralmente os elementos da solução são ordenados por uma função gulosa, que avalia a melhora trazida por cada elemento. Os elementos também podem ser escolhidos de forma aleatória, sendo este o tipo mais simples de construção.

As construções gulosas apresentam, como principais desvantagens, a impossibilidade de alterar a escolha de um elemento ruim e a pequena diversidade das soluções geradas, mesmo quando utilizadas regras para conduzir a procura. Essas desvantagens podem ser minimizadas com a utilização de uma componente aleatória na construção.

As heurísticas construtivas (gulosas e aleatórias) geralmente não encontram soluções de boa qualidade para problema difíceis, necessitando, assim, de métodos de refinamento. Esses métodos são discutidos na Subseção 4.1.2.

4.1.2 Heurísticas de Refinamento

As heurísticas de refinamento, ou técnicas de busca local, são baseadas na noção de vizinhança, e tentam melhorar uma dada solução. Esses métodos partem de

Algoritmo 1: Heurística Construtiva

```
   Saída: s
1  início
2  |  s ← ∅;
3  |  Inicializar o conjunto de elementos candidatos C
4  |  enquanto (C ≠ ∅) faça
5  |  |  e ← escolha um elemento do conjunto C
6  |  |  s = s ∪ e
7  |  |  Atualize o conjunto de elementos candidatos C
8  |  fim
9  |  Retorne s
10 fim
```

uma solução inicial qualquer, e caminham, em cada iteração, de vizinho para vizinho, de acordo com a definição de vizinhança adotada, tentando melhorar a solução construída (Lopes et al., 2013).

Uma estrutura de vizinhança é uma função N que associa, a cada solução $s \in S$, um conjunto de vizinhos $N(S) \subseteq S$. A função N é definida de acordo com o problema estudado e cada solução $s' \in N(s)$ é denominada vizinho de s.

Uma solução s_0 faz parte da vizinhança da solução s se e somente se s_0 for resultado de uma modificação em s, causada por um determinado movimento m, de tal maneira que continue a fazer parte do conjunto de soluções factíveis. Estes movimentos não possuem regras definidas a serem aplicadas em todos os problemas.

Dessa forma, as heurísticas de refinamento utilizam o conceito de estrutura de vizinhança para explorar o espaço de soluções do problema de otimização na busca de melhores soluções.

4.2 Metaheurísticas

A principal diferença das metaheurísticas para as heurísticas de construção e refinamento é sua capacidade de escapar dos ótimos locais. Isso é feito através de estratégias que permitem uma melhor exploração do espaço de busca.

As metaheurísticas partem de uma solução inicial, que pode ser factível ou infactível. Utilizando estratégias de transição, que variam em cada tipo de metaheurística, passam de uma solução para outra solução vizinha, até que um critério de parada preestabelecido. Estas estratégias, em geral, conseguem evitar que o algoritmo fique preso nos ótimos locais, de forma a explorar melhor o espaço de busca, obtendo soluções de excelente qualidade ou, dependendo do problema, encontrar a solução ótima global. Uma revisão sobre metaheurísticas pode ser encontrada em Blum e Roli (2013).

As metaheurísticas são divididas em dois grupos. O primeiro grupo é composto pelas metaheurísticas que exploram o espaço de busca utilizando estratégias de busca local, e o segundo que explora o espaço de busca utilizando busca populacional.

As metaheurísticas baseadas em busca populacional são inspiradas nos mecanismos encontrados na natureza. Consistem em manter um conjunto de boas soluções

e combiná-las, de forma a tentar produzir soluções ainda melhores. Nessa categoria, se enquadram as metaheurísticas Algoritmos Genéticos (AG), Algoritmos Meméticos (AM), *Ant Colony System* (ACS) e *Particle Swarm Optimization* (PSO).

Nas metaheurísticas baseadas em busca local, a exploração é realizada através da aplicação de movimentos à solução corrente, que levam o algoritmo a uma solução promissora de sua vizinhança (Lopes et al., 2013). Pertencem a essa categoria as metaheurísticas *Variable Neighborhood Search* (VNS), *Greedy Randomized Adaptative Search Procedure* (GRASP), *Simulated Annealing* (SA) e *Iterated Local Search* (ILS).

4.2.1 Metaheurística VNS

A metaheurística mono-objetivo *Variable Neighborhood Search* (VNS) foi proposta em 1997 por Hansen e Mladenovic (1997) e tem se mostrado eficiente e de fácil adaptação para muitos problemas de otimização. Ela se baseia em mudanças sistemáticas da estrutura de vizinhança no processo de busca local.

Ela explora vizinhanças diferentes da solução corrente e focaliza a busca em torno de uma nova solução se e somente se um movimento de melhora é realizado (Dutra e Montane, 2010). Como consequência, caso o algoritmo encontre a solução ótima global (no caso de problemas convexos ou quando é conhecido o ótimo global), a busca fica estagnada nesse ponto. O Algoritmo 2 apresenta o pseudocódigo metaheurística VNS.

Algoritmo 2: VNS

```
   Saída: s
1  início
2      Seja s_0 uma solução inicial
3      Seja r o número de diferentes estruturas de vizinhança
4      s ← s_0
5      enquanto (CondicaodeParada = falso) faça
6          K ← 1
7          enquanto (K ≤ r) faça
8              Gere um vizinho qualquer s' ∈ N^k(s)
9              s'' ← BuscaLocal(s')
10             se f(s'') < f(s) então
11                 s ← s''
12                 K ← 1
13             senão
14                 K ← K + 1
15             fim
16         fim
17     fim
18     Retorne s
19 fim
```

Segundo Lopes et al. (2013), a estratégia do algoritmo VNS foi inspirada em três fatos:

(i) um mínimo local em relação a uma estrutura de vizinhança não é necessariamente um mínimo local em relação a todas as outras estruturas de vizinhança;

(ii) um mínimo global é um mínimo local em relação a todas as possíveis estruturas de vizinhança;

(iii) para muitos problemas, mínimos locais em relação a uma ou várias estruturas de vizinhanças são relativamente próximos uns dos outros.

Os fatos acima sugerem o uso de várias estruturas de vizinhança durante o processo de buscas locais para resolver um problema de otimização. A idéia, então, é definir um conjunto de estruturas de vizinhanças e a forma como serão utilizadas, seja de forma determinística, aleatória ou ambas, para a formulação do VNS. Tais decisões produzirão desempenhos diferentes do algoritmo.

Quando o algoritmo VNS encontra um ponto de ótimo local da região em que se encontra o ótimo global, esse tem grandes chances de encontrar este ótimo global. Entretanto, caso a solução ótima global esteja em outra região, a única possibilidade de encontrá-la é utilizando uma estratégia de diversificação. Sendo assim, para uma metaheurística pode ser muito importante um equilíbrio entre intensificação e diversificação em seu processo de busca.

Outro aspecto importante da lógica de implementação do VNS está relacionado com a qualidade de um ótimo local. Não necessariamente um ponto de ótimo local com uma função objetivo de melhor qualidade pode ser mais adequado para encontrar um ponto de ótimo global. Assim, a solução ótima local mais adequada será aquela que possuir características mais próximas da solução ótima global. Entretanto, na grande maioria dos problemas de otimização não conhecemos a solução ótima.

A seguir, será apresentado o método de busca local *Variable Neighborhood Descent* (VND), que é utilizado na implementação original do VNS.

4.2.2 VND

O método *Variable Neighborhood Descent* (VND) ou Máxima Descida em Vizinhança Variável envolve também a substituição da solução atual pelo resultado da busca local. Quando há uma melhora, porém, a estrutura de vizinhança é trocada de forma determinística, cada vez que se encontra um mínimo local.

A solução resultante é um mínimo local em relação a todas as K_{max} estruturas de vizinhança exploradas. A característica principal do VND é sua monotonicidade, o que, dependendo da solução inicial, das características da instância do problema, e das estruturas de vizinhança, frequentemente limita-se a uma pequena região do espaço de busca (Maciel et al., 2005).

O Algoritmo 3 apresenta o pseudocódigo da metaheurística VND.

4.2.3 GRASP

A metaheurística *Greedy Randomized Adaptative Search Procedure* (GRASP), apresentada em Feo e Resende (1995), consiste na aplicação iterativa de duas fases (construção e refinamento), retornando a melhor das soluções obtidas ao longo da busca. O Algoritmo 4 apresenta o pseudocódigo da metaheurística GRASP. As duas fases são discutidas a seguir.

Algoritmo 3: VND

```
   Saída: s
1  início
2  |  Seja r o número de diferentes estruturas de vizinhança
3  |  K ← 1
4  |  enquanto (K ≤ r) faça
5  |  |  Encontre um vizinho s' ∈ N^k(s)
6  |  |  s'' ← BuscaLocal(s')
7  |  |  se f(s'') < f(s) então
8  |  |  |  s ← s''
9  |  |  |  K ← 1
10 |  |  senão
11 |  |  |  K ← K + 1
12 |  |  fim
13 |  fim
14 |  Retorne s
15 fim
```

Algoritmo 4: Pseudo-Código GRASP

```
   Entrada: graspMax
   Saída: s
1  início
2  |  f* ← ∞
3  |  para (i → graspMax)) faça
4  |  |  ConstruçãoSolucaoInicial()
5  |  |  Busca Local()
6  |  |  se f(s) < f* então
7  |  |  |  s* ← s
8  |  |  |  f* ← f(s)
9  |  |  fim
10 |  fim
11 |  s ← s*
12 |  Retorne s
13 fim
```

4.2.3.1 Fase de Construção

Na fase de construção do GRASP, uma solução viável é construída iterativamente, um elemento da solução por vez, até que a solução esteja completa. Os elementos candidatos que compõem a solução são ordenados em uma lista, chamada de lista de candidatos. Esta lista é ordenada por uma função gulosa que mede o benefício que o elemento escolhido mais recentemente concede à parte da solução já construída. Um subconjunto denominado Lista Restrita de Candidatos (LRC) é formado pelos melhores elementos que compõem a lista de candidatos (Feo e Resende, 1995).

O tamanho da LRC é controlado por um parâmetro real $\alpha \in [0, 1]$. Para $\alpha = 1$,

tem-se uma solução totalmente aleatória; para $\alpha = 0$, tem-se uma solução gulosa. A componente probabilística do método é devida à escolha aleatória de um elemento da LRC. Este procedimento permite que diferentes soluções de boa qualidade sejam geradas.

A média e a variância do valor de função objetivo das soluções construídas são diretamente afetadas por este parâmetro: se o número de elementos da lista LRC (dado por $|LRC|$) for pequeno, menor a variância, menor o espaço de soluções percorrido e maior a chance de aprisionar a busca local em um ótimo local pobre; se $|LRC|$ for grande, maior a variância, menor a possibilidade de prisão em ótimo local pobre, porém, maior número de iterações com soluções sub-ótimas e maiores as quantidades de soluções, a serem exploradas, até que um mínimo local seja alcançado.

Para a aplicação eficaz do método é necessária, portanto, a definição de um intervalo de valores para $|LRC|$ de forma a balancear a relação entre qualidade das soluções, quantidade de iterações necessárias e espaço de busca explorado (Feo e Resende, 1995).

4.2.3.2 Fase de Busca Local

Na fase de busca local do GRASP, é aproveitada a solução inicial da fase de construção e explorada a vizinhança ao redor desta solução. Se um melhoramento é encontrado, a solução corrente é atualizada e novamente a vizinhança ao redor da nova solução é pesquisada. O processo se repete até que nenhum melhoramento é encontrado.

4.3 Otimização Multiobjetivo

Problemas de otimização multiobjetivo buscam minimizar e/ou maximizar simultaneamente um conjunto de objetivos satisfazendo um conjunto de restrições. Em otimização multiobjetivo, não existe uma única solução ótima para cada um dos objetivos, mas sim um conjunto de soluções eficientes, ou soluções Pareto-ótimas, no qual nenhuma solução é melhor que outra solução para todos os objetivos. É sempre necessário analisar o conjunto Pareto-Ótimas e escolher uma solução eficiente que satisfaça os objetivos globais do problema.

Nesta seção são apresentados os conceitos básicos usados neste tipo de problema.

4.3.1 Problemas de Otimização Multiobjetivo

A formulação geral de um problema de otimização multiobjetivo consiste em encontrar um conjunto de soluções que satisfaça as restrições e otimize uma função vetorial, cujos elementos representam as funções objetivos. Os objetivos considerados, geralmente, são conflitantes entre si. Neste contexto, portanto, o termo "otimizar" significa encontrar soluções nas quais os valores de todos objetivos não podem ser melhorados simultaneamente.

Um problema de otimização multiobjetivo pode ser formulado da seguinte forma:

$$\min ou \max \quad \{f_1(x) = z_1, f_2(x), \ldots, f_n(x) = z_n\} \tag{4.1}$$
$$\text{sujeito a} \quad x \in X^* \tag{4.2}$$

em que a solução $x = [x_1, x_2, ..., x_I]$ é um vetor com as variáveis de decisão e X^* é o conjunto de soluções factíveis presentes no espaço de decisões X.

A imagem da solução x no espaço objetivo factível, denotado por Z^*, é um ponto $z^x = [z_1^x, z_2^x, ..., z_n^x] = f(x)$, tal que $z_j^x = f_j(x)$, $j = 1, ..., n$.

Na otimização mono-objetivo, o espaço dos objetivos é inteiramente ordenado, ou seja, dados dois elementos $x \in X^*$ e $y \in X^*$, existem apenas duas possibilidades: ou $f(x) \geq f(y)$ ou $f(x) \leq f(y)$.

Na otimização multiobjetivo, no entanto, não existe uma solução que seja ótima para todos os objetivos. Pode ocorrer, por exemplo, que a minimização de um determinado objetivo provoque um aumento nos outros objetivos.

Apesar do espaço objetivo não ser, em geral, completamente ordenado, é possível ordená-lo parcialmente (Pareto, 1896). Essa ordenação parcial permite a distinção entre as soluções de um problema de otimização multiobjetivo. Em um conjunto parcialmente ordenado, dadas duas soluções $\vec{x}$ e $\vec{y} \in X^*$, existem três possibilidades:

(i) ou $f(\vec{x}) \leq f(\vec{y})$

(ii) ou $f(\vec{y}) \leq f(\vec{x})$

(iii) ou ($f(\vec{x}) \nleq f(\vec{y})$ e $f(\vec{y}) \nleq f(\vec{x})$).

Com a ordenação parcial, surge o conceito de dominância. Uma solução $\vec{x}$ domina uma solução $\vec{y}$ se:

(i) A solução $\vec{x}$ não for pior que a solução $\vec{y}$ em nenhum dos objetivos, ou seja, $f_m(\vec{x}) \leq f_m(\vec{y})$, para $m = 1, ..., M$, em que M é número de objetivos, e;

(ii) A solução $\vec{x}$ for estritamente melhor que a solução $\vec{y}$ em pelo menos um objetivo, ou seja, $f_m(\vec{x}) < f_m(\vec{y})$ para algum $m \in \{1, ..., M\}$.

Assim, uma solução eficiente, ou Pareto-ótima, é uma solução não-dominada por nenhuma outra solução factível. O conjunto de todas as soluções não-dominadas, dentre as soluções factíveis, é chamado de Conjunto Pareto-ótimo. Os pontos no espaço das funções-objetivo que correspondem ao conjunto Pareto-ótimo formam a Fronteira de Pareto. Na Figura 4.2 é ilustrado o conjunto de soluções factíveis, o espaço objetivo factível e o grau de dominância em um problema de minimização com dois objetivos.

Caso não exista diferença na relevância relativa entre os objetivos a serem atendidos, todos os pontos na Fronteira de Pareto são qualitativamente equivalentes, sob a perspectiva de otimização. Isso implica que problemas multiobjetivo podem apresentar infinitas soluções equivalentes. Esse fato exige o desenvolvimento de algoritmos de busca capazes de identificar e amostrar da melhor maneira possível a Fronteira de Pareto, dada uma quantidade finita de recursos computacionais.

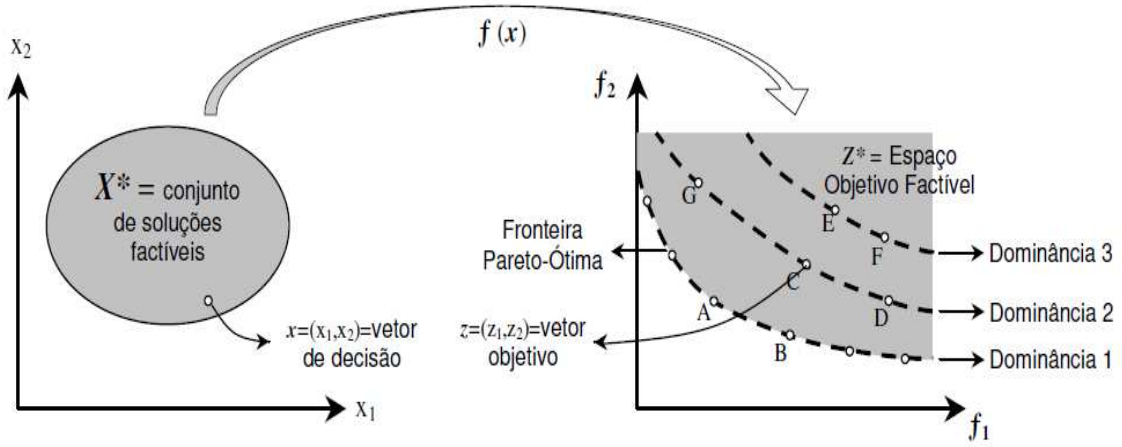

Figura 4.2: Espaço de Decisões e Espaço Objetivo Factível de um problema de minimização com dois objetivos. Fonte: Hashimoto (2004)

Obter a amostra ótima implica em encontrar e manter soluções não-dominadas que se distribuam uniformemente por toda a fronteira de Pareto.

As fronteiras de Pareto podem apresentar diversas conformações, incluindo descontinuidade. Além disso, durante o processo de busca, o fato de uma solução ser não-dominada frente às propostas de solução já investigadas não implica que a mesma pertença à fronteira de Pareto. Assim sendo, o processo de busca terá sempre dois objetivos principais:

(i) Convergir para a Fronteira de Pareto;

(ii) Manter uma distribuição tão uniforme quanto possível das soluções não-dominadas.

A Figura 4.3 representa graficamente todos os conceitos de uma Fronteira de Pareto.

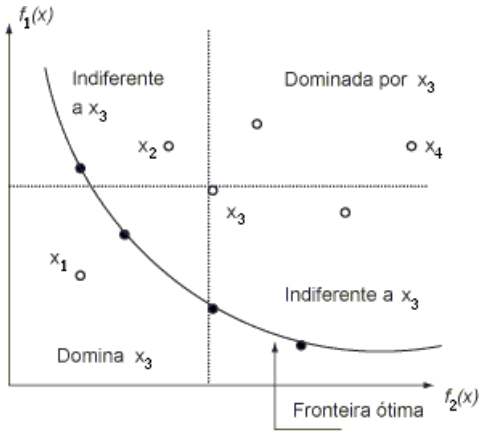

Figura 4.3: Dominância de Pareto. Fonte: Hashimoto (2004)

A resolução de problemas multiobjetivo é dividida, basicamente, em duas etapas: determinação das soluções eficientes e a etapa de decisão. A primeira etapa consiste

na busca de soluções Pareto-ótimas dentro do espaço factível. O segundo aspecto, que envolve um procedimento chamado de decisor, diz respeito à seleção da solução, que é um compromisso final dentre aquelas de Pareto. Nessa fase, o decisor faz uma escolha externa ao processo de otimização. Dependendo de como e quando o processo de otimização e a etapa de decisão são combinados, os métodos de resolução podem ser classificados em três categorias:

- Decisão antes do processo de busca (*a priori*): o decisor opta pelo compromisso que ele quer obter antes de lançar o método de resolução. Basicamente, o que se faz é transformar um problema multiobjetivo em uma aproximação mono-objetivo. Após esta transformação, pode-se aplicar qualquer técnica mono-objetivo para a resolução do problema. Entretanto, a adequação dos pesos atribuídos a cada objetivo não é trivial, principalmente quando os objetivos são extremamente conflitantes.

- Decisão durante o processo de busca (progressivo): é o procedimento de decidir durante o processo de obtenção das soluções não-dominadas. A escolha do decisor é utilizada na busca de novas soluções eficientes. Esta abordagem exige certa experiência do decisor, já que as escolhas deverão ser tomadas de modo a orientar o processo de otimização no sentido de formar a fronteira Pareto-ótima.

- Decisão após o processo de busca (*a posteriori*): é considerada a opção mais correta, pois as escolhas serão feitas de acordo com os resultados encontrados. Ou seja, com o conjunto Pareto-ótimo definido, torna-se possível conhecer o comportamento do problema em relação aos objetivos em questão. Conhecendo-se as relações de dependência entre eles, a escolha torna-se mais fácil.

A qualificação das metodologias de busca de soluções eficientes depende do conhecimento de algumas dificuldades usualmente encontradas nos problemas multiobjetivo.

Assim como na otimização mono-objetivo, as dificuldades de resolução de problemas multiobjetivo se devem à presença de restrições e do comportamento das funções objetivo. As principais dificuldades encontradas na otimização de problemas multiobjetivo são: convexidade, descontinuidades e multimodalidade (múltiplos ótimos locais ou globais). Além destas dificuldades, pode-se citar a não uniformidade das soluções no espaço dos objetivos. Certos métodos de resolução podem apresentar características que concentram as soluções em determinadas áreas. Se estas regiões não forem próximas às soluções Pareto ou se elas contemplarem apenas uma parte do fronteira ótima, a caracterização de todo o conjunto Pareto-ótimo pode ser comprometida.

O objetivo desta pesquisa é desenvolver um algoritmo multiobjetivo baseado em buscas para solucionar o problema proposto. Por essa razão, na seção 4.4 é detalhada a metaheurística multiobjetivo VNS.

4.4 Metaheurística Multiobjetivo VNS (MOVNS)

Uma das primeiras aplicações da metaheurística VNS, já descrita na seção 4.2.1, para resolver problemas de otimização multiobjetivo foi proposta por Geiger (2008).

Algoritmo 5: MOVNS

Saída: s
início
  $s \leftarrow$ *Construir uma solução inicial aleatória*
  $D \leftarrow s$*//Conjunto de soluções dominantes*
  Definir um conjunto de t vizinhanças $V_1, V_2, ..., V_t$
  enquanto $(CondicaodeParada = falso)$ **faça**
    $s \leftarrow$ *Escolher uma solução de D tal Marque(s)* $= falso$
    Marque(s) = verdadeiro //Solução Visitada
    $V \leftarrow$ *Escolher aleatoriamente uma vizinhança*
    Determine aleatoriamente uma solução vizinha $s' \in V(s)$ *//shaking*
    para cada *cada vizinho* $s'' \in V(s')$ **faça**
      $D \leftarrow$ *Conjunto de soluções dominantes de* $D \cup s''$
    fim
  fim
  Retorne s
fim

Neste artigo, o autor propõe o algoritmo denominado *Multiobjective Variable Neighborhood Search* (MOVNS) para resolver um problema de sequenciamento de tarefas *flowshop* multiobjetivo.

O objetivo do MOVNS é determinar um conjunto D de soluções dominantes (uma aproximação do conjunto Pareto-ótimo). Inicialmente, é gerada uma solução aleatória s, que fará parte do conjunto de soluções dominantes D. Define-se, também, t estruturas de vizinhanças $\{V_1, V_2, ..., V_{|t|}\}$, sendo $V_k(s)$ a vizinhança k da solução s, para $1 \leq k \leq t$.

A cada iteração, é escolhida uma solução s do conjunto D e uma estrutura de vizinhança V_k. A solução escolhida é marcada como visitada, não podendo ser selecionada nas próximas iterações, caso permaneça no conjunto D. Se a condição de parada do algoritmo não tiver sido satisfeita e todas as solução estiverem selecionadas, então todas as soluções serão desmarcadas.

Em cada iteração, a solução s é perturbada por um movimento aleatório dentro da vizinhança V, originando uma solução s'. A partir de s', serão geradas todas as soluções vizinhas. Em seguida, o conjunto D das soluções dominantes é atualizado com as soluções vizinhas s'' obtidas. Esse processo é repetido até que a condição de parada seja satisfeita, retornando, por fim, o conjunto D, obtido após todo o processo de busca. O pseudocódigo da metaheurística MOVNS é apresentado no Algoritmo 5.

4.5 NSGA-II

O algoritmo de otimização multiobjetivo *Nondominated Sorting Genetic Algorithm* II (NSGA-II), proposto por Deb et al. (2002), é resultado de três alterações no algoritmo NSGA, proposto inicialmente por Srinivas e Deb (1994). As três mudanças implementadas foram: aperfeiçoamento do processo de ordenação das soluções não-dominadas, adição de elitismo e a eliminação da necessidade do parâmetro σ_{share} para incrementar a variedade da população.

Algoritmo 6: *Fast Non-Dominated Sorting*

Entrada: P

```
1  para cada p ∈ P faça
2  |  S_p ← ∅ ;
3  |  n_p = 0 ;
4  |  para cada q ∈ P faça
5  |  |  se (p ≺ q) então
6  |  |  |  S_p ← S_p ∪ {q} ;
7  |  |  fim
8  |  |  senão se (q ≺ p) então
9  |  |  |  n_p ← n_p + 1 ;
10 |  |  fim
11 |  fim
12 |  se (n_p = 0) então
13 |  |  F ← F ∪ {p} ;
14 |  fim
15 fim
16 i ← 1 ;
17 enquanto F ≠ ∅ faça
18 |  Q ← ∅ ;
19 |  para cada p ∈ F_i faça
20 |  |  para cada q ∈ S_p faça
21 |  |  |  n_q ← n_q − 1 ;
22 |  |  |  se (n_q = 0) então
23 |  |  |  |  Q ← Q ∪ q ;
24 |  |  |  fim
25 |  |  fim
26 |  fim
27 |  i ← i + 1 ;
28 |  F_i ← Q ;
29 fim
30 retorna F ;
```

4.5.1 *Fast Non-Dominated Sorting*

O NSGA-II utiliza o método *Fast Non-Dominated Sorting* para ordenar uma população, que tem complexidade $O(MN^2)$, sendo M o número de objetivos e N o tamanho da população. Este método recebe, como parâmetro de entrada, uma população P e retorna um conjunto de frentes não-dominadas $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k)$.

O Algoritmo 6 apresenta o pseudocódigo do método *Fast Non-Dominated Sorting*, que compara todas as possíveis soluções p e q presentes na população P, sendo p diferente de q. Caso p domine q, então a solução q é inserida no conjunto S_p; caso q domine p, então n_p é incrementado em uma unidade. Na linha 6, verifica-se o valor de n_p: caso ele seja igual a zero, então a solução p é inserida na primeira fronteira $\mathcal{F}_1$ pois ela não é dominada por nenhuma outra solução. Em seguida, cria-se um novo

Algoritmo 7: *Crowding Distance*

Entrada: $\mathcal{I}$
1 $l \leftarrow |\mathcal{I}|$;
2 **para cada** $i \in \mathcal{I}$ **faça**
3 | $\mathcal{I}[i] \leftarrow 0$;
4 **fim**
5 **para cada** objetivo m **faça**
6 | $\mathcal{I} \leftarrow ordenar(\mathcal{I}, m)$;
7 | $\mathcal{I}[1].distance \leftarrow \infty$;
8 | $\mathcal{I}[l].distance \leftarrow \infty$;
9 | **para** $i = 2$ to $(l-1)$ **faça**
10 | | $\mathcal{I}[i].distance \leftarrow \mathcal{I}[i].distance + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)/(f_m^{\max} - f_m^{\min})$;
11 | **fim**
12 **fim**
13 **retorna** $\mathcal{I}$;

laço, linhas 17 'a 29, para gerar as demais fronteiras. Nesse laço, percorre-se cada solução p da fronteira $\mathcal{F}_i$ e todas as q soluções de S_p, realizando um decremento em n_q. Caso o valor de n_q alcance o valor zero, então a solução q é incluída em Q. Após percorrer todas as soluções de $\mathcal{F}_i$, o valor de i é incrementado e atribui-se Q para a fronteira $\mathcal{F}_i$. Esse laço é repetido enquanto for possível gerar uma nova fronteira para as soluções de P.

4.5.2 *Crowding Distance*

Para estimar a densidade de soluções em volta de um certo ponto A da população, é calculada a distância média entre os dois pontos adjacentes de A, para cada um dos objetivos. A medida *distance* para um ponto i serve como uma estimativa do perímetro do maior cubóide que envolve este ponto, sem incluir nenhum outro ponto da população. Soluções localizadas em regiões com menor número de pontos recebem um valor maior do que soluções localizadas em regiões com maior número de pontos em relação ao espaço de objetivos.

O algoritmo que calcula o valor para a *Crowding Distance* é apresentado pelo Algoritmo 7.

O Algoritmo 7 recebe, como entrada, um conjunto de soluções $\mathcal{I}$. Este conjunto tem tamanho l e, inicialmente, é atribuído o valor zero para a *crowding distance* de todos os indivíduos de $\mathcal{I}$. A cada iteração do laço mais externo (linhas 5 à 12):

(i) ordenam-se as soluções do conjunto $\mathcal{I}$, linha 6, considerando cada objetivo m;

(ii) atribui-se infinito para o valor da *crowding distance* da primeira solução, linha 7, e da última solução de $\mathcal{I}$, linha 8.

Em seguida, o algoritmo entra no laço mais interno (linhas 9 à 11), no qual é feito o cálculo da *crowding distance* para cada solução i, linha 10, com i variando de 2 a $l-1$. O laço mais externo é repetido para os m objetivos. O método retorna o valor da *crowding distance* para todas as soluções do conjunto $\mathcal{I}$.

Algoritmo 8: NSGA-II

Entrada: pop, t_c, t_m, *critério de parada*

```
1  ND ← Solução Inicial();
2  P_0 ← Solução Inicial Aleatória(pop, ND) ;
3  Q_0 ← ∅ ;
4  t ← 0 ;
5  enquanto critério de parada não satisfeito faça
6  |  R_t ← P_t ∪ Q_t ;
7  |  F ← Fast non-dominated sorting(R_t) ;
8  |  P_{t+1} ← ∅ ;
9  |  i ← 1 ;
10 |  enquanto |P_{t+1}| + |F_i| ≤ N faça
11 |  |  Atribuir Crowding Distance(F_i) ;
12 |  |  P_{t+1} ← P_{t+1} ∪ F_i ;
13 |  |  i ← i + 1 ;
14 |  fim
15 |  se |P_{t+1}| < N então
16 |  |  Ordenar(F_i, ≺_n);
17 |  |  j = 1 ;
18 |  |  enquanto |P_{t+1}| < N faça
19 |  |  |  P_{t+1} ← P_{t+1} ∪ F_i[j] ;
20 |  |  |  j ← j + 1 ;
21 |  |  fim
22 |  fim
23 |  Q_{t+1} ← Gerar Indivíduos da Próxima Geração(P_{t+1}, t_c, t_m) ;
24 |  t ← t + 1 ;
25 fim
26 ND ← soluções não-dominadas obtidas de P_t ;
27 retorna ND ;
```

No Algoritmo 8 apresenta-se o pseudocódigo do algoritmo NSGA-II.

O algoritmo NSGA-II parte de um população inicial P_0, linha 2, de tamanho *pop*, em que *pop* é um parâmetro de entrada. A cada iteração do laço principal (linhas 5 à 25), as populações de pais P_t e de filhos Q_t se unem formando a população R_t, linha 6. O método *fast non-dominated sorting* é aplicado em R_t, linha 7, para dividir a população em conjuntos não-dominados, denominados fronteiras, $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k$, em que $\mathcal{F}_i$ domina $\mathcal{F}_j$ se e somente se $i < j$ e $R_t = \mathcal{F}_1 \cup \mathcal{F}_2 \ldots \mathcal{F}_k$. Em seguida, selecionam-se as i melhores fronteiras de $\mathcal{F}$, com o objetivo de formar a população P_{t+1}. Caso as fronteiras selecionadas não formem um conjunto de tamanho N, ordenam-se os indivíduos da fronteira $i + 1$, de acordo com *crowding distance*, e escolhem-se os $N - |P_{t+1}|$ primeiros para a população P_{t+1}. Na linha 23, uma população filha Q_t é criada a partir da aplicação de operadores de cruzamento e de mutação em uma população P_{t+1}.

4.6 Métricas de Avaliação de Desempenho

Realizar comparações entre dois conjuntos de pontos não-dominados, A e B, obtidos respectivamente por dois algoritmos de otimização multiobjetivo não é uma tarefa simples. Existem hoje na literatura diversas propostas de métricas para avaliação de desempenho de algoritmos multiobjetivo (Hansen e Jaszkiewicz, 1998; Zitzler e Thiele, 1999). No entanto, estas métricas devem ser escolhidas de forma adequada para realizar comparações justas entre os algoritmos. Neste trabalho, são utilizadas três métricas de avaliação de desempenho denominadas cardinalidade (Hansen e Jaszkiewicz, 1998), hipervolume (Zitzler e Thiele, 1999) e epsilon Zitzler et al. (2005).

A qualidade do conjunto A de pontos não-dominados obtido por um algoritmo, para uma determinada instância, é sempre avaliada com relação ao conjunto constituído por todos os pontos não-dominados encontrados em todos os experimentos realizados. Este conjunto é denominado conjunto de pontos de referência REF.

As subseções 4.6.1, 4.6.2 e 4.6.3 a seguir apresentam as definições das métricas utilizadas.

4.6.1 Métrica de Cardinalidade

A métrica de cardinalidade $C1_{REF}$ mensura a quantidade de soluções não-dominadas presentes no conjunto de referência, sendo dada por:

$$C1_{REF}(A) = \frac{|A \cap REF|}{|REF|} \times 100 \tag{4.3}$$

em que $C1_{REF}(A)$ é a cardinalidade do conjunto A de soluções não-dominadas em relação ao conjunto de referência REF. O valor de $C1_{REF}(A)$ está no intervalo $[0, 100]$, de modo que $C1_{REF}(A) = 100$ significa que todas as soluções do conjunto REF estão em A e $C1_{REF}(A) = 0$ significa que nenhuma solução de REF está em A.

Uma desvantagem da métrica de cardinalidade é que ela não consegue avaliar a distribuição das soluções do conjunto em relação ao conjunto de referência, e a proximidade de um conjunto de soluções em relação ao conjunto de referência.

4.6.2 Métrica Hipervolume

O hipervolume, ou HV, é uma métrica bastante utilizada para comparar resultados de algoritmos multiobjetivos. Ela avalia a distribuição do conjunto de soluções em relação ao espaço de busca e foi proposta por Zitzler e Thiele (1999). Para se obter o hipervolume de um conjunto P, é calculado o volume da região coberta entre os pontos das soluções do conjunto P e um ponto de referência W. Para cada solução i pertencente a P, é construído um hipercubo v_i com referência ao ponto W. Em problemas de minimização, o ponto de referência pode ser encontrado construindo-se um vetor com os piores valores de função-objetivo. Em problemas de maximização, é comum utilizar o ponto de coordenadas $(0, 0)$. O hipervolume é calculado por:

$$HV = \sum_{i \in P} v_i \tag{4.4}$$

em que v_i representa o hipercubo da solução i.

O resultado da métrica é a união de todos os hipercubos encontrados. A área sombreada da Figura 4.4 define o hipervolume de um conjunto de soluções P para um problema de minimização com duas funções objetivo f_1 e f_2, em que o ponto W é utilizado para limitar essa área.

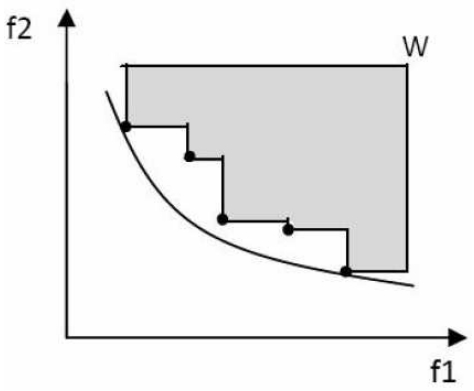

Figura 4.4: Hipervolume gerado pelas soluções da fronteira de P.

Quanto maior o valor do hipervolume, melhor, pois um alto valor de hipervolume indica que houve um elevado espalhamento entre as soluções de P e indica também que houve uma melhor convergência.

4.6.3 Métrica *Epsilon*

A métrica *epsilon* multiplicativo foi proposta por Zitzler et al. (2005). Essa métrica é baseada no conceito de ϵ-dominância.

Diz-se que um vetor de objetivos $z^a = (z_1^a, z_2^a, \ldots, z_n^a) \in Z$ ϵ-domina um outro vetor de objetivos $z^b = (z_1^b, z_2^b, \ldots, z_n^b) \in Z$, em um problema de minimização, se e somente se:

$$\forall i \in \{1, \ldots, n\} : z_i^a \leqslant \epsilon z_i^b \tag{4.5}$$

A Figura 4.5 ilustra o funcionamento da ϵ-dominância para uma solução z^a em relação a três diferentes valores de ϵ. A Figura 4.5(a) considera um valor de ϵ menor que 1. Nesta situação, é possível observar que a solução z^a ϵ-domina as soluções z^b, z^d e z^e. A Figura 4.5(b) considera um valor de ϵ é igual a 1; neste caso, a solução z^a ϵ-domina as soluções z^d e z^e. Por fim, na Figura 4.5(c), em que o valor de ϵ é maior que 1, a solução z^a ϵ-domina apenas a solução z^d.

O valor da métrica *epsilon* multiplicativo (I_ϵ) para dois conjuntos aproximados A e B representa o valor mínimo pelo qual cada ponto pertencente a B deve ser multiplicado de forma que o resultado seja fracamente dominado por A. O valor de ϵ pode ser calculado conforme a Equação (4.6):

$$I_\epsilon(A, B) = \max_{z^b \in B} \left\{ \min_{z^a \in A} \left\{ \max_{1 \leqslant i \leqslant r} \frac{z_i^a}{z_i^b} \right\} \right\} \tag{4.6}$$

Observa-se que o valor de ϵ precisaria ser calculado para todos os pares de resultados, com a finalidade de comparar os algoritmos. Todavia, (Zitzler et al., 2005)

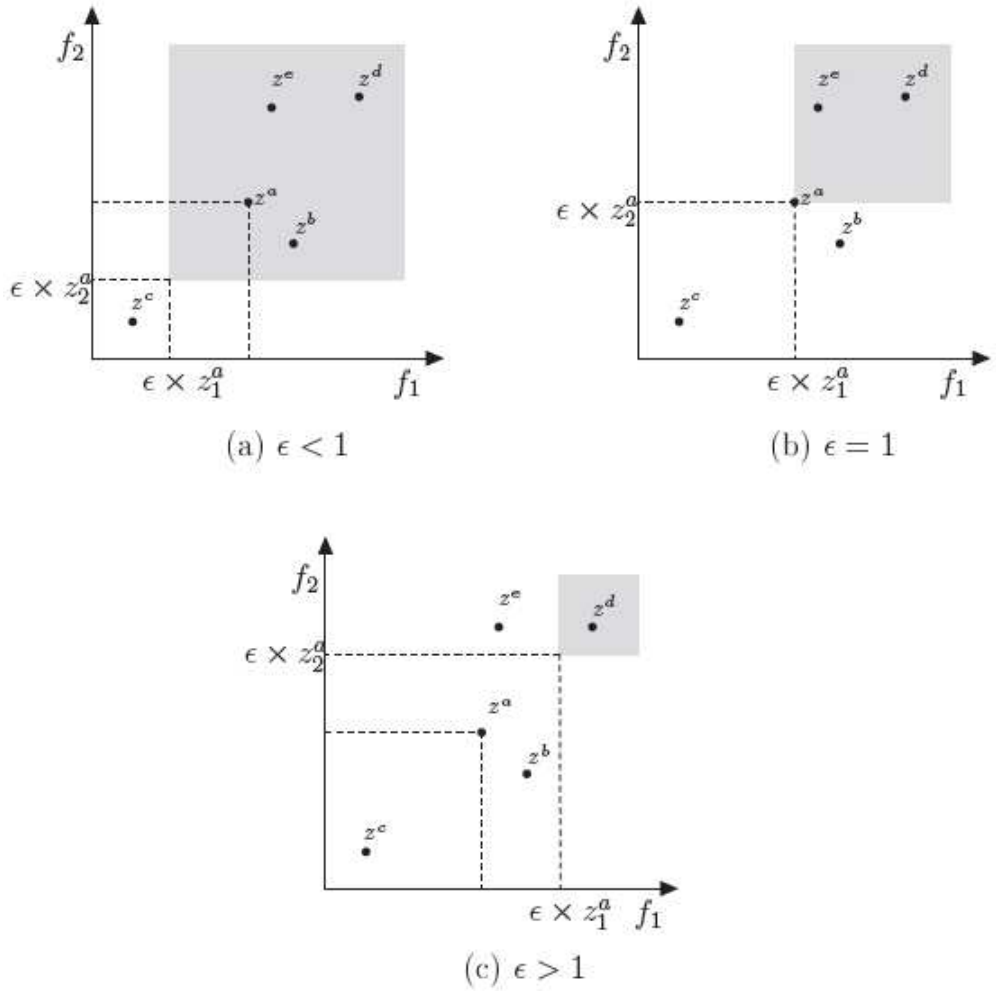

Figura 4.5: Métrica Epsilon.

propuseram uma versão unária, em que A é o conjunto de soluções não-dominadas fornecida por cada algoritmo, e o valor de ϵ é calculado apenas em relação ao conjunto REF das melhores soluções conhecidas. Desta forma, tem-se que $I_\epsilon^1(A) = I_\epsilon(A, R)$.

Como $I_\epsilon(A, REF)$ mede a distância máxima do conjunto A com relação a REF, então um valor próximo a 1 da medida $I_\epsilon^1(A)$ indica uma boa qualidade do conjunto A.

Capítulo 5

Metodologia

Este capítulo apresenta a metodologia proposta para a solução do Problema de Desenvolvimento de Cronogramas (PDC) de projetos de software, utilizando os algoritmos expostos no Capítulo 4.

5.1 Representação de uma Solução

Cada solução s do problema é representada por uma matriz bidimensional, denominada Matriz de Alocação X, e um vetor, denominado Vetor Cronograma Y, que armazena o instante de início de cada tarefa.

Uma dimensão da Matriz de Alocação representa os recursos humanos disponíveis $\{r_1, r_2, ..., r_{|R|}\}$, e a outra as tarefas que devem ser executadas $\{j_1, j_2, ..., j_{|J|}\}$. Na matriz X, cada posição representa o valor da variável $x_{r,j}$, que recebe o valor -1 ou um valor inteiro entre 0 e a dedicação diária de cada recurso, acrescida do tempo máximo de hora extra, se for o caso. Esse valor inteiro é interpretado dividindo-o pela dedicação diária, de forma a se obter porcentagens de 0 a 100%, que representam o esforço dedicado pelo recurso r na execução da tarefa j. Quando o percentual for superior a 100%, indica a realização de hora extra. Quando a variável $x_{r,j}$ recebe o valor -1, representa a incompatibilidade entre o recurso r e a tarefa j, ou seja, o recurso não pode executar a tarefa, pois não possui a habilidade requerida.

Matriz Alocação

	j_1	j_2	j_3	j_4	j_5	J_6
r_1	0	8	-1	-1	1	4
r_2	3	5	-1	-1	2	0
r_3	1	2	-1	-1	-1	-1
r_4	-1	2	6	4	5	-1
r_5	-1	0	8	7	-1	1

Vetor Cronograma

0	4.56	4.56	6.38	8	12.3
j_1	j_2	j_3	j_4	j_5	j_6

Figura 5.1: Representação de uma solução do problema.

O vetor Cronograma possui dimensão J, sendo J o total de tarefas. Os índices do vetor representam as tarefas, e cada posição do vetor é preenchida por um valor

real, que indica o tempo de inicio de execução da tarefa. Na Figura 5.1, tem-se um exemplo de uma solução para o PDC.

5.2 Geração da Solução Inicial

A geração de uma solução inicial pode ser dividida em três estágios. Nos dois primeiros estágios é construída a Matriz de Alocação X (veja na Seção 5.1) e, no terceiro estágio, é construído o vetor Cronograma Y (veja na Seção 5.1). Para o algoritmo GRASP-MOVNS, optou-se por utilizar somente soluções factíveis. Portanto cada estágio de construção das soluções iniciais é feito respeitando-se todas as restrições definidas para o problema. Para o algoritmo NSGA-II é permitido a construção de soluções inicialmente infactíveis.

No primeiro estágio, são alocados de forma aleatória os recursos disponíveis para trabalhar nas tarefas do projeto. Sempre é alocado pelo menos um recurso que atenda cada habilidade requerida para execução da tarefa. O segundo estágio define a dedicação do recurso para executar a tarefa para qual foi alocado. O valor da dedicação é um valor inteiro aleatório, maior que zero e menor que a dedicação máxima diária do recurso r, somado à quantidade de horas extra que o recurso r pode trabalhar.

No terceiro estágio, o vetor Cronograma é gerado por uma heurística construtiva parcialmente gulosa, baseada em duas regras de prioridade.

- Regra 1: a solução é construída a partir da regra de Tempo de Início mais Tarde LS (*Latest Starting Time*), gerada a partir do sequenciamento das tarefas em ordem crescente dos tempos de início mais tarde de cada tarefa;
- Regra 2: a solução é construída a partir da regra do Caminho Crítico do Projeto CMP (*Critical Method Path*), gerada a partir do sequenciamento prioritário das tarefas que fazem parte do caminho crítico do projeto.

Segundo o conhecimento da autora dessa dissertação, não existem regras de prioridade específicas para o PDC. Portanto é selecionada a regra LS (Hartmann e Kolish, 2000; Kolish e Hartmann, 2006), que é uma regra conhecida na literatura para o Problema de Escalonamento de Tarefas com Restrições de Recursos (RCPSP). E a regra CMP (Fitzsimmons e Fitzsimmons, 2005), que é a regra mais utilizada pelas ferramentas de gerenciamento de projeto.

5.3 Vizinhança

Para explorar o espaço de soluções, foram adaptados para o PDC 9 movimentos, descritos a seguir:

- **Movimento Realocar Recurso entre Tarefas Distintas - $M^{RD}(s)$:** este movimento consiste em selecionar duas células x_{ri} e x_{rk} da matriz X e repassar a dedicação de x_{ri} para x_{rk}. Assim, um recurso r deixa de trabalhar na tarefa i e passa a trabalhar tarefa k. Restrições de compatibilidade entre recursos e tarefas são respeitadas, havendo realocação de recursos apenas quando houver

compatibilidade. A Figura 5.2 ilustra a realização desse movimento, com a dedicação do recurso r_2 realocada da tarefa j_2 para a tarefa j_1.

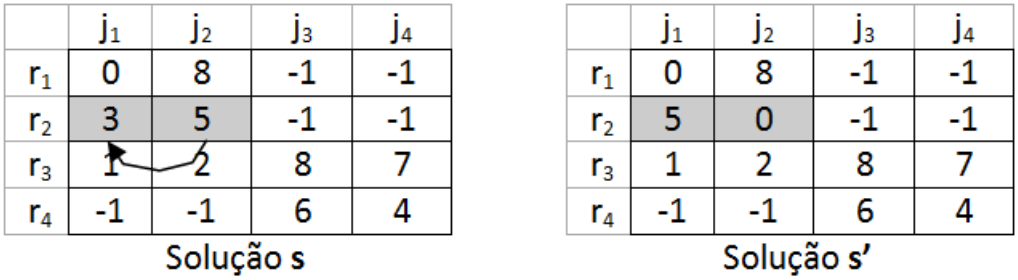

Figura 5.2: Movimento Realocar Recurso entre Tarefas Distintas

- **Movimento Realocar Recurso de uma Tarefa -** $M^{RT}(s)$**:** este movimento consiste em selecionar duas células x_{it} e x_{kt} da matriz X e repassar a dedicação de x_{it} para x_{kt}. Assim, a dedicação de um recurso i é realocada para um recurso k que esteja trabalhando na tarefa t. Restrições de compatibilidade entre recursos são respeitadas, havendo realocação apenas quando houver compatibilidade. A Figura 5.3 ilustra a realização desse movimento, com a dedicação do recurso r_2 realocada para o recurso r_1, ambos trabalhado na tarefa j_2.

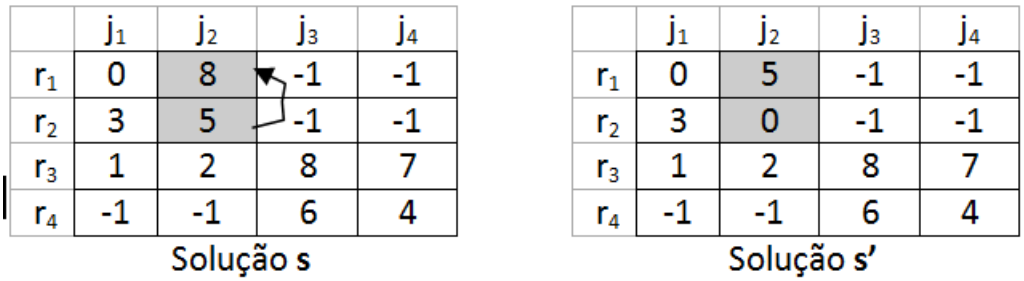

Figura 5.3: Movimento Realocar Recurso de uma Tarefa

- **Movimento Desalocar Recurso de uma Tarefa -** $M^{DT}(s)$**:** consiste em selecionar uma célula x_{rt} da matriz X e zerar seu conteúdo, isto é, retirar a alocação de um recurso r que estava trabalhando na tarefa t. A Figura 5.4 ilustra a realização desse movimento, com a desalocação do recurso r_2, que estava executando a tarefa j_2.
- **Movimento Desalocar Recurso no Projeto -** $M^{DP}(s)$**:** consiste em desalocar toda a dedicação de um recurso r no projeto. O movimento retira todas as alocações do recurso r, que deixa de trabalhar no projeto. O recurso volta a trabalhar no projeto assim que uma nova tarefa for associada a ele. A Figura 5.5 ilustra a realização desse movimento, com a desalocação do recurso r_1 do projeto.
- **Movimento Dedicação de Recursos -** $M^{DR}(s)$**:** este movimento consiste em aumentar ou diminuir a dedicação de um determinado recurso r na execução

	j_1	j_2	j_3	j_4
r_1	0	8	-1	-1
r_2	3	5	-1	-1
r_3	1	2	8	7
r_4	-1	-1	6	4

Solução s

	j_1	j_2	j_3	j_4
r_1	0	8	-1	-1
r_2	3	0	-1	-1
r_3	1	2	8	7
r_4	-1	-1	6	4

Solução s'

Figura 5.4: Movimento Desalocar Recurso de uma Tarefa

	j_1	j_2	j_3	j_4
r_1	0	8	-1	-1
r_2	3	5	-1	-1
r_3	1	2	8	7
r_4	-1	-1	6	4

Solução s

	j_1	j_2	j_3	j_4
r_1	0	0	-1	-1
r_2	3	5	-1	-1
r_3	1	2	8	7
r_4	-1	-1	6	4

Solução s'

Figura 5.5: Movimento Desalocar Recurso no Projeto

de uma tarefa t. Neste movimento, uma célula x_{rt} da matriz X tem seu valor acrescido ou decrescido em uma unidade. A Figura 5.6 ilustra a realização das duas variações desse movimento. A dedicação do recurso r_1 para executar a tarefa j_1 acrescida em uma unidade, e a dedicação do recurso r_3 para executar a tarefa j_2 diminuida em uma unidade.

	j_1	j_2	j_3	j_4
r_1	3	8	-1	-1
r_2	3	5	-1	-1
r_3	1	2	7	8
r_4	-1	-1	6	4

Solução s

	j_1	j_2	j_3	j_4
r_1	4	8	-1	-1
r_2	3	5	-1	-1
r_3	1	1	7	8
r_4	-1	-1	6	4

Solução s'

Figura 5.6: Movimento Dedicação de Recursos

- **Movimento Troca de Recursos entre Tarefas -** $M^{TB}(s)$**:** duas células x_{ri} e x_{rk} da matriz X são selecionadas e seus valores são permutados, isto é, os recursos que trabalham nas tarefas i e k são trocados. Restrições de compatibilidade entre recursos e tarefas são respeitadas, havendo troca de recursos apenas quando houver compatibilidade. A Figura 5.7 ilustra a realização desse movimento, com a dedicação do recurso r_3 para executar a tarefa j_3 trocada com sua dedicação para executar a tarefa j_4.

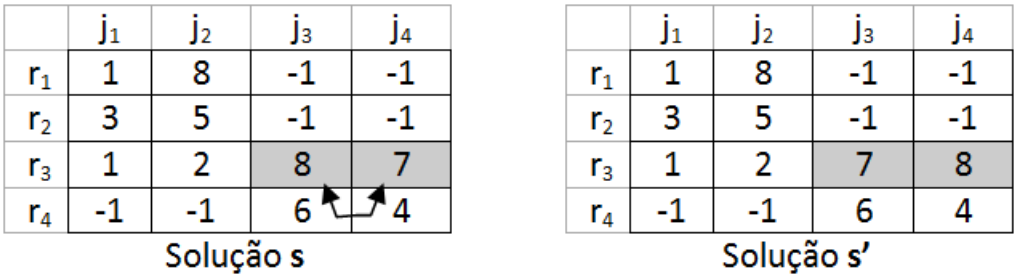

Figura 5.7: Movimento Troca de Recursos entre Tarefas

- **Movimento Troca de Recursos de uma Tarefa -** $M^{TO}(s)$**:** duas células x_{it} e x_{kt} da matriz X são selecionadas e seus valores são permutados, isto é, os recursos i e k que trabalham na tarefa t são trocados. Restrições de compatibilidade entre recursos são respeitadas, havendo realocação apenas quando houver compatibilidade. A Figura 5.8 ilustra a realização desse movimento, com a dedicação do recurso r_1 trocada com a dedicação do r_2 para executar a tarefa j_1.

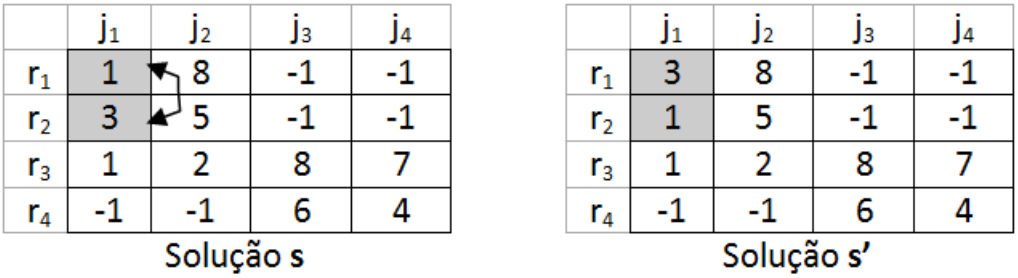

Figura 5.8: Movimento Troca de Recursos de uma Tarefa

- **Movimento Insere Tarefa -** $M^{IT}(s)$**:** este movimento consiste em inserir o tempo de inicio de uma tarefa que está em uma posição i em outra posição j do vetor Y. Esse movimento é realizado somente entre tarefas sem relações de precedência. Caso o movimento quebre alguma restrição de recurso, a nova solução gerada é descartada. A Figura 5.9 ilustra a realização desse movimento, com o tempo de início da tarefa j_2 inserido na tarefa j_4.

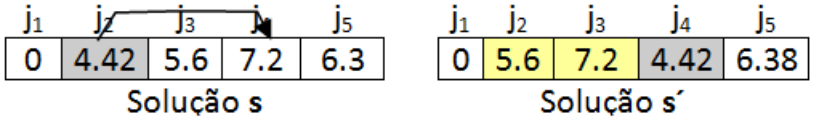

Figura 5.9: Movimento Insere Tarefa

- **Movimento Troca Tarefa -** $M^{TT}(s)$**:** consiste em trocar duas células distintas y_i e y_k do vetor Y, ou seja, trocar o tempo de início de execução das tarefas

i e k. Os movimentos de trocas serão feitos sempre respeitando a ordem de precedência para executar as tarefas. Caso esse movimento quebre alguma restrição de recurso, a nova solução gerada é descartada. A Figura 5.10 ilustra a realização desse movimento, com o tempo de inicio da tarefa j_2 trocado com o tempo de inicio da tarefa j_3.

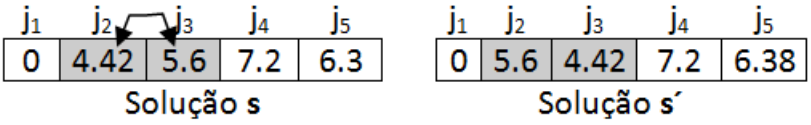

Figura 5.10: Movimento Troca Tarefa

5.4 Operadores

Nas subseções seguintes são descritos os três operadores utilizados: seleção, cruzamento e mutação. Todos foram implementados como proposto por Colares (2010).

5.4.1 Seleção

Em um algoritmo genético é necessário implementar um mecanismo que realize a seleção dos indivíduos da população atual que sofrerão cruzamento ou mutação. O mecanismo de seleção utilizado foi o Torneio Binário. Nesse operador, dois pares de indivíduos são selecionados aleatoriamente, mas apenas o melhor indivíduo de cada par é escolhido para fazer a reprodução. Na implementação do NSGA-II escolhe-se o indivíduo que domina o outro; caso eles sejam não-dominados, escolhe-se o que tiver maior valor para a *crowding distance*.

5.4.2 Mutação

A mutação é uma mudança na estrutura genética do indivíduo. Nos algoritmos genéticos, os operadores de mutação são importantes instrumentos que permitem introduzir características novas ao indivíduo. Com isso, é possível manter a diversidade genética da população, diminuindo as chances do algoritmo ficar preso a um ótimo local. Foi utilizado o operador de mutação *Bit Flip*, que altera aleatoriamente o valor da variável $x_{r,j}$ com uma probabilidade p_m. O valor da parâmetro p_m é definido na Seção 5.6.

5.4.3 Cruzamento

O operador de cruzamento atua em dois indivíduos de cada vez, gerando descendentes através da combinação dos cromossomos dos dois indivíduos. Esses operadores possuem diferentes variações, dependendo do problema tratado. Os operadores tradicionais são unidimensionais, ou seja, são operadores aplicáveis em estrutura de dados como vetores.

No PDC, o indivíduo é representado por uma matriz. Portanto não é adequado o uso de um operador tradicional. Por esse motivo, foi utilizado um operador multidimensional de cruzamento, baseado no operador de cruzamento tradicional *Single Point*. O operador implementado seleciona um ponto aleatório (r, j) e, a partir dele, troca as informações genéticas dos indivíduos pais, formando-se dois novos indivíduos filhos. A Figura 5.11 ilustra o cruzamento multidimensional, de modo que dois indivíduos são combinados a partir do ponto $(3, 2)$. O cruzamento é realizado com uma probabilidade p_c. O valor da parâmetro p_c é definido na Seção 5.6.

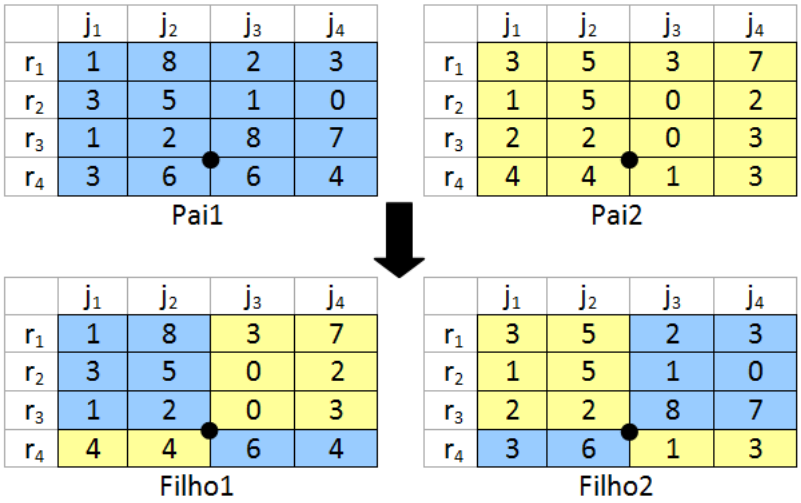

Figura 5.11: Operador de Cruzamento Bidimensional

5.5 Algoritmos Implementados

5.5.1 GRASP-MOVNS

Foi implementado o algoritmo multiobjetivo denominado GRASP-MOVNS, que consiste na combinação dos procedimentos Heurísticos *Greedy Randomized Adaptative Search Procedure* (GRASP), que está detalhado no Algoritmo 4 da Seção 4.2.3, e o Algoritmo *Multiobjective Variable Neighborhood Search* (MOVNS), detalhado no Algoritmo 9. O algoritmo GRASP-MOVNS foi implementado utilizando a mesma estratégia proposta por Coelho et al. (2012).

O Algoritmo 9 apresenta o pseudocódigo do algoritmo GRASP-MOVNS. Na linha 2, é gerado o conjunto inicial de soluções não dominadas através do procedimento parcialmente guloso GRASP (Algoritmo 4). São geradas duas soluções iniciais: s_1, que é gerada pela regra de prioridade LS; e s_2, que é gerada pela regra de prioridade CMP. As duas soluções são construídas como descrito na Seção 5.2. Cada solução do problema é representada conforme descrito na Seção 5.1.

A versão do GRASP implementada utiliza, como método de busca local, a heurística *Variable Neighborhood Descent* (VND), conforme mostrado no Algoritmo 4.2.2, que envolve a substituição da solução atual pelo resultado da busca local, quando há uma melhora. Porém, a estrutura de vizinhança é trocada de forma determinística,

Algoritmo 9: GRASP-MOVNS

Entrada: *Vizinhança $N_K(s)$; graspMax; levelMax*
Saída: *Aproximação de um conjunto eficiênte D*

```
1  início
2      D ← GRASP(graspMax)
3      level ← 1
4      shaking ← 1
5      enquanto (Critério de parada não satisfeito) faça
6          Seleciona uma solução não visitada s ∈ D
7          Marque s como visitada
8          s ← s'
9          for i → shaking do
10             Selecione aleatóriamente uma vizinhança N_k(.)
11             s' ← Pertubao(s',k)
12         end
13         k_ult ← k
14         incrementa ← verdadeiro
15         for s'' ∈ N_{k_ult}(s') do
16             addSolution(D, s", f(s"), added)
17             se added = verdadeiro então
18                 incrementa ← falso
19             fim
20         end
21         se incrementa = verdadeiro então
22             level ← level + 1
23         senão
24             level ← 1
25             shaking ← 1
26         fim
27         se level ≥ levelMax então
28             level ← 1
29             shaking ← shaking + 1
30         fim
31         se todo s ∈ D estão marcadas como visitadas então
32             Marque todos s ∈ D como não visitado
33         fim
34     fim
35     Retorne D
36 fim
```

cada vez que se encontra um mínimo local. A solução resultante é um mínimo local em relação a todas as nove estruturas de vizinhanças: M^{IT}, M^{TT}, M^{DR}, M^{RD}, M^{RT}, M^{DP}, M^{DT}, M^{TB}, M^{TO}.

Nas linhas 6 e 7 do Algoritmo 9 é feita a seleção de um indivíduo do conjunto de soluções não-dominadas D, marcando-o como "visitado". Quando todos os indivíduos

Algoritmo 10: addSolution

Entrada: conjunto D, s',$z(s')$
Saída: $conjuntoD, added$

```
1  início
2      added ← true
3      for s ∈ D do
4          se z(s) ≤ z(s') então
5              added ← false
6              break
7          fim
8          se z(s') < z(s) então
9              D = D s
10         fim
11     end
12     se added = true então
13         D = D ⋃ s'
14     fim
15     Retorne D, added
16 fim
```

estiverem marcados como visitados, a linha 32 retira estes marcadores. As variáveis *level* e *shaking* (linhas 3 e 4 do Algoritmo 9) regulam a perturbação utilizada no algoritmo. Esta versão do algoritmo MOVNS, proposta por (Coelho et al., 2012), possui um mecanismo que regula o nível de perturbação do algoritmo, ou seja, a variável *shaking* é incrementada quando o algoritmo passa um determinado tempo sem obter boas soluções. Da linha 9 à linha 12 do Algoritmo 9 ocorre o laço de perturbação do algoritmo. A heurística *AdicionarSolucao*, mostrada no Algoritmo 10 e acionado na linha 16, adiciona, ao conjunto solução D, as soluções geradas pelo GRASP-MOVNS. No mecanismo utilizado, quanto maior o valor da variável *shaking*, maior será a intensidade da perturbação na solução. Para cada unidade dessa variável, aplica-se um movimento aleatório dentre as nove vizinhanças: M^{IT}, M^{TT}, M^{DR}, M^{RD}, M^{RT}, M^{DP}, M^{DT}, M^{TB}, M^{TO}. A variável *level* regula quando a variável *shaking* será incrementada. As linhas 24 e 25 retornam os valores das variáveis *level* e *shaking* para uma unidade, quando, pelo menos, uma solução é adicionada ao conjunto eficiente D.

5.5.2 NSGA-II

O algoritmo NSGA-II foi implementado como proposto por Colares (2010) para resolver o PDC. Em sua pesquisa Colares (2010), implementou a versão clássica do algoritmo NSGA-II, que está detalhada no Algoritmo 8.

As seguintes adaptações foram feitas no Algoritmo 8:

- Na linha 1, é determinada a solução inicial com base no método descrito na Seção 5.2 combinado com um procedimento que gera soluções aleatoriamente, uma vez que é necessário que a população inicial do NSGA-II tenha tamanho

P.

- Relaxamento das restrições do problema, permitindo construir soluções inicialmente infactíveis.

Optou-se por essa estratégia para que o operador de cruzamento descrito na Seção 5.4.3 pudesse ser usado. O tratamento das soluções infactíveis é feito penalizando as soluções que apresentam quebra de restrições, de forma que soluções válidas sejam sempre priorizadas. Esse tratamento das soluções é feito durante a avaliação dos indivíduos, como proposto por Colares (2010).

Na linha 5, o critério de parada adotado está descrito na Seção 5.6. A geração de novos indivíduos da linha 23 é realizada por mutação e cruzamento, conforme descrito nas Seções 5.4.2 e 5.4.3, respectivamente. A mutação acontece com uma probabilidade p_m e o cruzamento acontece com uma probabilidade p_c. Esses parâmetros foram definidos conforme descrito na Seção 5.6. Cada indivíduo é representado apenas pela Matriz de Alocação X, como definida na Seção 5.1.

5.6 Parâmetros

Nesta seção é descrito o procedimento utilizado para se determinar os valores dos parâmetros dos algoritmos.

Para o algoritmo GRASP-MOVNS, foram experimentados valores para os parâmetros a fim de escolher o mais adequado. Esses testes consistiram em executar 30 vezes o GRASP-MOVNS em 2 instâncias, escolhidas aleatoriamente do conjunto de instâncias utilizadas, e escolher o valor do parâmetro que fornecesse os melhores resultados. Para o algoritmo NSGA-II, foram utilizados os mesmos parâmetros propostos em Colares (2010).

O algoritmo GRASP-MOVNS possui dois parâmetros:

- $graspMax$: número máximo de iterações da fase de construção GRASP;
- $levelMax$: controla a intensidade da perturbação na solução.

Foram definidos os seguintes valores para essas variáveis: $graspMax = 200, levelMax = 10$.

O algoritmo NSGA-II foi executado com os mesmo parâmetros definidos em Colares (2010):

- população inicial $P = 200$ indivíduos;
- probabilidade de mutação $p_m = 1/tamanho$;
- probabilidade de cruzamento $p_c = 90\%$.

Os dois algoritmos implementados têm em comum o parâmetro de critério de parada. Para esse parâmetro, adotou-se, como tempo limite, o valor de 180 segundos, como proposto por Colares (2010).

Capítulo 6

Resultados Computacionais

Os algoritmos foram implementados em linguagem java e os experimentos realizados em um notebook Dell Inspirion 14 Core i3-3110M, 3 MB Cache, 2.4 GHz, 4GB de RAM, com sistema operacional windows 7 64 bits.

O critério de parada de cada algoritmo é o tempo de CPU. Para executar os testes, adotou-se o tempo de 180 segundos. Para cada instância, foram realizadas 30 execuções, com diferentes sementes geradas aleatoriamente.

6.1 Instâncias

Para testar os algoritmos, foram geradas instâncias a partir de dados de projetos reais de desenvolvimento de software. As instâncias $inst1$ e $inst2$ são as mesmas instâncias utilizadas em Colares (2010). As demais instâncias utilizadas foram coletadas para esse estudo em uma empresa de desenvolvimento de sistemas de Belo Horizonte, que está no mercado há mais de 8 anos.

Ao todo, foram utilizadas 4 instâncias. A instância $inst1$ possui 5 casos de uso, totalizando 27 tarefas, 18 recursos e 12 habilidades. A instância $inst2$ possui 10 casos de uso, totalizando 72 tarefas, 21 recursos e 7 habilidades envolvidas. A instância $inst3$ possui 16 casos de uso, totalizando 100 tarefas, 16 recursos e 4 habilidades. A instância $inst3$ possui 20 casos de uso, com 120 tarefas, 11 recursos e 4 habilidades. A Tabela 6.1 apresenta um resumo das características das instâncias, sendo T o total de tarefas, R o total de recursos e H o total de habilidades.

Tabela 6.1: Instâncias de teste

Instância	T	R	H
$inst1$	27	18	12
$inst2$	72	21	7
$inst3$	100	16	4
$inst4$	120	11	4

Todos os recursos possuem uma carga horária de 8 horas diárias, com salários de acordo com a função desempenhada e experiência. O algoritmo foi configurado durante os testes, para não permitir que sejam feitas horas extras. Cada recurso possui suas próprias habilidades e cada tarefa possui uma lista de habilidades reque-

ridas para sua execução. Como os dados utilizados não especificavam proficiência e experiência dos recursos, foi considerado sempre o valor “normal".

6.2 Experimentos Computacionais

As tabelas a seguir apresentam os resultados obtidos pelos algoritmos desenvolvidos, considerando-se a aplicação de diferentes métricas. Para cada métrica, são apresentados o resultado médio e o melhor resultado de cada instância para cada algoritmo. O valor do resultado médio de cada instância é obtido pela média aritmética dentre as 30 execuções. O valor para o melhor resultado é obtido considerando apenas o melhor resultado dentre as 30 execuções para cada instância, em relação a cada métrica. Listam-se, ainda, os valores médios para as 4 instâncias da métrica em análise.

Na Tabela 6.2 estão os resultados obtidos pelos algoritmos implementados neste trabalho (GRASP-MOVNS e NSGA-II) com relação ao número de soluções não dominadas geradas por cada um deles. A primeira coluna possui o nome da instância, as duas colunas seguintes detalham o tamanho da instância, sendo T o total de tarefas e R o total de recursos. Na quarta coluna é apresentado o total de soluções distintas que compõem o conjunto de referência REF. O conjunto REF é gerado por 30 execuções de 2 horas de cada algoritmo, mais o conjunto de soluções gerado durante os demais testes. Nas demais colunas, mostra-se, para cada algoritmo e para cada instância, o número total de soluções não-dominadas geradas pelo respectivo algoritmo em 30 execuções de 180 segundos. Na última linha totalizam-se os resultados de cada algoritmo.

Tabela 6.2: Número de soluções não-dominadas

Instância	T	R	REF	Algoritmo	
				GRASP-MOVNS	**NSGA-II**
*inst*1	27	18	91	**88**	0
*inst*2	72	21	574	**475**	0
*inst*3	100	16	689	**558**	189
*inst*4	120	11	537	**380**	172
Total			1891	**1501**	361

Considerando que, ao todo, 1501 soluções do conjunto de referência foram geradas pelo algoritmo GRASP-MOVNS, de um total de 1891, pode-se afirmar que esse algoritmo conseguiu encontrar o maior número de soluções pertencentes ao conjunto de referência. O algoritmo NSGA-II teve um desempenho muito inferior, encontrando apenas 361 soluções do conjunto de referência. O algoritmo NSGA-II não conseguiu encontrar nenhuma solução pertencente ao conjunto de referência nas instâncias com menos de 100 tarefas e maior disponibilidade de recursos. Para as instâncias que apresentam uma maior quantidade de tarefas e menor disponibilidade de recursos *inst*3 e *inst*4, o algoritmo NSGA-II obteve seu melhor desempenho, conseguindo gerar soluções pertencentes ao conjunto de referência REF. O pior desempenho do algoritmo GRASP-MOVNS foi na instância *inst*4, mas, mesmo assim, ainda foi superior ao desempenho do algoritmo NSGA-II para a mesma instância.

Observa-se que cada algoritmo foi executado 30 vezes para uma determinada instância do problema e o conjunto de soluções não-dominadas obtidas nessas execuções é considerado no cálculo da medida de desempenho do algoritmo.

A Tabela 6.3 apresenta os resultados obtidos com relação à métrica de cardinalidade $C1_{REF}(A)$. A primeira coluna possui o nome da instância, as duas colunas seguintes detalham o tamanho da instância, sendo T o total de tarefas e R o total de recursos. Nas demais colunas, mostra-se, para cada algoritmo e para cada grupo de instâncias, o resultado para a métrica de cardinalidade $C1_{REF}(A)$, calculada de acordo com a expressão (4.3). Abaixo do conjunto de resultados estão as médias para cada algoritmo.

Tabela 6.3: Resultados da Métrica Cardinalidade $C1_{REF}(A)$

Instância	T	R	Algoritmo			
			GRASP-MOVNS		NSGAII	
			Média	**Melhor**	**Média**	**Melhor**
*inst*1	27	18	**29.10**	**71.00**	0	0
*inst*2	72	21	**11.06**	**57.00**	0	0
*inst*3	100	16	**21.40**	**69.00**	3.97	38.00
*inst*4	120	11	**8.83**	**85.00**	2.17	27.00
Média			**17.60**	**70.50**	1.54	16.25

Como pode ser visto na Tabela 6.3, o algoritmo GRASP-MOVNS consegue gerar uma maior quantidade de soluções não-dominadas, quando comparado com o algoritmo NSGA-II. O número de soluções do conjunto referência geradas pelo algoritmo GRASP-MOVNS é, em média, 10 vezes maior que o gerado pelo algoritmo NSGA-II.

O algoritmo NSGA-II obteve seu pior desempenho nas instâncias *inst*1 e *inst*2, não conseguindo gerar nenhuma solução pertencente ao conjunto de soluções de referência. Nas instâncias maiores, *inst*3 e *inst*4, o algoritmo NSGA-II apresentou seus melhores resultados. O algoritmo GRASP-MOVNS apresentou seu pior resultado na instância *inst*4. Em relação aos melhores resultados, o algoritmo GRASP-MOVNS também conseguiu o melhor desempenho, com resultados bem superiores ao NSGA-II.

A Tabela 6.4 apresenta os resultados obtidos com relação à métrica de hipervolume HV. A primeira coluna possui o nome da instância, as duas colunas seguintes detalham o tamanho da instância, sendo T o total de tarefas e R o total de recursos. Nas demais colunas, mostra-se, para cada algoritmo e para cada grupo de instâncias, o resultado para a métrica de hipervolume HV, calculada de acordo com a expressão (4.4). Abaixo do conjunto de resultados estão as médias para cada algoritmo.

Na Tabela 6.4 podemos observar que os maiores valores de hipervolume foram obtidos pelo algoritmo GRASP-MOVNS, mostrando que o GRASP-MOVNS gera soluções com um grau de espalhamento mais elevado. O algoritmo NSGA-II obteve o seu melhor desempenho na instância *inst*4, conseguindo produzir na média resultados mais próximos aos gerados pelo algoritmo GRASP-MOVNS.

A Tabela 6.5 apresenta os resultados obtidos com relação à métrica de *Epsilon* I_ϵ^1. A primeira coluna possui o nome da instância, as duas colunas seguintes detalham o tamanho da instância, sendo T o total de tarefas e R o total de recursos. Nas demais colunas, mostra-se, para cada algoritmo e para cada grupo de instâncias, o resultado

Tabela 6.4: Resultados da Métrica Hipervolume HV

Instância	T	R	Algoritmo			
			GRASP-MOVNS		NSGAII	
			Média	**Melhor**	**Média**	**Melhor**
*inst*1	27	18	**0.64**	**0.83**	0.12	0.32
*inst*2	72	21	**0.65**	**0.92**	0.26	0.73
*inst*3	100	16	**0.80**	**0.85**	0.54	0.79
*inst*4	120	11	**0.77**	**0.89**	0.71	0.79
Média			**0.72**	**0.87**	0.41	0.66

para a métrica I_ϵ^1, calculada de acordo com a expressão (4.6). Abaixo do conjunto de resultados estão as médias para cada algoritmo.

Tabela 6.5: |Resultados da Métrica Epsilon I_ϵ^1

Instância	T	R	Algoritmo			
			GRASP-MOVNS		NSGAII	
			Média	**Melhor**	**Média**	**Melhor**
*inst*1	27	18	**1.12**	**1.03**	2.11	1.91
*inst*2	72	21	**1.12**	**1.05**	2.03	1.89
*inst*3	100	16	**1.09**	**1.01**	1.69	1.22
*inst*4	120	11	**1.10**	**1.01**	1.58	1.16
Média			**1.11**	**1.02**	1.85	1.54

Na Tabela 6.5 verifica-se que o algoritmo GRASP-MOVNS consegue produzir os menores valores para a métrica epsilon, indicando que suas melhores soluções estão mais próximas dos conjunto de referência REF. O algoritmo NSGA-II obteve melhor desempenho nas instâncias *inst*3 e *inst*4, conseguindo gerar um melhor resultado, próximo ao gerado pelo algoritmo GRASP-MOVNS para a instância *inst*4.

6.3 Gráficos *BoxPlot*

Nesta seção são apresentados os gráficos de caixa (*BoxPlot*) relativos à variabilidade das métricas dos algoritmos, quando aplicados ao conjunto de instâncias descritas na Seção 6.1.

O gráfico *BoxPlot* apresenta, em uma caixa, o primeiro quartil, o terceiro quartil e a mediana. A haste inferior se estende a partir do limite inferior até o menor valor, e a haste superior se estende do quartil superior até o maior valor. Os pontos fora das hastes inferior e superior são considerados valores discrepantes (*outliers*).

As Figuras 6.1, 6.2, 6.3 e 6.4 apresentam a comparação dos algoritmos GRASP-MOVNS e NSGA-II em relação a métrica de cardinalidade. Nota-se que o algoritmo GRASP-MOVNS apresenta um desempenho, de maneira geral, superior ao algoritmo NSGA-II, como indicado pelo valor das medianas e pela localização das caixas em todas as instâncias.

Para as instâncias *inst*1 e *inst*3 o algoritmo GRASP-MOVNS apresenta uma dispersão elevada, com notável diferença entre os valores máximo e mínimo obtidos.

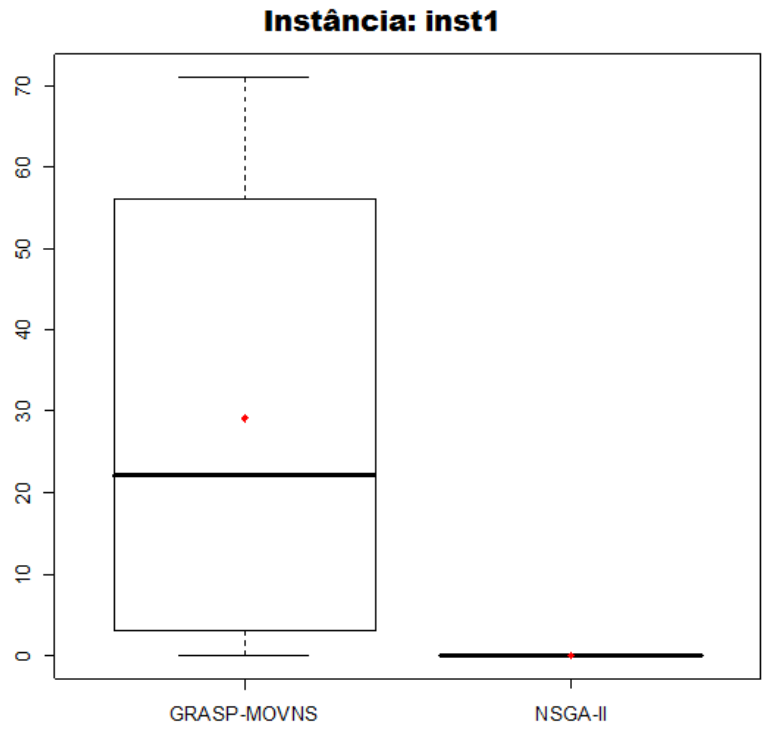

Figura 6.1: Gráficos *BoxPlot* para Métrica Cardinalidade da instância *inst*1.

O algoritmo NSGA-II apresenta uma menor dispersão dos dados, mas nesse caso, isso apenas evidencia, que o NSGA-II encontra resultados parecidos, mas que geralmente são ruins, já que em todas as comparações o NSGA-II obteve pior resultado. O GRASP-MOVNS gerou mais *outliers* que o NSGA-II, ou seja, apresentou mais valores discrepantes dos demais da série.

Como pode ser observado nas Figuras 6.1 e 6.2, para o algoritmo NSGA-II não foi gerado o gráfico *boxplot*, pois, para as instâncias *inst*1 e *inst*2, o algoritmo NSGA-II não gerou nenhuma solução pertencente ao conjunto de referência *REF*.

As Figuras 6.5, 6.6, 6.8 e 6.8 apresentam a comparação dos algoritmos GRASP-MOVNS e NSGA-II em relação à métrica de hipervolume. Pode-se observar que o algoritmo GRASP-MOVNS apresenta uma baixa dispersão dos resultados para as instâncias *inst*1, *inst*3 e *inst*4.

Para as instâncias *inst*3 (resultado mostrado na Figura 6.7) e *inst*4 (resultado mostrado na Figura 6.8), o GRASP-MOVNS apresentou um comportamento simétrico em torno da mediana, apesar de um pequeno desvio em um dos extremos para cada instância. De modo geral, o GRASP-MOVNS mostrou-se mais eficiente na exploração das regiões visitadas.

Como pode ser visto na Figura 6.8, o Algoritmo NSGA-II obteve seu melhor desempenho na instância *inst*4, em que apresentou resultados mais próximos dos gerados pelo algoritmo GRASP-MOVNS.

Finalmente, as Figuras 6.9, 6.10, 6.11 e 6.12 apresentam a comparação entre os algoritmos GRASP-MOVNS e NSGA-II em relação à métrica *Epsilon*. O algoritmo GRASP-MOVNS novamente apresentou melhor desempenho que o algoritmo

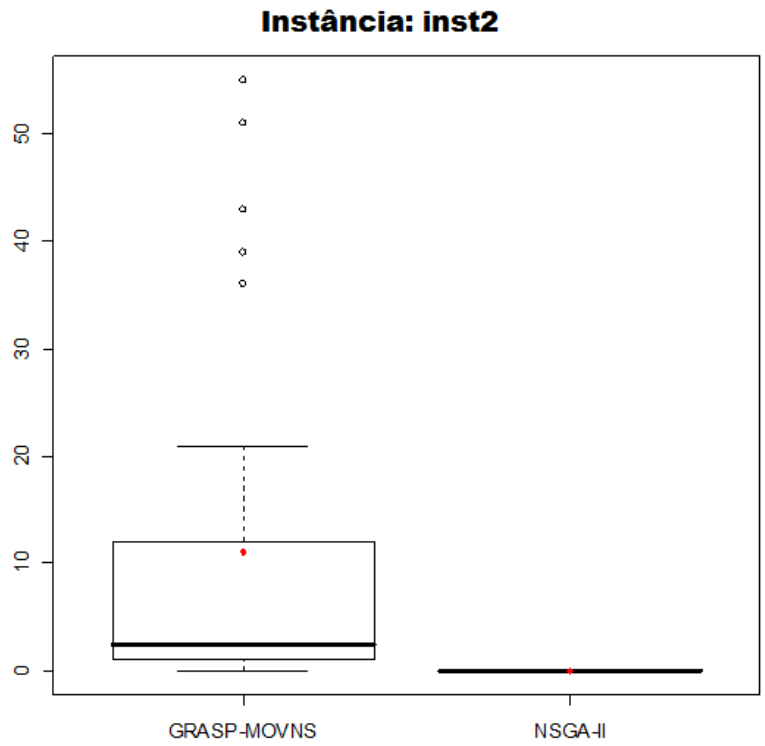

Figura 6.2: Gráficos *BoxPlot* para Métrica Cardinalidade da instância *inst*2.

NSGA-II, conseguindo gerar soluções não-dominadas mais próximas do conjunto de referência REF. Para todas as instâncias testadas, o algoritmo GRASP-MOVNS apresentou uma baixa dispersão dos resultados gerados.

O algoritmo NSGA-II conseguiu obter uma baixa dispersão dos resultados para as instâncias *inst*1 (Figura 6.9) e *inst*2 (Figura 6.10), mas as soluções geradas estão muito distantes do conjunto de referência REF.

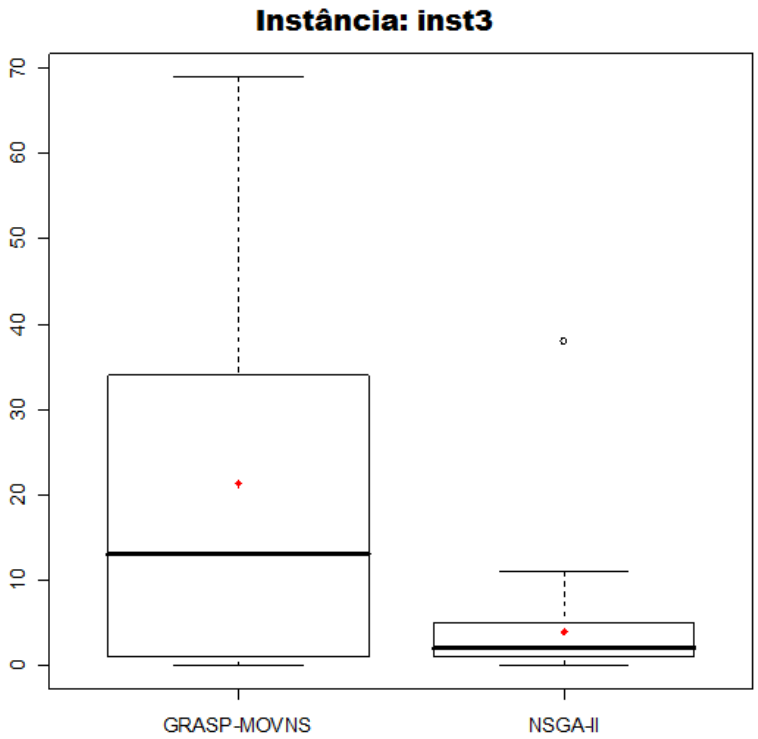

Figura 6.3: Gráficos *BoxPlot* para Métrica Cardinalidade da instância *inst*3.

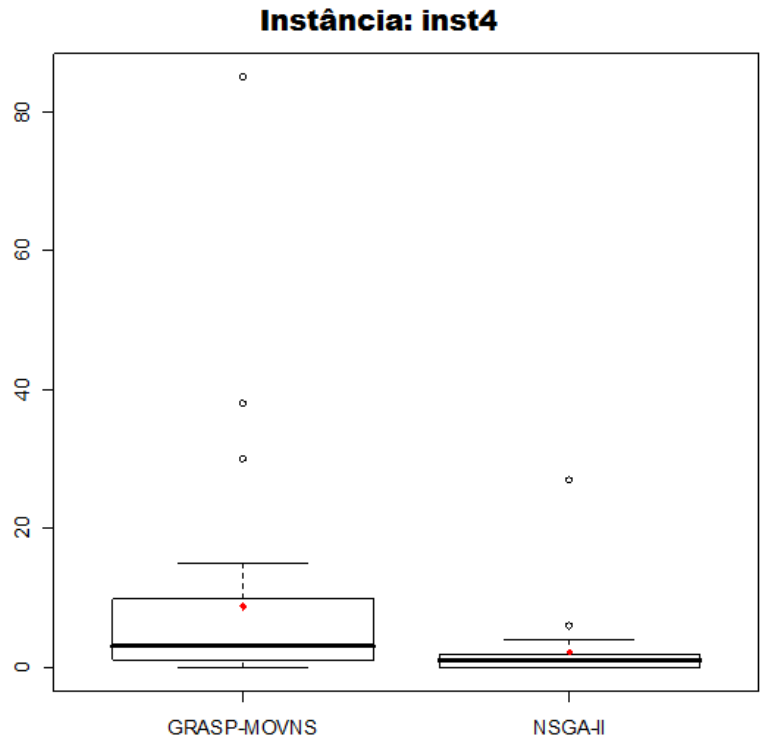

Figura 6.4: Gráficos *BoxPlot* para Métrica Cardinalidade da instância *inst*4.

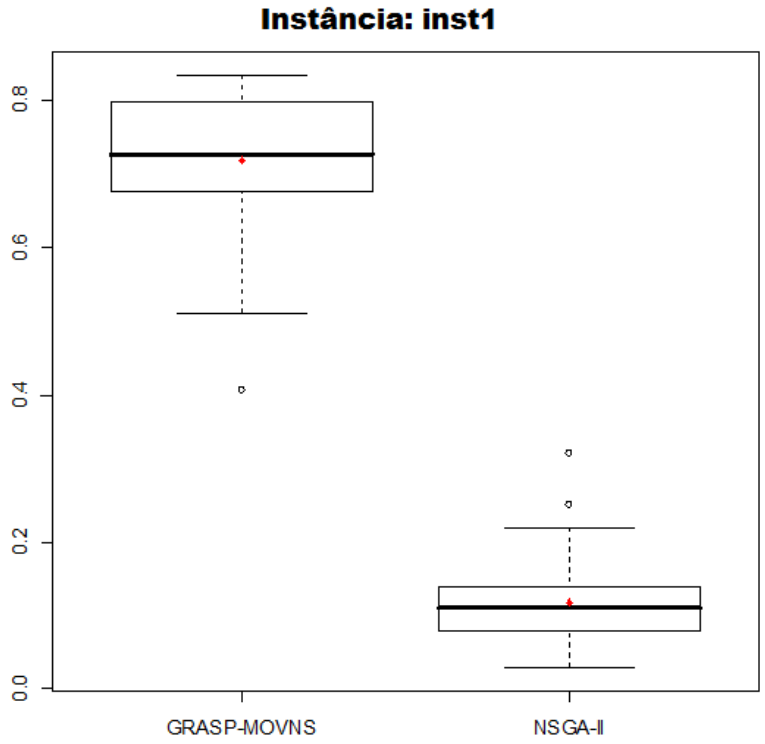

Figura 6.5: Gráficos *BoxPlot* para Métrica Hipervolume da instância *inst*1.

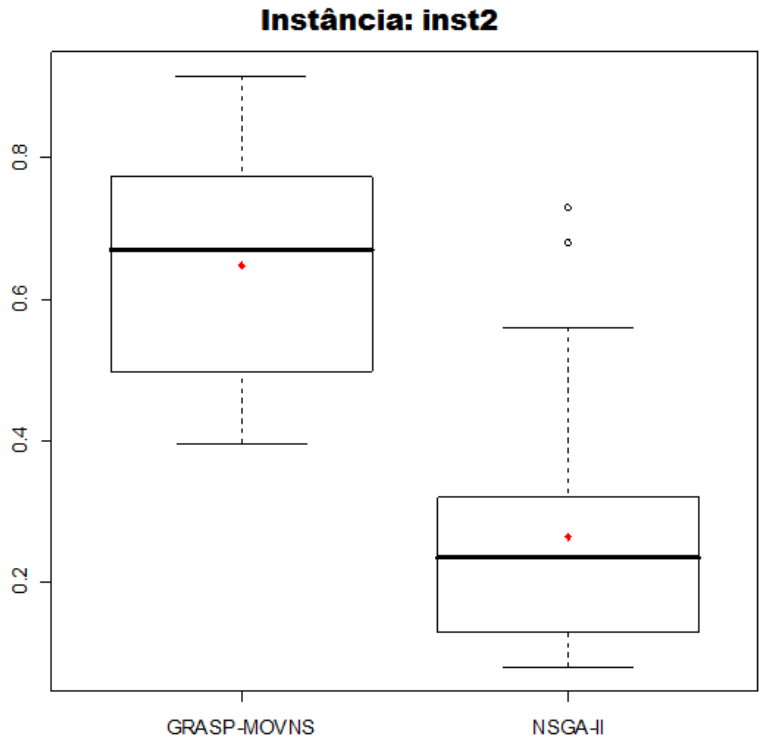

Figura 6.6: Gráficos *BoxPlot* para Métrica Hipervolume da instância *inst*2.

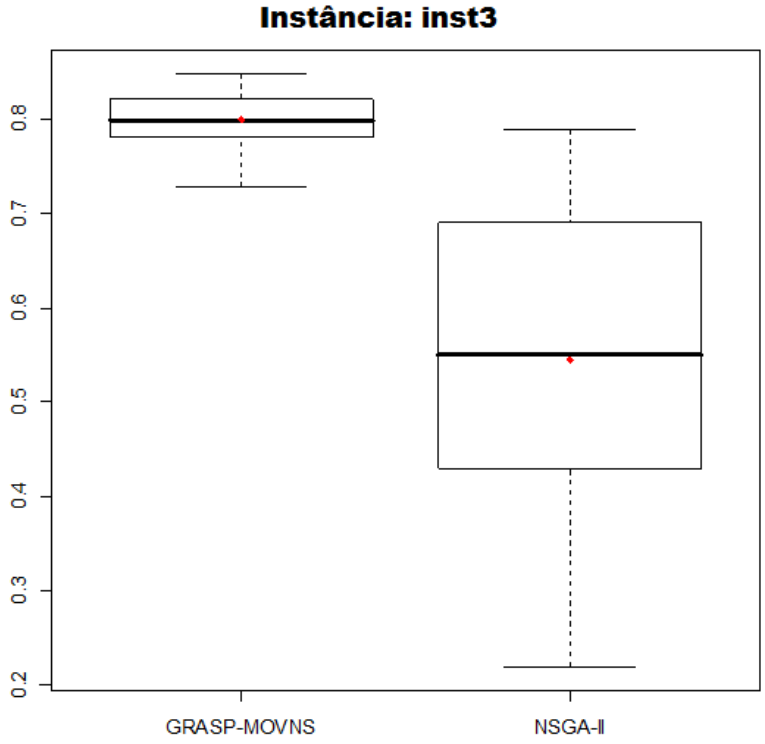

Figura 6.7: Gráficos *BoxPlot* para Métrica Hipervolume da instância *inst*3.

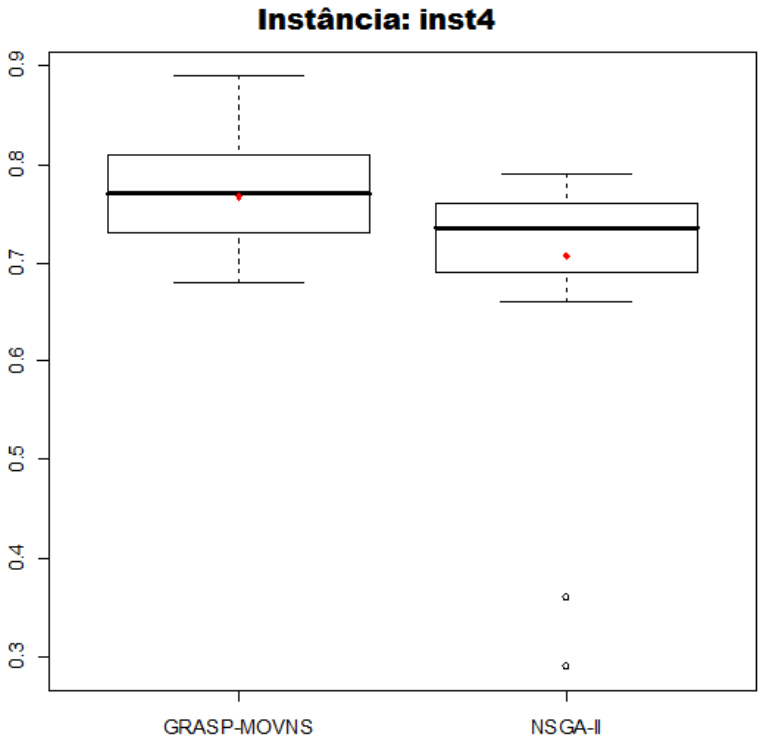

Figura 6.8: Gráficos *BoxPlot* para Métrica Hipervolume da instância *inst*4.

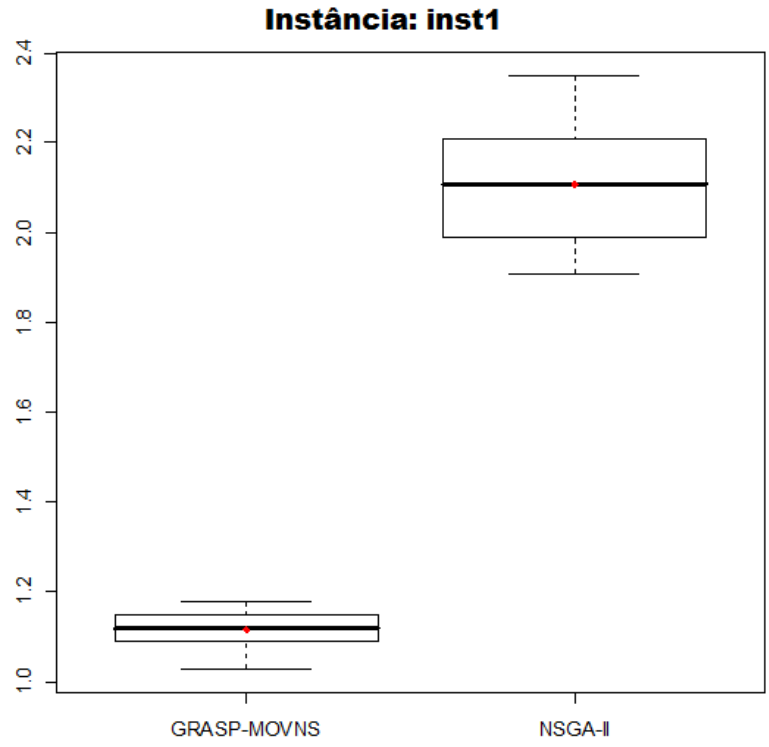

Figura 6.9: Gráficos *BoxPlot* para Métrica Epsilon da instância *inst*1.

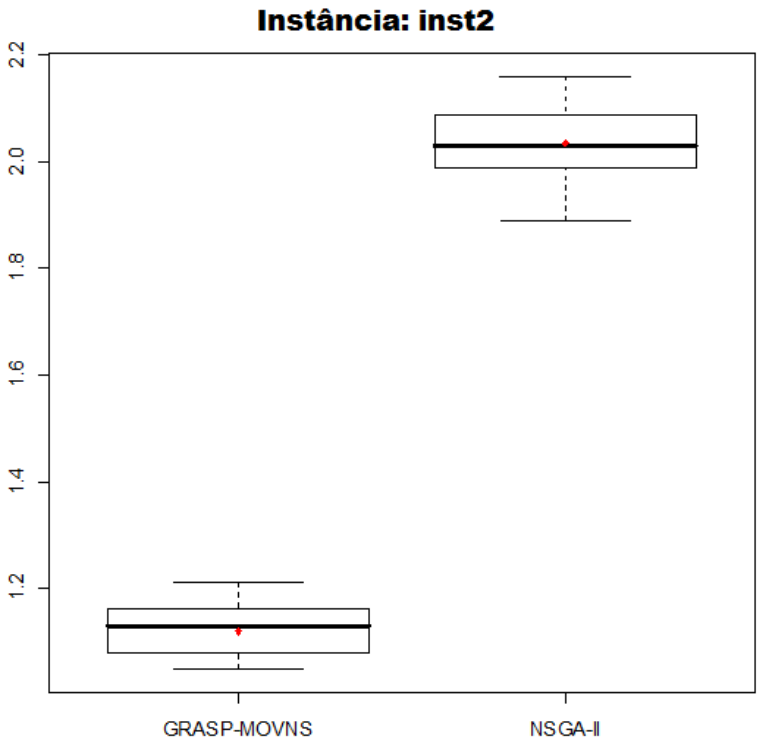

Figura 6.10: Gráficos *BoxPlot* para Métrica Epsilon da instância *inst*2.

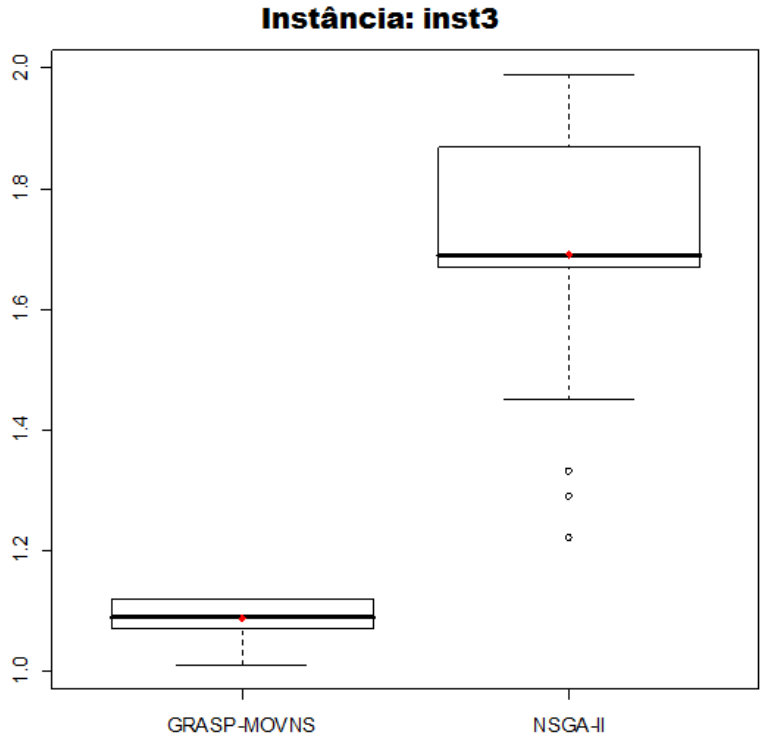

Figura 6.11: Gráficos *BoxPlot* para Métrica Epsilon da instância *inst*3.

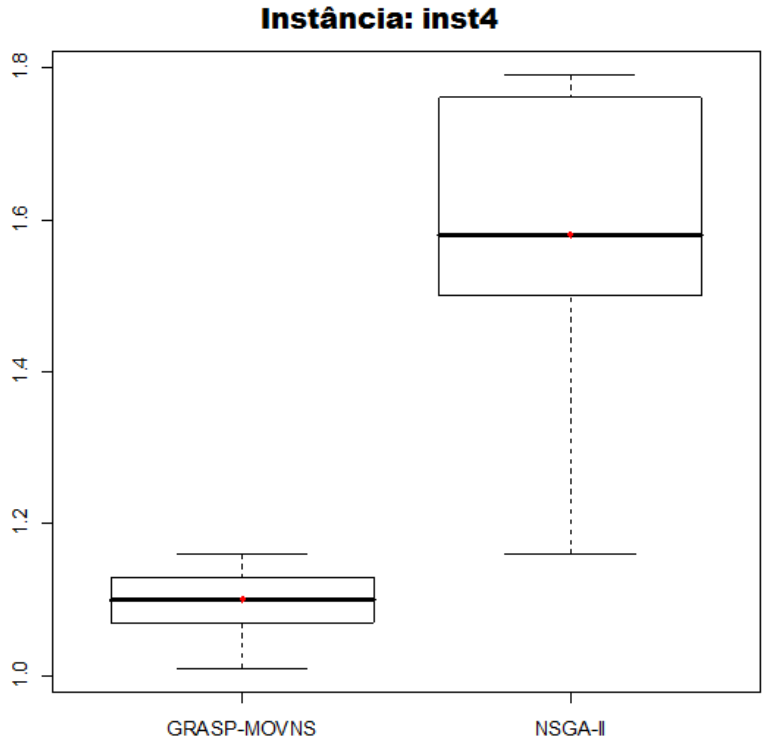

Figura 6.12: Gráficos *BoxPlot* para Métrica Epsilon da instância *inst*4.

6.4 Fronteiras de Pareto

Nessa seção são apresentados exemplos de fronteira de Pareto geradas pelos algoritmos GRASP-MOVNS e NSGA-II para cada uma das instâncias.

As Figuras 6.13 e Figuras 6.14 apresentam uma fronteira de Pareto gerada para as instâncias *inst*1 e *inst*2, respectivamente. Como pode ser observado, o algoritmo NSGA-II gerou fronteiras de Pareto totalmente dominadas pelas fronteiras geradas pelo GRASP-MOVNS para as duas instâncias. As fronteiras geradas pelo NSGA-II indicam uma possível exploração ineficiente de determinadas regiões do espaço de busca. Pode-se observar também que os dois algoritmos geraram soluções bem diversificadas

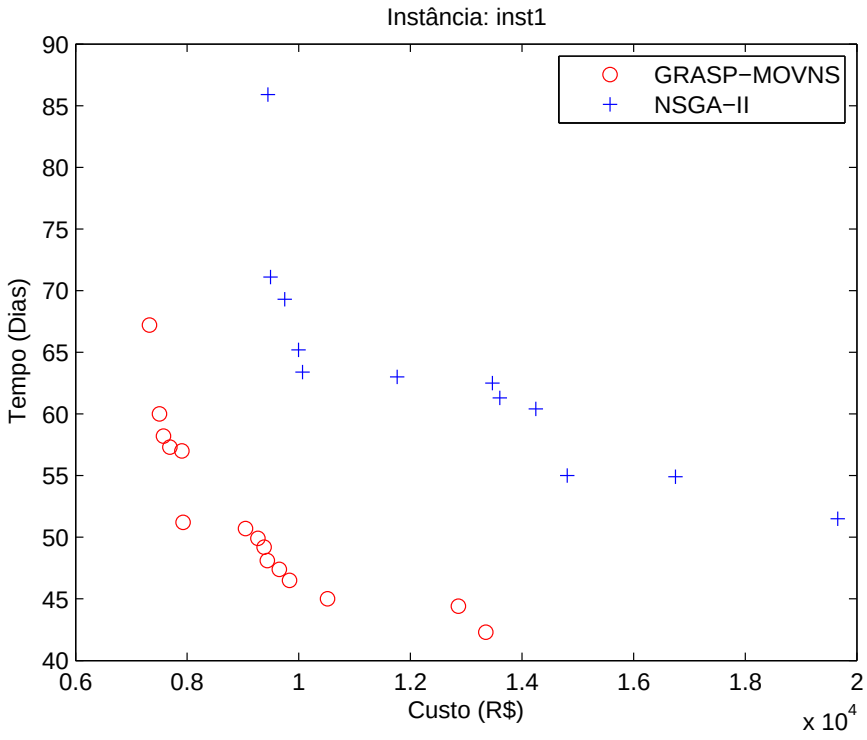

Figura 6.13: Exemplo de fronteira de Pareto para instância *inst*1.

Como pode ser visto nas Figuras 6.15 e 6.16, o algoritmo NSGA-II gerou soluções mais próximas das geradas pelo GRASP-MOVNS.

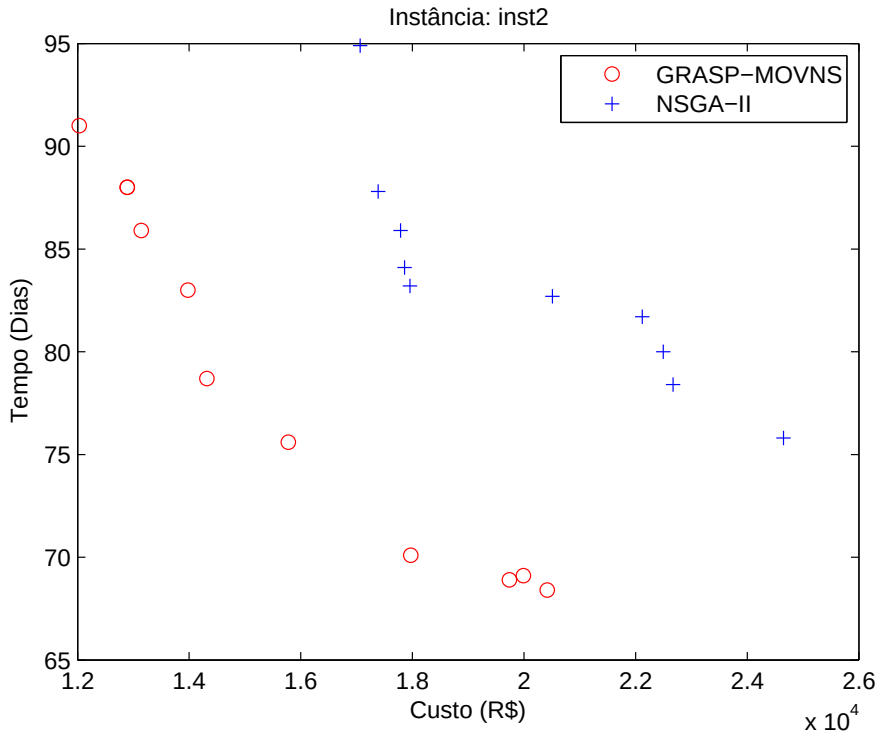

Figura 6.14: Exemplo de fronteira de Pareto para instância *inst*2.

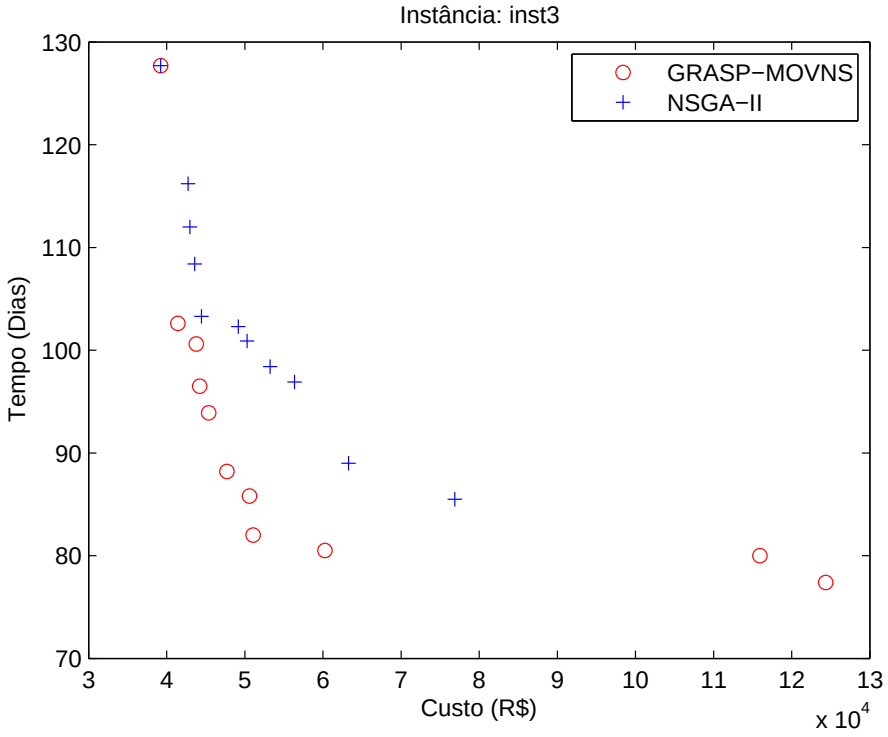

Figura 6.15: Exemplo de fronteira de Pareto para instância *inst*3.

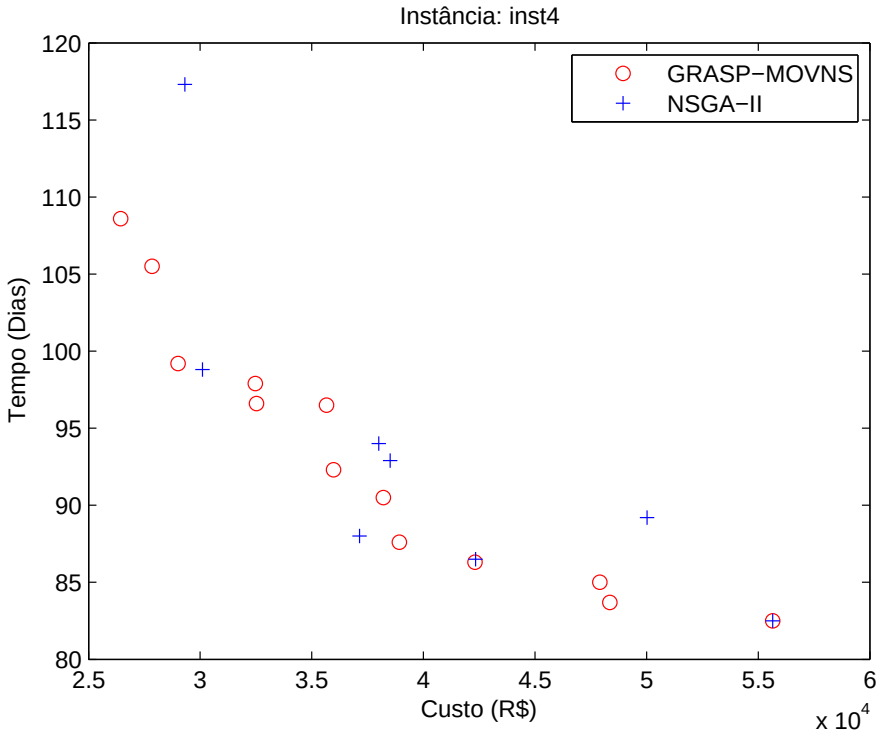

Figura 6.16: Exemplo de fronteira de Pareto para instância *inst*4.

Capítulo 7

Conclusões e Trabalhos Futuros

Esse trabalho trata o Problema de Desenvolvimento de Cronogramas (PDC) com o uso de metaheurísticas. Para a solução do problema em questão, é utilizada uma modelagem matemática adaptada da modelagem proposta por Colares (2010). Na seção 2.3.1, é apresentada a modelagem proposta por Colares (2010) e, na Seção 2.3.3, é definida a modelagem adaptada para esse trabalho. As adaptações realizadas têm, como principal objetivo, melhorar o desempenho do algoritmo proposto, de modo que trabalhe somente com soluções válidas para o problema estudado. É utilizado um modelo matemático simplificado, mantendo somente as principais características que definem o problema em questão. Dessa forma, todos os elementos apresentados na modelagem foram testados.

Para resolver o problema, foram implementados os algoritmos multiobjetivo GRASP-MOVNS e o NSGA-II. O algoritmo NSGA-II foi implementado como descrito na literatura em pesquisas sobre SBSE. O GRASP-MOVNS implementado nesse trabalho é uma variante do GRASP-MOVNS encontrado na literatura. Essa variante consiste na adaptação de 9 movimentos de vizinhança específicos para explorar o espaço de soluções do PDC.

Após a validação dos algoritmos por meio dos experimentos realizados, foram feitas comparações entre os algoritmos implementados utilizando as métricas de Cardinalidade, Hipervolume e *Epsilon*. O resultado das comparações das três métricas está apresentado no Capítulo 6. Foram realizados testes estatísticos de gráficos *Boxplot*, mostrados na Seção 6.3, e, posteriormente, são mostrado os gráficos de dominância de Pareto, na Seção 6.4.

Após a apresentação dos resultados, foi possível destacar o algoritmo GRASP-MOVNS como o melhor algoritmo implementado neste trabalho. Os resultados computacionais realizados em instâncias de projetos de desenvolvimento de software reais mostram que o GRASP-MOVNS é superior ao NSGA-II em relação às três métricas avaliadas. Na avaliação dos algoritmos pela dominância de Pareto, apresentados na Seção 6.4, pode ser observado que o NSGA-II apresentou desempenho muito ruim, principalmente para as instâncias $inst1$ (Figura 6.13) e $inst2$ (Figura 6.14), nas quais obteve fronteiras de Pareto totalmente dominadas pelas soluções do GRASP-MOVNS.

O objetivo geral do projeto de propor um algoritmo multiobjetivo eficiente para resolver o Problema de Desenvolvimento de Cronogramas de um projeto de software foi, neste sentido, atingido, conforme apresentado nos Capítulos 5 e 6. Os objetivos

específicos listados na Subseção 1.3.2 também foram atingidos.

Desta forma, fica evidenciada a contribuição científica deste trablho, que apresenta a primeira proposta de uso da metaheurística GRASP-MOVNS com os 9 movimentos de vizinhança específicos para resolver o PDC. Como apresentado nas Tabelas 3.2 e 3.3 a grande maioria dos estudos encontrados na literatura sobre o PDC é tratado por meio de algoritmos baseados em busca populacional, sendo por meio de uma abordagem mono-objetivo ou multiobjetivo. O algoritmo GRASP-MOVNS proposto neste trabalho é baseado em busca local, permitindo a comparação de algoritmos multiobjetivos pertencentes a duas classes de problemas, de um lado o GRASP-MOVNS baseado em busca local e, do outro, o algoritmo NSGA-II de busca populacional.

O segundo aspecto é o interesse prático da abordagem, já que o desenvolvimento de cronogramas é uma atividade complexa, que tem impacto direto no sucesso de um projeto de desenvolvimento de software.

Para trabalhos futuros, são feitas as seguintes sugestões:

- Utilização de outras metaheurísticas como SPEA-II, *Simulated Annealing* (SA), *Pareto Iterated Local Search* (PILS) e *MultiObjective Iterated Local Search* (MOILS);
- Comparar com os limites utópicos definidos para o PDC. Para calcular esses limites, é necessário construir uma abordagem mono objetivo para cada um dos objetivos abordados no problema e utilizar o CPLEx para resolver.
- Desenvolver novos operadores para o algoritmo NSGA-II, que permitam uma exploração mais eficiente do espaço de busca para o problema.
- Utilização de novas estruturas de vizinhança para as buscas locais feitas pelo algoritmo GRASP-MOVNS;
- Realizar novos testes estatísticos nos algoritmos NSGA-II e GRASP-MOVNS para classificação automática dos mesmos.
- Realizar novos testes em novos conjuntos de instâncias de projetos reais de maior dimensão.

Capítulo 8

Publicações

A seguir são listados os trabalhos oriundos desta dissertação que foram apresentados e publicados em anais de eventos:

1. **Título**: The MOVNS metaheuristic for the Problem of Development Schedule for Software Project
   Autores: Sophia Nóbrega, Sérgio Ricardo de Souza e Marcone Jamilson Freitas Souza
   Evento: 4th Symposium on Search Based Software Engineering (SSBSE-2012)
   Data: 28 a 30 de Setembro de 2012
   Local: Trento, Itália

2. **Título**: Uma Abordagem MultiObjetivo para o Problema de Desenvolvimento de Cronogramas de Projetos de Software
   Autores: Sophia Nóbrega, Sérgio Ricardo de Souza e Marcone Jamilson Freitas Souza
   Evento: IV Workshop de Engenharia de Software Baseada em Busca (WESB-2013)
   Data: 29 de Setembro de 2013
   Local: Brasília, Brasil

3. **Título**: Metaheurística MOVNS para o Problema de Desenvolvimento de Cronogramas de Projetos de Software
   Autores: Sophia Nóbrega, Sérgio Ricardo de Souza e Marcone Jamilson Freitas Souza
   Evento: X Encontro Nacional de Inteligência Artificial e Computacional (ENIAC=2013)
   Data: 20 a 24 de Outubro de 2013
   Local: Fortaleza, Brasil

Referências Bibliográficas

Aguilar-Ruiz, J.; Ramos, I.; Riquelme, J. C. e Toro, M. (2001). An evolutionary approach to estimating software development projects. *Information and Software Technology*, v. 43, n. 14, p. 875–882.

Alba, E. e Chicano, F. (2007). Software project management with GAs. *Information Sciences*, v. 177, n. 11, p. 2380–2401.

Antoniol, G.; Di Penta, M. e Harman, M. (2004)a. A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty. *In Proceedings of the 10th International Symposium on the Software Metrics*, p. 172–183.

Antoniol, G.; Di Penta, M. e Harman, M. (2004)b. Search-based techniques for optimizing software project resource allocation. *In Proceedings of the 2004 Conference on Genetic and Evolutionary Computation*, v. 3103/2004, p. 1425–1426.

Antoniol, G.; Di Penta, M. e Harman, M. (2005). Search-based techniques applied to optimization of project planning for a massive maintenance project. *In Proceedings of the 21st IEEE International Conference on Software Maintenance*, p. 240–249.

Arroyo, J. E. C.; dos Santos Ottoni, R. e dos Santos, A. G. (2011). Algoritmo vns multi-objetivo para um problema de programação de tarefas em uma máquina com janelas de entrega. *XLIII SBPO - Simpósio Brasileiro de Pesquisa Operacional*, p. 15–18.

Bagnall, A.; Rayward-Smith, V. e Whittley, L. (2001). The next release problem. *Information and Software Technology*, p. 883–890.

Barreto, A.; Barros, M. O. e Werner, C. M. L. (2008). Staffing a software project: A constraint satisfaction and optimization-based approach. *Computers and Operations Research*, v. 35, p. 3073–3089.

BFPUG,. Bfpug - brazilian function point users group, (2013). URL `http://www.bfpug.com.br/`. [Online; acessado 01-Setembro-2013].

Blum, C. e Roli, A. (2013). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, v. 35, n. 3, p. 268–308.

Boehm, B. (2000). COCOMO ii model definition manual. Technical report, University of Southern California - Center for Software Engineering.

Britto, R.; Neto, P. S.; Rabelo, R.; Ayala, W. e Soares, T. (2012). A hybrid approach to solve the agile team allocation problem. *Evolucionary Computing CEC*, p. 1–8, (2012).

Chang, C.; Chao, C.; Hsieh, S. e Alsalqan, Y. (1994). SPMNet: a formal methodology for software management. *Proceedings of the 18th Annual International Computer Software and Applications Conference*, p. 57.

Chang, C. K.; Chao, C.; Nguyen, T. T. e Christensen, M. J. (1998). Software project management net: a new methodology on software management. *In Proceedings of the 22nd Annual International Computer Software and Applications Conference*, p. 534–539.

Chang, C. K.; Christensen, M. J. e Zhang, T. (2001). Genetic algorithms for project management. *Annals of Software Engineering*, p. 107–139.

Chang, C. K.; Jiang, H.; Di, Y.; Zhu, D. e Ge, Y. (2008). Time-line based model for software project scheduling with genetic algorithms. *European Journal of Information and Software Technology*, v. 50, p. 1142–1154.

Coelho, V. N.; Souza, M. J. F.; Coelho, I. M.; Guimarães, F. G. e Lust, T. (2012). Algoritmos multiobjetivos para o problema de planejamento operacional de lavra. *Anais do XV Simpósio de Pesquisa Operacional e Logística da Marinha - SPOLM 2012*, (2012).

Colares, F. (2010). Alocação de equipes e desenvolvimento de cronogramas em projetos de software utilizando otimização. Dissertação de mestrado, UFMG - Universidade Federal de Minas Gerais.

Colares, F.; Souza, J.; Carmo, R.; Pádua, C. e Mateus, G. R. (2009). A new approach to the software release planning. *XXIII Brazilian Symposium on Software Engineering*, p. 207–215.

Datta, D.; Fonseca, C. M. e Deb, K. (2008). A multi-objetive evolucionary algorithm to exploit the similarities of resource allocation problems. *Springer Science+Business Media*, v. 11, p. 405–419.

Deb, K.; Pratap, A.; Agarwal, S. e Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transaction on Evolutionary Computation*, v. 6, p. 181–197.

Di Penta, M.; Harman, M. e Antoniol, G. (2011). The use of search-based optmization techniques to schedule and staff software projects: an approach and an empirical study. *Software - Practice and Experience*, v. 41, p. 495–519.

Di Penta, M.; Harman, M.; Antoniol, G. e Qureshi, F. (2007). The effect of communication overhead on sofware maintenance project staffing: a search-based approach. *Software Maintenance, ICMS 2007, IEEE International*, p. 315.

Dolado, J. (2000). A validation of the component-based method for software size estimation. *IEEE Transactions on Software Engineering*, p. 1006–1021.

Dolado, J. (2001). On the problem of the software cost function. *Information and Software Technology*, p. 61–72.

Dutra, C. E. e Montane, F. A. T. (2010). Variable neighborhood search and iterated local search aplicados ao problema de rede de transporte rodoviário com carga fracionada. *XLII Simpósio Brasileiro de Pesquisa Operacional.*

Feo, T. A. e Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109–133.

Fitzsimmons, J. A. e Fitzsimmons, M. J. (2005). *Administração de Serviços: Operações, Estratégia e Tecnologia da Informação.* 4 ed. edição.

Geiger, M. J. (2008). Randomized variable neighborhood search for multi objective optimization. *Proceedings of the 4th EU/ME Workshop: Design and Evaluation of Advanced Hybrid Meta-Heuristics*, p. 34–42.

Gueorguiev, S.; Harman, M. e Antoniol, G. (2009). Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering. *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (CECOO'09)*, p. 1673–1680.

Hansen, M. P. e Jaszkiewicz, A. (1998). *Evaluating the quality of approximations to the non-dominated set.* IMM, Department of Mathematical Modelling, Technical Universityof Denmark.

Hansen, P. e Mladenovic, N. (1997). Variable neighborhood search. *Computers and Operations Research*, v. 24, p. 1097–1100.

Harman, M. (2006). Search-based software engineering for maintenance and reengineering. *In Proceeding of the 10th European Conference on Sotware Maintenance and Reengineering CSMR2006*, p. 311.

Harman, M. (2007). Automated test data generation using search based software engineering. *In Proceedings of the Second International Workshop on Automation of Software Test - International Conference on Software Engineering. IEEE Computer Society.*

Harman, M. e Jones, B. F. (2001). Search-based software engineering. *Information and Software Technology*, , n. 43, p. 833–839.

Harman, M.; Mansouri, S. A. e Zhang, Y. (2009). Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Technical Report TR-09-03, King's College London.

Harman, M.; McMinn, P.; de Souza, J. T. e Yoo, S. (2012). *Search Based Software Engineering: Tecniques, Taxonomy, Tutorial*, volume 7007 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

Hartmann, S. e Kolish, R. (2000). Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, v. 127, p. 394–407.

Hashimoto, K. *Técnicas de Otimização Combinatória Multiobjetivo aplicadas na estimação do desempenho elétrico de redes de distribuição.* Dissertação de doutorado em engenharia, Escola Politécnica da Univerdade de São Paulo, (2004).

Hermadi, I. e Ahmed, M.A. (2003). Genetic algorithm based test data generator. *CEC03 - The 2003 Congress on Evolutionary Computation.*

Inc., Serena Software. Openproj, (2011). URL `http://www.serena.com/products/openproj/`. [Online; acessado 10-Dezembro-2011].

Khor, S. e Grogono, P. (2004). Using a genetic algorithm and formal concept analysis to generate branch coverage test data automatically. *19th International Conference on Automated Software Engineering*, p. 346–349.

Kolish, R. e Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, v. 174, p. 23–37.

Lai, K. K. e Li, Lushu. (1999). A dynamic approach to multiple objetive resource allocation problem. *European Journal of Operational Research*, p. 293–309.

Li, Z.; Harman, M. e Hierons, R. M. (2007). Search algorithms for regression test case prioritization. *IEEE Transactions on Software Engineering*, p. 225–37.

Lopes, Heitor S.; Rodrigues, Luiz Carlos A. e Steiner, Maria T. A. (2013). *Metaheurísticas em Pesquisa Operacional.* Omnipax, 1 edição edição.

Maciel, A. C. M.; Martinhon, C. A. e Ochi, L. S. (2005). Heurísticas e metaheurísticas para o problema do caixeiro viajante. *XXXVII Simpósio Brasileiro de Pesquisa Operacional.*

Maia, C.; Carmo, R.; Campos, G. e Souza, J. (2008). A reactive grasp approach for regression test case prioritization. *XL Simpósio Brasileiro de Pesquisa Operacional.*

Michael, C.C.; McGraw, G. e Schatz, M. (2001). Generating software test data by evolution. *Proceedings of IEEE Transactions on Software Engineering*, v. 27, n. 12, p. 1085–1110.

Microsoft,. Ms project 2010, (2011). URL `http://www.microsoft.com/project/en-us/project-management.aspx`. [Online; acessado 10-Dezembro-2011].

Miller, W. e Spooner, D. L. (1976). Atomatic generation of floating-point test data. *IEEE Transactions on Software Engineering*, v. SE-2, n. 3, p. 223–226.

Minku, L. L.; Sudholt, D. e Yao, X. (2012). Evolutionary algorithms for the project scheduling problem: runtime analysis and improved design. *GECCO'12*, p. 1221–1228, (2012).

O'Keeffe, M. e Cinnéide, M. Ó. (2004). Towards automated design improvement through combinatorial optimization. *In Proceedings of the 26th International Conference on Software Engineering and Workshop on Directions in Software Engineering Environments*, p. 75–82.

Pareto, V. (1896). Cours d'economie politique. *F. Rouge*.

Paula Filho, W. P. (2009). *Engenharia de Software: Fundamentos, Métodos e Padrões*. 3 edição edição.

PMI, Project Management Institute. *Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK)*, 4 edição edição, (2008).

Project, The GNOME. GNOME planner, (2012). URL `https://live.gnome.org/Planner`. [Online; acessado 10-Abril-2012].

Rocha, I. M.; Viana, G. V. R. e Souza, J. T. (2011). Uma abordagem otimizada para o problema de alocação de equipes e escalonamento de tarefas para obtenção de cronogramas eficientes. *II Workshop de Engenharia de Software Baseada em Buscas - WESB*, v. 12, p. 41–48.

Simons, C. L. e Parmee, I. C. (2008)a. Agent-based support for interactive search in conceptual software engineering design. *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, p. 1785–1786.

Simons, C. L. e Parmee, I. C. (2008)b. User-centered, evolutionary search in conceptual software design. *In Proceedings of the IEEE Congress on Evolutionary Computation*, p. 869–876.

Srinivas, N. e Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, v. 2, n. 3, p. 221–248.

Vergilio, S. R.; Colanzi, T. E.; Pozo, A. T. R. e Assunção, W. K. G. (2011). Search based software engineerig: A reviw from the braziliam symposium on software engineering. *XXV Simpósio Brasileiro de Engenharia de Software*, v. 1.

Viana, A. e de Sousa, J. P. (2000). Using metahuristics in multiobjective resource constrained projet scheduling. *European Journal of Operational Research*, p. 359–374.

Xanthakis, S.; Ellis, C.; Skourlas, C.; Gall, A. L.; Katsikas, S. e Karapoulios, K. (1992). Aplication of genetic algorithmsto software testing. *In Proceddings of the 5th International Conference on Software Engineering and Aplications*, p. 625–636.

Zhang, Y.; Harman, M. e Mansouri, S. A. (2007). The multi-objective next release problem. *GECCO07*.

Zitzler, E.; Fonseca, C. M.; Knowles, J. D. e Thiele, L. (2005). A tutorial on the performance assessment of stochastic multiobjective optimizers. *Third International Conference on Evolutionary Multi-Criterion Optimization (EMO)*, v. 216.

Zitzler, E. e Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, v. 3, n. 4, p. 257–271.