ALOCAÇÃO DE EQUIPES E DESENVOLVIMENTO DE CRONOGRAMAS EM PROJETOS DE SOFTWARE UTILIZANDO OTIMIZAÇÃO

FELIPE COLARES

Orientador: Claríndo Isaías Pereira da Silva e Pádua

ALOCAÇÃO DE EQUIPES E DESENVOLVIMENTO DE CRONOGRAMAS EM PROJETOS DE SOFTWARE UTILIZANDO OTIMIZAÇÃO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte

Junho de 2010

Colares, Felipe
C683a Alocação de Equipes e Desenvolvimento de Cronogramas em Projetos de Software Utilizando Otimização / Felipe Colares. — Belo Horizonte, 2010
xv, 90 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Claríndo Isaías Pereira da Silva e Pádua

1. Search-Based Software Engineering. 2. Alocação de Equipes. 3. Planejamento de Projetos. I. Título.

CDU 519.6*32

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Alocação de equipes e desenvolvimento do cronograma em projetos de software utilizando otimização

FELIPE COLARES TORRES

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. CLARINDO ISAÍAS PEREIRA DA SILVA E PÁDUA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. JERFFESON TEIXEIRA DE SOUZA
Departamento de Computação - UECE

PROF. GERALDO ROBSON MATEUS
Departamento de Ciência da Computação - UFMG

PROF. RODOLFO SÉRGIO FERREIRA DE RESENDE
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 13 de julho de 2010.

Dedico este trabalho a todos que de alguma forma me ajudaram a chegar aqui. Aos meus pais, por terem me ensinado o caminho e por sempre estarem lá nos momentos turvos; aos meus amigos, fieis companheiros de caminhada; aos meus professores, responsáveis por semear em mim o conhecimento necessário, com destaque aos meu orientadores neste trabalho Clarindo e Geraldo Robson, além do prof. Jerffeson Souza; e, finalmente, a Deus.

Agradecimentos

Agradeço a UFMG, aos membros do DCC, meus professores, orientadores e colegas. Agradeço também aos órgãos CNpQ e CAPES, que financiaram minhas bolsas de pesquisa.

"A desobediência é uma virtude necessária à CRIATIVIDADE"
(Raul Seixas)

Resumo

No gerenciamento de projetos de software, o planejamento é considerado fator chave para o sucesso. A alocação de recursos em projetos de software, que compreende atribuição de responsabilidades e definição do cronograma, é reconhecida como uma atividade tão importante quanto complexa. Abordagens ad hoc possuem várias desvantagens, grande parte advinda do fato do desenvolvimento se tornar dependente de indivíduos e não do processo. Visando superar tais questões, esse trabalho se propõe a abordar o problema utilizando técnicas de otimização computacional, com o objetivo de contornar limitações atuais através de uma abordagem automatizada.

Abstract

In software management, planning is recognized as a key aspect to the success. Project scheduling, which includes assigning responsibilities and developing the schedule, is an activity as much important as complex. Ad hoc approaches have many negative points, most of them due the development becoming dependent of individuals rather than the process. Based on this, the present work deals with the problem by using computational optimization techniques to overcome current limitations by presenting an automated approach. Applying optimization techniques to software engineering problems is a recent paradigm that has received growing attention by the academic community, leading to a new field called Search-Based Software Engineering (SBSE). This work presents an approach based on multi-objective optimization, both proposing a new formulation to the problem and presenting an algorithm to solve it. Tests and experiments validate the proposed approach and demonstrate its efficacy and applicability.

Resumo Estendido

No gerenciamento de projetos de software, o planejamento é considerado fator chave para o sucesso. A alocação de recursos em projetos de software, que compreende atribuição de responsabilidades e definição do cronograma, é reconhecida como uma atividade tão importante quanto complexa. Abordagens ad hoc possuem várias desvantagens, grande parte advinda do fato do desenvolvimento se tornar dependente de indivíduos e não do processo. Visando superar tais questões, esse trabalho se propõe a abordar o problema utilizando técnicas de otimização computacional, com o objetivo de contornar limitações atuais através de uma abordagem automatizada. A aplicação de técnicas de otimização a problemas da engenharia de software é um recente paradigma que tem ganhado crescente atenção da comunidade acadêmica, constituindo um novo campo denominado Search-Based Software Engineering (SBSE). Esse trabalho apresenta uma abordagem baseada em otimização multiobjetivo, propondo uma nova formulação para o problema e desenvolvendo uma algoritmo para resolvê-la. Testes e experimentos validam a abordagem proposta e demonstram sua eficácia e aplicabilidade.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O gerenciamento de projetos é reconhecido como atividade fundamental aos projetos de software. Segundo Pressman (2005), projetos de software precisam ser gerenciados pois "desenvolver um software é uma tarefa complexa, especialmente se isso envolve muitas pessoas trabalhando durante um tempo relativamente longo". Ele ainda afirma que projetos de software precisam ser planejados e controlados pelo fato dessa ser a única forma conhecida de gerir sua complexidade.

No gerenciamento de projetos de software, o planejamento é considerado fator chave para o sucesso. Kerzner (2009) afirma que "as responsabilidades mais importantes de um gerente de projeto são o planejamento, integração e execução de planos. Quase todos os projetos, devido à sua duração relativamente curta e muitas vezes priorizando o controle dos recursos, requerem um planejamento formal e detalhado".

A alocação de recursos em projetos de software, que compreende a atribuição de responsabilidades e definição do cronograma, é uma atividade de grande importância que se enquadra no contexto supracitado. Sua importância pode ser medida pela pelas palavras de Jones (1994, 1997), quando este afirma que "cronogramas exagerados ou irracionais são provavelmente a principal influência destrutiva a tudo que está relacionado a software". No PMBoK (PMI, 2008), conhecido guia de boas práticas da gerência de projetos, esta atividade enquadra-se no processo denominado desenvolvimento do cronograma.

Tão importante quanto complexa, a alocação de recursos é uma atividade que possui diversas variáveis a serem consideradas, tais como: prazos, custos, interdependências entre tarefas, planejamentos multiprojetos, replanejamentos, estimações e incertezas e manipulação direta de recursos humanos.

Ainda que existam ferramentas, técnicas e manuais de boas práticas relacionados ao assunto, eles não tiram do gerente de projetos a responsabilidade pela obtenção de uma solução de qualidade que esteja de acordo com os objetivos da sua organização. Para

isso, tal solução geralmente é obtida a partir de heurísticas baseadas em seu *feeling*, ou seja, em seu conhecimento, experiência e intuição. Essa abordagem *ad hoc* possui várias desvantagens, grande parte advinda do fato do desenvolvimento se tornar dependente de indivíduos e não do processo. Visando superar tais questões, este trabalho se propõe a abordar o problema utilizando técnicas de otimização computacional, com o objetivo de contornar limitações atuais através de uma abordagem automatizada.

A abordagem proposta consiste principalmente em duas partes: a utilização de um modelo matemático para formular o problema abordado, e o desenvolvimento de um algoritmo a ser aplicado ao modelo. O algoritmo desenvolvido é baseado em técnicas de otimização computacional, mais especificamente em uma conhecida meta-heurística evolutiva multiobjetivo.

A aplicação de técnicas de otimização a problemas da engenharia de software é um recente paradigma que tem ganhado crescente atenção da comunidade acadêmica. Essa nova área, denominada *Search-Based Software Engineering* (SBSE), procura focar-se em problemas de natureza complexa – como é o caso do problema abordado neste trabalho – com o objetivo de oferecer um mecanismo de apoio ao engenheiro de software para a resolução de tais problemas. Assim, pois, o objetivo da abordagem aqui proposta não é substituir a presença do gerente, mas sim oferecer uma ferramenta capaz de apoiá-lo e guiá-lo no problema em questão, a alocação de equipes e desenvolvimento de cronogramas em projetos de software.

Este trabalho considera o problema propondo uma abordagem baseada em otimização multiobjetivo, a qual oferece diversos benefícios a este tipo de problema, como será discutido mais à frente. Como também será mostrado, a formulação proposta engloba diversos e importantes aspectos e variáveis encontrados cotidianamente por gerentes de projetos, tais como custos, prazos, habilidades individuais, inter-relacionamentos, aprendizado, horas extras e etc. Testes e experimentos, realizados em vista de validar e mensurar a abordagem, comprovam sua eficácia e viabilidade.

1.1 Contribuições

Esse trabalho apresenta algumas importantes contribuições à comunidade, conforme se segue:

1. A nova formulação proposta, na qual se considerou diversos fatores importantes intrínsecos ao problema, em vista de torná-la o mais condizente possível com situações reais. Bem fundamentada na literatura relacionada ao assunto, a formulação foca em situações práticas do cotidiano de gerentes de projetos.

2. É apresentado um algoritmo baseado em otimização multiobjetivo evolucionária, desenvolvido para a formulação proposta.

3. O uso de otimização multiobjetivo também pode ser apontado como uma boa contribuição, uma vez que pouquíssimos trabalhos da área fazem uso desse paradigma.

4. A realização de experimentos controlados com a participação de especialistas, em vista de avaliar a qualidade dos resultados e quantificar os ganhos obtidos pela abordagem. Esse tipo de experimento tem se mostrado raro em trabalhos de SBSE e mesmo na própria engenharia de software.

5. A realização de análises de sensibilidade do problema, onde o assunto abordado – a alocação de equipes e desenvolvimento de cronogramas – é estudado e analisado conforme diferentes parâmetros e variáveis.

1.2 Estrutura do Documento

Os capítulos a seguir estão dispostos da seguinte maneira: o capítulo 2 discute sobre a literatura relacionada ao assunto, incluindo uma breve discussão sobre SBSE, além de trabalhos relacionados ao tema e trabalhos semelhantes a este.

O problema abordado no trabalho é discutido no capítulo 3. Além de uma contextualização, são discutidos os aspectos relativos ao problema e os fatores considerados em sua formulação. Em seguida, essa formulação é apresentada e matematicamente modelada. Ao final, sua complexidade é formalmente analisada.

O algoritmo desenvolvido é apresentado no capítulo 4. Nele, são discutidos conceitos de otimização e de técnicas multiobjetivo e é apresentada uma meta-heurística multiobjetivo, utilizada como base para a implementação. O algoritmo em si é apresentado e sua implementação é discutida. Ao final, é realizada uma análise de complexidade do algoritmo.

Testes e experimentos são descritos no capítulo 5, onde são apresentados e discutidos diferentes cenários e seus resultados. Entre os cenários estão um estudo de caso envolvendo dados de uma empresa de desenvolvimento de software, um experimento comparativo realizado com seres humanos e análises de sensibilidade do problema.

Finalmente, o capítulo 6 apresenta a conclusão do trabalho e discute sobre possíveis trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Esse capítulo é divido em três seções. A primeira discute sobre o emergente campo no qual está inserido este trabalho, que envolve engenharia de software e otimização computacional. A segunda apresenta trabalhos relativos a assuntos paralelos ao tema abordado, importantes para sua compreensão e da abordagem proposta. Finalmente, a terceira seção discute sobre trabalhos semelhantes a este, apresentando análises e comparativos.

2.1 Otimização na Engenharia de Software

Técnicas de otimização são utilizadas em várias áreas, incluindo diversas disciplinas de engenharia. Como em grande parte destas, engenheiros de software constantemente se deparam com problemas que envolvem variáveis complexas, como restrições concorrentes, balanceamento de interesses, riscos e imprecisão. Quanto maior sua complexidade, mais difícil e mais suscetível a erros será obter uma solução para o problema. Felizmente, tais problemas normalmente podem ser formulados como problemas de buscas, o que os torna ideais para aplicação de técnicas de otimização. O uso de tais técnicas tende a transferir, pelo menos parcialmente, a responsabilidade do engenheiro de software para a máquina, oferecendo-lhe ferramentas de apoio, e deixando-lhe a responsabilidade de guiar a tarefa em vez de executá-la. Dessa forma o desenvolvimento se torna menos dependente do indivíduo e mais dependente do processo.

Nesse contexto, surge um novo campo da engenharia de software, denominado *Search-Based Software Engineering* (SBSE). Esse campo emergente baseia-se na utilização de técnicas de otimização, principalmente meta-heurísticas, para resolução de problemas da engenharia de software. Esta seção apresenta uma breve discussão acerca dessa nova área, a fim de expandir a visão de engenheiros de software para esse novo paradigma e, assim, contribuir para o seu crescimento e fortalecimento. A seguir, são

apresentados um histórico da área, alguns de seus principais trabalhos e uma discussão sobre seu futuro.

2.1.1 Histórico

Miller e Spooner (1976) publicaram um dos primeiros trabalhos a utilizar métodos de otimização na engenharia de software, o qual fez uso de métodos de maximização numérica para geração de dados de teste. Mais tarde, Xanthakis et al. (1992) aplicaram pela primeira vez uma meta-heurística para resolução de um problema de engenharia de software, em um trabalho baseado na utilização de algoritmos genéticos para geração de casos de teste. Outros trabalhos foram surgindo no decorrer da década de 90, porém esporádicos e em pequena quantidade (Hartmann e Robson, 1989; Cho e Ray, 1995; Jones et al., 1996; Dolado e Fernandez, 1998; Doval et al., 1999).

O termo SBSE surgiu no início dos anos 2000, no primeiro *survey* sobre o assunto, de Harman e Jones (2001). A partir desse trabalho, pôde-se observar um grande crescimento nas atividades de SBSE em diversos domínios da engenharia de software, o que motivou o gradual surgimento de novos *surveys*: Clarke et al. (2003) apresentam um *guideline* para trabalhos futuros em SBSE; Harman (2007b) apresenta um paralelo entre passado, presente e futuro da área; e Harman (2010) discute por que a engenharia de software oferece um campo ideal para aplicação de técnicas de otimização.

2.1.2 Estado da Arte

Esta subseção apresenta e discute os principais domínios da engenharia de software aos quais SBSE tem sido aplicada com sucesso. Há trabalhos relacionados a praticamente todas as atividades do processo de desenvolvimento de software, desde requisitos, passando por arquitetura, desenho, codificação, testes e manutenção, até as de suporte, como gerenciamento de projetos e garantia da qualidade.

No campo de *requisitos*, uma das principais aplicações de SBSE é no planejamento de liberações, que consiste em determinar quais requisitos devem ser implementados em cada liberação do software (Colares et al., 2009; Ruhe e Saliu, 2005). Também valem ser ressaltados dois trabalhos para auxílio na tomada de decisões relacionadas a requisitos (Feather et al., 2006; Harman et al., 2009), além de um *survey* sobre requisitos de software e SBSE, de Zhang et al. (2008).

Em relação ao *desenho e arquitetura de software*, SBSE tem sido utilizada com diferentes propósitos, como a geração (Räihä, 2008; Lutz, 2001) ou o melhoramento (Amoui et al., 2006; Grunske, 2006) de soluções arquiteturais, o uso de refatoração visando otimizar o acoplamento e coesão modular (Mitchell e Mancoridis, 2006), a

atribuição de responsabilidades a classes/objetos (Bowman et al., 2007) e a seleção (Vijayalakshmi et al., 2008) ou substituição (Desnos et al., 2008) de componentes no paradigma da programação orientada a componentes.

Apesar de ser uma atividade altamente baseada nos intelecto e esforço individuais, a *codificação de software* também apresenta bom potencial para a aplicação de SBSE. Katz e Peled (2010) apresentam um método para verificação e correção automática de código fonte. Shevertalov et al. (2009) propõem a identificação de autores baseada em métricas para identificação dos perfis dos desenvolvedores. Hoste e Eeckhout (2008) apresentam uma abordagem para otimização automática de compilação.

A maioria das atividades de SBSE é relacionada à área de *testes de software*, principalmente na geração de dados de teste. McMinn (2004) e Harman (2007a) apresentam *surveys* sobre o assunto. Outro problema bem abordado é a geração e seleção (ou priorização) de casos de teste, como nos trabalhos de Ali et al. (2009), Baudry et al. (2005) e Li et al. (2007).

No campo da *manutenção*, a refatoração de software tem sido a principal área estudada com uso de SBSE. Penta et al. (2005) apresentam um arcabouço de ferramentas para refatoração e renovação de software, enquanto um estudo empírico acerca da refatoração de software utilizando SBSE é apresentado por O'Keeffe e Cinnéide (2008).

Devido à complexidade envolvida em suas tarefas, a *gerência de projetos de software* tem se mostrado uma área com grande potencial para aplicação de SBSE. Além do tema abordado neste trabalho, que é discutido na seção 2.3, outros problemas têm sido bem estudados nessa área. Um dos principais tópicos é a estimação de software, problema abordado por Dolado (2000, 2001) e Sheta (2006). O primeiro combina programação genética com a técnica de regressão simbólica, enquanto o segundo propõe um modelo que utiliza algoritmo genético juntamente com o conhecido modelo COCOMO (Boehm et al., 2000). Outros trabalhos também podem ser destacados, como o modelo de previsão de falhas, de Catal e Diri (2009), e a utilização de integração fuzzy para a predição da qualidade, de Pizzi (2008).

Mais trabalhos podem ser encontrados em (SBSE Repository, 2010), um repositório de publicações de SBSE mantido por Yuanyuan Zhang.

2.1.3 O Futuro de SBSE

Segundo Harman (2007b), o surgimento de um novo campo da ciência segue um caminho natural dividido em duas fases. A primeira, denominada "corrida pelo ouro", caracteriza-se pelo entusiasmo e por atividades não bem direcionadas. Já em uma segunda fase, é necessário um conhecimento mais profundo sobre características dos problemas e de suas soluções, de forma a criar raízes maduras que permitirão o cres-

cimento do novo campo. Ainda segundo Harman, SBSE encontrava-se na transição entre as duas fases naquela ocasião.

Passados três anos da afirmação de Harman, pode-se observar o início de uma aparente estabilização no crescimento quantitativo da área, conforme ilustra a figura 2.1. Tal fato aponta para o término da transição entre as duas fases supracitadas e o consequente crescimento na busca pelo amadurecimento da área. Observa-se que alguns domínios já se mostram bem maduros no âmbito acadêmico, como é o caso da área de testes, enquanto outros parecem caminhar firmemente na mesma direção. Em paralelo, novas áreas continuam a surgir com novos problemas sendo abordados com o uso de SBSE.

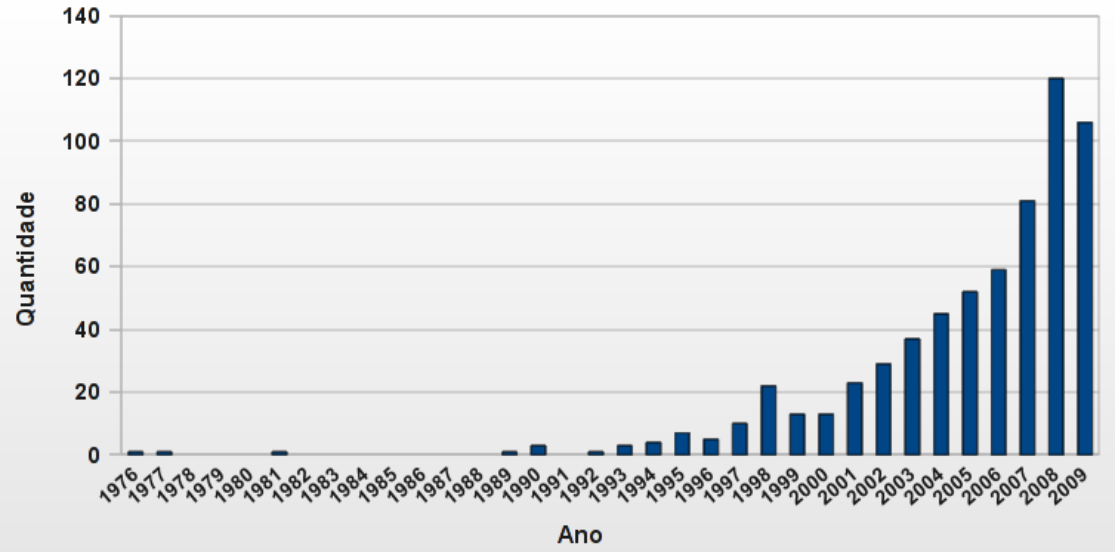

Figura 2.1. Crescimento quantitativo de SBSE

Para o autor deste trabalho, o que se pode esperar desse campo ainda emergente é que continue se fortalecendo pelos próximos anos, principalmente em relação à sua qualidade e ao seu amadurecimento. Naturalmente, dado seu curto tempo de vida, esse campo ainda possui inúmeros desafios a serem superados, como uma maior interatividade humana com as abordagens e a competitividade dos resultados de SBSE com resultados obtidos por seres humanos sem o seu uso. Ambas as questões são consideradas neste trabalho.

2.2 Assuntos relacionados

Esta seção discute alguns trabalhos cujo assunto está diretamente relacionado com o tema deste trabalho e que foram importantes referências para a sua realização.

Estimações constantemente se mostram necessárias às atividades do planejamento de projetos. No desenvolvimento do cronograma de um projeto, o esforço necessário e o tempo para a execução de cada tarefa são exemplos de dados que se enquadram nesse caso. Se o primeiro é estimado independentemente da alocação das equipes, o mesmo não ocorre com o segundo, o qual sofre interferência direta dessa alocação, uma vez que a equipe é fator determinante no tempo de execução de uma tarefa. Diante disso, a abordagem proposta neste trabalho leva em consideração a questão da estimação de tempo, tendo, para isso, o conhecido método de estimação COCOMO-II (Boehm et al., 2000) como importante referência nesse sentido. Como será mostrado mais adiante, estimado o esforço necessário para a realização de cada tarefa, seja utilizando o próprio COCOMO ou qualquer outro método de estimação, a abordagem proposta é capaz de estimar o tempo de duração da tarefa em função das equipes alocadas pela própria abordagem. Para isso, são utilizados conceitos do método COCOMO relativos a capacidade e experiência de pessoal.

Outros fatores também interferem na produtividade e, consequentemente, na estimação do tempo de execução das tarefas. A lei de Brooks (1995) é um conhecido princípio da engenharia de software que afirma que quanto maior for a equipe menor será sua produtividade individual, devido ao *overhead* de comunicação. Como será mostrado, a abordagem proposta baseia-se nesse conceito para formular a perda de produtividade relativa à comunicação interpessoal.

Modelos de aprendizado são aplicados para determinar o ganho de produtividade na realização de uma tarefa devido à experiência adquirida durante sua execução. A abordagem proposta utiliza os trabalhos de Badiru (1992), Zorgios et al. (2009), Raccoon (1996) e Kerzner (2009) como referências para a modelagem de uma curva temporal de ganho relativo ao aprendizado.

Tendo em vista o cotidiano de organizações de desenvolvimento de software, a realização de projetos simultâneos e consecutivos é uma realidade. Assim, o gerenciamento multiprojetos precisa ser considerado sempre que for necessária a utilização compartilhada de recursos da organização, principalmente os humanos. Nesse sentido, trabalhos que abordam o gerenciamento e a alocação de recursos multiprojetos, como os de Engwall e Jerbrant (2003), Lee e Miller (2004) e Turner (2009) também foram importantes referências para o desenvolvimento desse trabalho.

Finalmente, porém não menos importante, o livro de Deb (2001) representou uma importante base para o uso dos conceitos de técnicas de otimização multiobjetivo utilizados no trabalho.

2.3 Alocação de Equipes em Projetos de Software

O bom gerenciamento de recursos é um ponto chave para o sucesso de um projeto. Há vários modelos, técnicas e ferramentas que visam apoiar a gerência de projetos no que tange a alocação de equipes, controle de pessoal e desenvolvimento de cronogramas, seja para projetos em geral ou no âmbito específico de projetos de software. A abordagem proposta neste trabalho enquadra-se nesse escopo, assim como os trabalhos discutidos nesta seção.

A ISO 10006 (ISO, 2003), norma internacional para gerenciamento de qualidade em projetos, discute sobre aspectos do gerenciamento de recursos humanos, em termos de planejamento e controle, incluindo um processo de alocação de pessoal. O PMBoK (PMI, 2008), conhecido guia de boas práticas na gerência de projetos, possui escopo semelhante e inclui uma área de conhecimento denominada gerenciamento de recursos humanos. A alocação de recursos, porém, é compreendida nele como parte da área de gerenciamento de tempo, mais especificamente nos processos de desenvolvimento do cronograma e de controle do cronograma. De fato, a alocação em si é abordada no primeiro, enquanto o segundo está relacionado ao seu andamento e a eventuais mudanças.

Além dos aspectos relativos à alocação, como as entradas e saídas do processo, o guia PMBoK sugere ferramentas e técnicas para suporte às suas atividades. Softwares de planejamento de projeto, tais como MS Project (Microsoft Corporation, 2010) e OpenProj (Serena Software, 2010), são bem conhecidas e amplamente utilizadas nesse sentido. Essas ferramentas, porém, apresentam limitações em relação ao gerenciamento otimizado (Chang et al., 2008; Schwaber, 2002), principalmente no que se refere a recursos humanos (Plekhanova, 1998). Visando superar tais limitações, a abordagem proposta neste trabalho pode ser vista como uma ferramenta de apoio que busca englobar grande parte dos aspectos relativos ao processo a fim de possibilitar a ele um maior grau de automatização e, consequentemente, menor tempo e maior confiabilidade e qualidade em sua execução.

Diferentes abordagens têm sido utilizadas na tentativa de tratar da alocação de equipes e do desenvolvimento de cronogramas no planejamento de projetos. As duas próximas subseções discutem sobre alguns trabalhos nesse sentido, sendo a subseção 2.3.1 para técnicas em geral e a subseção 2.3.2 especificamente para técnicas baseadas em SBSE.

2.3.1 Alocação de Equipes com Técnicas em Geral

Earned Value (Fleming e Koppelman, 2006) é uma técnica desenvolvida por iniciativa do governo americano para ser utilizada em gerenciamento de projetos, em especial em projetos de software. Ao analisar etapas iniciais de um projeto em andamento (até 15%), essa técnica possibilita boa precisão ao estimar parâmetros futuros do projeto, assim auxiliando o controle, dentre outros, de seu cronograma. Essa técnica pode ser utilizada para controle e (re)planejamento de cronograma, porém, diferentemente da abordagem proposta nesse trabalho, ela não entra no âmbito do planejamento inicial do projeto.

No intuito de realizar a alocação de recursos em projetos, Wei et al. (2002) utilizam conceitos da teoria das restrições (Goldratt e Cox, 2004), uma filosofia de negócios que visa alcançar os objetivos de uma organização contornando as restrições que impedem ou dificultam o alcance dessa meta. A abordagem proposta nesse trabalho é direcionada para projetos em geral, e, portanto, não considera aspectos especiais de projetos de software, como o foco em recursos humanos.

Padberg (2006) utiliza um modelo estocástico de decisões de Markov com o objetivo de desenvolver um guia para desenvolvimento de cronograma e alocação de recursos em projetos de software. O autor trabalha com o conceito de políticas de alocação, procurando avaliá-las através de simulações de projetos com o modelo proposto. Esse trabalho não apresenta uma nova técnica para alocação de equipes, mas sim um estudo que resulta em interessantes conclusões sobre o assunto.

O método Taguchi (Taguchi e Hsiang, 1988) é uma abordagem voltada ao conceito de qualidade, o qual busca diminuir os chamados ruídos que causam perda da qualidade do produto, processo ou alvo em questão. Fazendo uma associação entre ruídos internos e a incerteza acerca das tarefas a serem executadas em um projeto de software, Tsai et al. (2003) apresentam um método para alocação de equipes de software com o objetivo de minimizar custo e duração do projeto. O método foi desenvolvido sobre dois casos específicos de pequenos projetos industriais, de forma que a generalização do modelo para projetos em geral e para projetos de larga escala precisa ser melhor analisada.

Silva (2007) descreve um componente de software baseado em políticas para apoio à alocação de pessoas em processos de software, onde, a partir de políticas previamente definidas pelo gerente, o componente provê-lhe sugestões de alocação de pessoal. Para isso, as políticas são constituídas de condição, ação e restrições. Infelizmente, esses parâmetros limitam a abordagem a configurações simples que impossibilitam a consideração de diversas variáveis mais complexas envolvidas no problema, como algumas das tratadas neste trabalho.

Para o surpresa do autor, trabalhos na área de inteligência artificial para alocação de equipes mostraram-se escassos na literatura. À parte alguns poucos trabalhos relativos ao *Resource-constrained Project Scheduling Problem* (RCPSP)[1], como os de Colak et al. (2006) e Jedrzejowicz e Ratajczak-Ropel (2007), o autor nada conseguiu encontrar acerca da alocação de recursos utilizando técnicas de inteligência computacional.

2.3.2 Alocação de Equipes com SBSE

Como discutido anteriormente, técnicas de otimização têm ganhado crescente atenção na resolução de problemas da engenharia de software. Para o problema de alocação de equipes em projeto de software isso não é diferente. Essa seção discute alguns trabalhos de SBSE que se propõem a tratar dessa questão.

Resource-constrained Project Scheduling Problem (RCPSP)[1] é um tradicional problema que consiste na alocação de recursos a tarefas de um projeto. Diversas abordagens baseadas em otimização computacional já foram propostas na tentativa de resolvê-lo, conforme sintetiza um recente *survey* sobre o assunto, de Kolisch e Hartmann (2006). Apesar de conhecido e tradicional, o RCPSP infelizmente apresenta diversas limitações para o gerenciamento de projetos de software, principalmente por ser um problema genérico, o qual desconsidera os pormenores do desenvolvimento de software.

Antoniol et al. (2004) aplicaram algoritmo genético combinado com teoria das filas ao problema de alocação de equipes em projetos de manutenção de software. Na formulação do problema, foram considerados fatores interessantes como erros de estimativa, abandono e retrabalho. Por outro lado, o trabalho apresenta algumas limitações, como o fato de ser voltado para projetos de manutenção e não de desenvolvimento, e também de desconsiderar fatores relevantes, como as experiências e habilidades individuais e as interdependências entre tarefas. A partir de estudos empíricos, os autores concluíram que a utilização de SBSE pode reduzir a duração do projeto em até 50%.

Uma versão multiobjetivo do problema foi proposta por Alba e Chicano (2007), na tentativa de minimizar dois objetivos: o tempo e o custo do projeto. Porém, os autores utilizaram o tradicional algoritmo genético combinando os dois objetivos em uma única função de soma ponderada. A formulação proposta considera aspectos importantes, como habilidades individuais dos recursos. Algumas limitações são a restrita participação do gerente de projetos, que se resume à definição dos parâmetros de entrada do problema, e a impossibilidade de planejamentos multiprojetos. A validação

[1]RCPSP (Brucker et al., 1999) é um conhecido problema da classe de problemas NP-difíceis que se assemelha ao problema abordado nesse trabalho. Apesar de escopo semelhante, os dois problemas apresentam sensíveis diferenças, principalmente no que tange especificidades de projetos de software, como a manipulação de recursos humanos.

da abordagem foi realizada utilizando dados artificiais obtidos a partir de um gerador automático de projetos.

Chang et al. (2008) utilizam o conceito de "linha de tempo" para dividir o problema em pequenos subproblemas aos quais são aplicados algoritmos genéticos com o objetivo de minimizar o custo do projeto. A abordagem apresenta uma formulação bem completa, considerando fatores como experiência acumulada e treinamento de pessoal. Seu principal ponto negativo é a sua complexidade. Para projetos de larga escala, situação em que o gerente poderia tirar maior proveito do uso de SBSE, a dimensão temporal pode tornar o problema inviável mesmo para meta-heurísticas como o algoritmo genético. Nessa situação, uma solução seria considerar espaços de tempo maiores, a fim de minimizar o aumento da complexidade. Porém, essa alternativa provavelmente produziria soluções ruins, uma vez que, na prática, grandes espaços de tempo são incompatíveis com tarefas de curta duração. A abordagem foi testada em um projeto fictício de tamanho médio.

Barreto et al. (2008) formularam o problema como um *constraint satisfaction problem* (Kumar, 1992). Os autores consideram diversos possíveis objetivos na alocação, como equipes mais qualificadas, menos qualificadas, de menor custo ou mais velozes. O problema proposto é resolvido por um algoritmo exato denominado *backtracking* (Golomb e Baumert, 1965), um refinamento do algoritmo de busca por força bruta. Uma ferramenta foi implementada e um experimento *in vitro* com estudantes de mestrado e doutorado foi realizado. Análises qualitativas indicaram que ganhos podem ser obtidos com o uso da ferramenta desenvolvida. Apesar dos pontos positivos, o trabalho apresenta algumas limitações em relação à sua formulação, que considera, por exemplo, que cada tarefa é executada individualmente, não por equipes. Além disso, o experimento foi realizado com uma pequena instância de projeto fictício, de forma que é possível questionar sobre a escalabilidade da abordagem, especialmente por ela utilizar um algoritmo exato.

Outros trabalhos de SBSE apresentam estudos relacionados ao assunto e merecem ser ressaltados: Heričko et al. (2008) propõem uma abordagem que visa otimizar o tamanho das equipes; Di Penta et al. (2007) utilizam SBSE para realizar um estudo acerca do efeito do *overhead* de comunicação em projetos de software; Xiao et al. (2010) formulam um modelo para replanejamentos de cronograma considerando estabilidade e utilidade; finalmente, Ngo-The e Ruhe (2009) apresentam uma abordagem para a alocação de recursos visando o planejamento de liberações de software.

Capítulo 3

Alocação de Equipes e Desenvolvimento de Cronogramas

Este capítulo apresenta o problema abordado pelo trabalho, a alocação de equipes e desenvolvimento de cronogramas em projetos de software. Inicialmente, a seção 3.1 apresenta uma breve discussão contextual. Em seguida, os aspectos considerados na formulação do problema são discutidos na seção 3.2, a formulação em si é apresentada na seção 3.3 e é matematicamente modelada na seção 3.4, enquanto sua complexidade é analisada na seção 3.5. Finalmente, a seção 3.6 discute questões relevantes, como sua aplicabilidade, sua extensibilidade e suas limitações.

3.1 Contextualização

Pressman (2005) define o desenvolvimento do cronograma como "uma atividade que distribui o esforço estimado em toda a duração planejada do projeto alocando esforços a tarefas de engenharia de software específicas". Em outra palavras, uma vez definidas as tarefas do projeto e estimados os esforços necessários, a alocação de recursos é uma atividade que consiste em alocar os esforços disponíveis às tarefas e arranjá-las num cronograma planejado para o projeto.

As figuras 3.1 e 3.2 apresentam, respectivamente, uma visão geral do processo de desenvolvimento do cronograma e um diagrama que ilustra o relacionamento desse processo com os demais processos do desenvolvimento (PMI, 2008). A visão geral apresentada na primeira ilustra a relação entre métodos, ferramentas e modelos utilizados neste processo, enquanto o diagrama da segunda descreve o fluxo de artefatos entre os vários processos.

Ainda que as técnicas ilustradas na figura 3.1 sejam utilizadas para apoiar o desenvolvimento do cronograma e a alocação de equipes, muitas delas não consideram

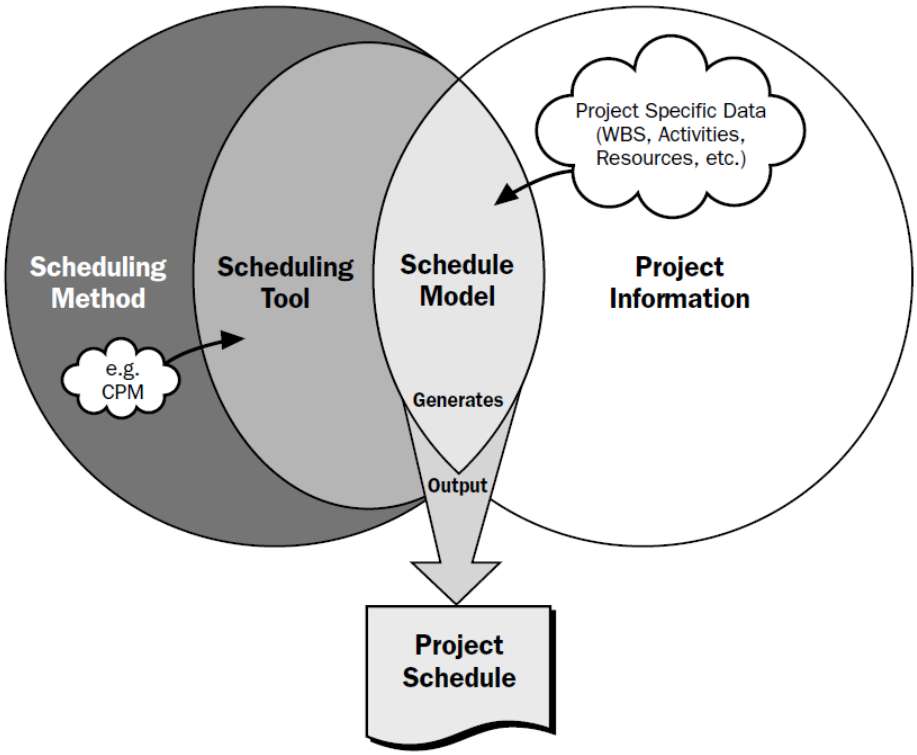

Figura 3.1. Visão geral do processo de desenvolvimento do cronograma (PMI, 2008)

significantes particularidades dos projetos de software. Por esse motivo e também pela complexidade dos fatores envolvidos, o planejamento de cronogramas em projetos de software acaba por basear-se quase que inteiramente no *feeling* do gerente de projetos, de forma que, por essa razão, essa estratégia tradicional pode ser considerada uma abordagem *ad hoc*. Como discutido anteriormente, este trabalho apresenta uma abordagem capaz de superar algumas das limitações atuais relativas a essa questão, automatizando a tarefa (ao menos parcialmente) em vista de torná-la menos dependente de um único indivíduo, e, consequentemente, tornar mais confiável o processo de desenvolvimento.

A figura 3.2 descreve a troca de artefatos entre processos, onde as linhas em negrito representam as relações dentro da área de conhecimento de gerenciamento de tempo e as linhas mais claras representam relações entre diferentes áreas. Conforme ilustra a figura, são entradas básicas para definição do cronograma e alocação de recursos:

- uma lista de atividades, incluindo seus atributos, suas interdependências e o esforço estimado para sua realização;
- os recursos necessários e suas especificações, as quais variam de acordo com a área de aplicação do projeto. Para os de software, recursos humanos devem receber

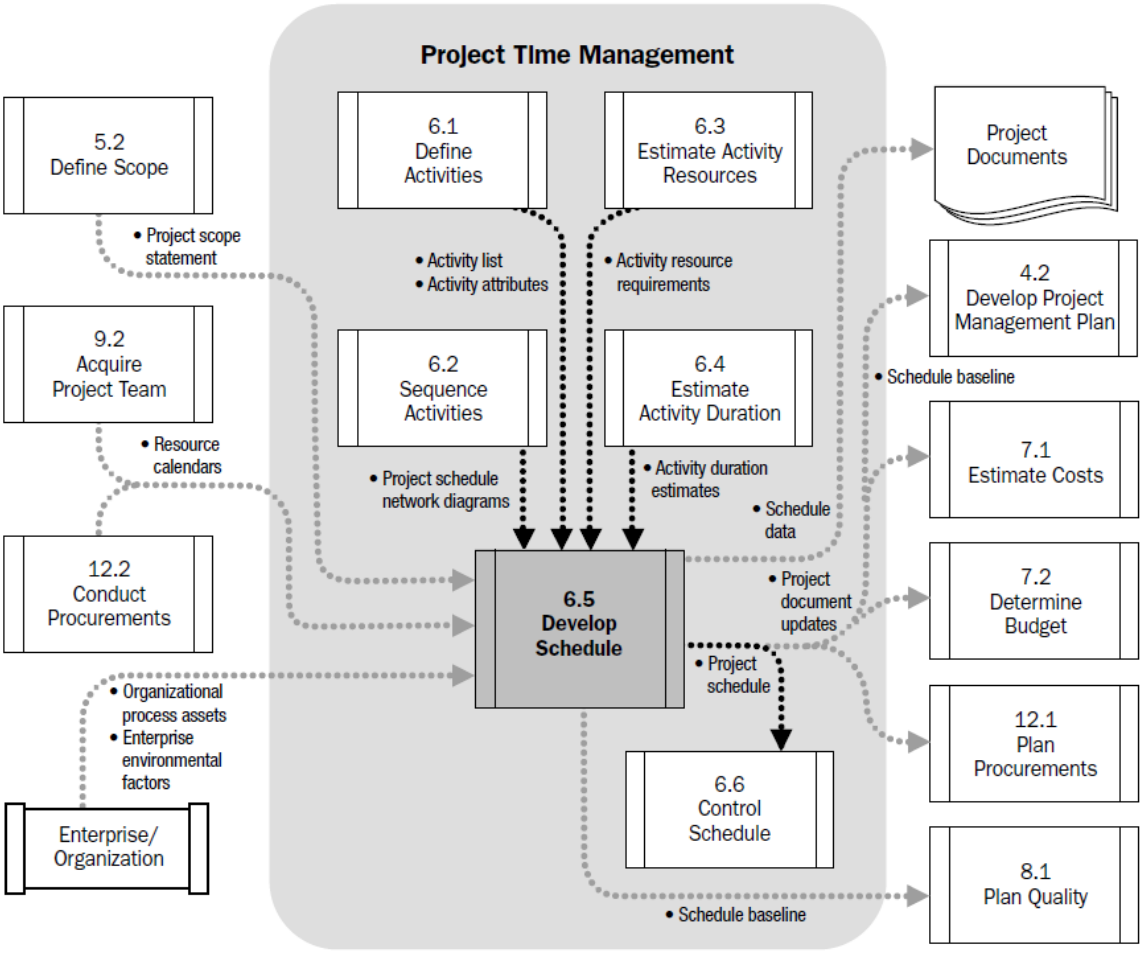

Figura 3.2. Fluxo de dados entre processos (PMI, 2008)

maior atenção, de forma que devem ser especificados fatores como habilidades, experiência e disponibilidade de pessoal; e

- outras informações gerenciais, como escopo do projeto, plano de gerenciamento, riscos e etc.

Além dos fatores básicos supracitados, a abordagem proposta e descrita neste capítulo ainda considera diversos aspectos particulares presentes no cotidiano de projetos de software. A próxima seção detalha e discute cada um desses aspectos.

3.2 Aspectos Considerados

Esta seção discute, um a um, os principais aspectos considerados na formulação do problema.

3.2.1 Tarefas

Um projeto de software naturalmente é dividido em atividades de engenharia de software. A formulação proposta considera como tarefas quaisquer atividades que precisem ser executadas por recursos humanos para completude do projeto. Seus atributos serão detalhados na seção 3.3, na qual a formulação do problema é apresentada.

3.2.2 Recursos Humanos

É sabido que o desenvolvimento de software é uma atividade essencialmente intelectual e social, conforme afirmam Sommerville e Rodden (1996). Assim, pois, é preciso que projetos de software sejam centrados nos recursos humanos, uma vez que esses constituem os recursos que maior impacto causam sobre o processo e o produto de software, seja positiva ou negativamente. Naturalmente, outros tipos de recursos também fazem parte do processo de desenvolvimento, como máquinas, artefatos, ferramentas e etc. Porém, devido à supracitada importância dos recursos humanos, além da grande complexidade em geri-los, a formulação proposta somente considera recursos do tipo humano, supondo que os demais tipos podem mais facilmente ser geridos sem necessidade de ferramentas mais avançadas.

Recursos humanos possuem particularidades que precisam ser observadas. Nesse trabalho são considerados, por exemplo, habilidades e experiências individuais, disponibilidade e tipo de trabalho, que pode ser empregatício ou por prestação de serviço (consultoria). A diferença entre eles é que o empregado possui salário e esforço fixos, enquanto o consultor é pago por período de tempo trabalhado. Da mesma forma que para as tarefas, os atributos dos recursos são melhor detalhados na seção 3.3.

3.2.3 Interdependências Entre Tarefas

Uma aspecto básico presente em praticamente todo projeto de software é a interdependência entre tarefas, descrita pelo PMBoK (PMI, 2008) como "sequenciamento das tarefas" e comumente ilustrada através de diagramas de rede. Visando se adequar ao padrão já estabelecido, a abordagem proposta considera os quatro conceitos de relacionamento presentes no maioria das ferramentas de gerência de projetos: término a início, início a início, início a término e término a término. Como os nomes sugerem, a definição de cada uma dessas relações é dada pela restrição de se terminar ou iniciar uma tarefa para que outra tarefa possa ser iniciada ou terminada.

3.2.4 Habilidades e Experiência Individuais

Entre as principais especificidades dos recursos humanos estão as habilidades e experiência individuais. O manual do método de estimação COCOMO-II (Boehm, 2000) afirma que "depois do tamanho do produto, os fatores pessoais têm a maior influência na determinação da quantidade de esforço necessário para desenvolver um produto de software". O manual descreve seis fatores pessoais, dos quais cinco são relativos a habilidade e experiência[1]. Reconhecida sua importância, a abordagem proposta considera tais fatores conforme descrito em 3.2.4.1 e 3.2.4.2.

3.2.4.1 Habilidades

As habilidades, ou *skills*, são quaisquer requisitos pessoais que uma tarefa possa precisar para ser executada. Cada tarefa possui um conjunto de requisitos (habilidades) requeridos, enquanto cada recurso possui suas habilidades, incluindo seu nível de proficiência em cada uma.

Visando a generalidade da abordagem, a definição das habilidades, incluindo seu nível de especificidade, pode variar de acordo com o projeto, a necessidade ou o desejo do gerente. Um nível mais detalhado poderia incluir habilidades como "programação em Java para web" ou ainda o conhecimento em alguma tecnologia específica, como um *framework*. Em um nível mais abstrato, poderiam ser considerados simplesmente papéis, como programador ou analista. Também podem ser utilizados outros conceitos tecnológicos, como um banco de dados, um sistema operacional ou uma ferramenta, além de atributos psicológicos como liderança e trabalho em equipe.

As habilidades ainda são divididas por grupos de acordo com seu tipo, como, por exemplo: análise, desenho, programação, teste, tecnologia, suporte ou psicológico. Essa divisão por tipo é útil para diferenciar os fatores de ajuste, que serão discutidos na subseção 3.2.5, além de enriquecer um eventual modelo de aprendizado em treinamentos, brevemente discutido como possível trabalho futuro na seção 6.1.

3.2.4.2 Experiência

O conceito de experiência pode ser aplicado em diferentes contextos: experiência em ferramentas, em linguagens, na tarefa, no negócio e etc. A maior parte desses, como ferramentas, linguagens e plataformas, podem ser considerados como habilidades, conforme descrito na subseção anterior. Tendo isso em vista, o autor considera que o mais importante conceito de experiência a ser utilizado é a experiência na aplicação, conceito também utilizado no método COCOMO. Mais especificamente, a formulação

[1] O sexto fator é relativo ao histórico de continuidade de pessoal.

propõe a experiência como um atributo dos recursos relativo a cada tarefa. Ou seja, cada recurso possui algum ou nenhum grau de experiência em cada uma das tarefas a serem executadas.

Ambas habilidades e experiência são consideradas no cálculo da produtividade da equipe e consequente na estimação de tempo para realização das tarefas, conforme discutido na próxima subseção.

3.2.5 Estimação de Esforços e a Produtividade das Equipes

É natural em qualquer projeto de software que a estimação de esforços seja realizada antes da alocação dos recursos. Assim, a abordagem considera que o esforço requerido para execução de cada tarefa é um dado conhecido. Várias técnicas podem ser utilizadas para isso, como a análise de pontos de função (Symons, 1988) e/ou de linhas de código combinadas com modelos tais quais o método COCOMO-II ou algum dos citados na subseção 2.1.2. Dados históricos da organização também costumam ser bem utilizados nesse sentido.

O método COCOMO utiliza os fatores pessoais, supramencionados na subseção anterior, como fatores de ajustes denominados multiplicadores de esforço, os quais ajustam a estimação de esforços de acordo com parâmetros previamente conhecidos. Como propõe o referido método, as habilidades e experiências individuais ou da equipe podem ser levadas em consideração no momento da estimação de esforços, principalmente quando a estimação é relativa ao projeto como um todo. Por outro lado, quando a estimação é realizada para cada tarefa, ela depende diretamente da produtividade da equipe que a executa. Por esse motivo e uma vez que no ato da estimação as equipes ainda não estão alocadas às tarefas, esse trabalho propõe uma abordagem que considera os fatores pessoais de habilidade e experiência no momento da alocação.

Dado o esforço necessário para execução de determinada tarefa, estimado utilizando um método de estimação qualquer, a produtividade da equipe é calculada em cima de diversas variáveis. Entre elas, os níveis de habilidade e experiência da equipe, que são utilizadas de maneira semelhante ao COCOMO, como fatores de ajuste. Baseados em seu manual (Boehm, 2000), seus valores variam de “muito baixo” a “muito alto”. Eles são detalhados na seção 3.3, que também apresenta matematicamente o cálculo da produtividade das equipes.

3.2.6 Prazos e Custos

Qualquer que seja o projeto, é natural que ele seja sujeito a restrições de custo e rigidez de prazos. Além de permitir a limitação de custo e um *deadline* para término

do projeto, o trabalho considera ambos os fatores como objetivos a serem minimizados ao se realizar a alocação de equipes e o desenvolvimento do cronograma. Em outras palavras, a abordagem proposta busca soluções que minimizem tanto o custo quanto a duração do projeto. Também em relação a prazos, considera-se ainda o atraso nas tarefas, quando sujeitas a *deadlines*, como outro objetivo a ser minimizado. É importante ressaltar que é o uso de uma técnica multiobjetivo que permite à abordagem trabalhar simultaneamente com vários objetivos, tais quais os três citados.

3.2.7 Horas extras

Outra particularidade dos recursos humanos é a possibilidade de realização de horas extras em relação ao seu esforço de trabalho diário. Embora o trabalho além do tempo regular não seja uma prática recomendável, há situações de cronograma em que o uso de horas extras mostra-se necessário, como para realização de cronogramas acelerados ou por escassez de recursos. Tendo isso em vista, a formulação proposta contempla também esse aspecto.

Como supracitado na subseção 3.2.2, a formulação divide os recursos em dois tipos: os de vínculo empregatício e os de prestação de serviços. Enquanto o primeiro possui uma carga horária diária regular, o segundo varia sua carga horária de acordo com o tempo trabalhado, respeitando uma carga máxima diária. Assim, essa divisão considera a alocação de horas extras uma exclusividade dos recursos do tipo empregado, que ocorre quando o recurso tem alocado para si um esforço além de sua carga regular. Ressalta-se que todo recurso tem um limite de horas extras que deve ser respeitado.

Como supracitado, a prática de horas extras não é recomendável. Estudos relacionam a prática de horas extras com queda de produtividade, queda da qualidade do produto e retrabalho (Nishikitani et al., 2005; Li et al., 2000). Uma vez muito escassos modelos que quantifiquem tais fatores, a formulação não contempla esse aspecto no cálculo da produtividade das equipes. Porém, essa questão não é ignorada na abordagem, sendo tratada como mais uma função objetivo a ser minimizada.

3.2.8 Curvas de Aprendizado

Curvas de aprendizado, que são baseadas no tradicional conceito de que a prática leva à perfeição, são utilizadas para se determinar quantitativamente o ganho de produtividade devido à experiência adquirida durante um processo de desenvolvimento. O desenvolvimento de software envolve diversas questões afetadas pelo aprendizado, tais como habilidades, conhecimento da aplicação, uso de ferramentas e aplicação de metodologias, o que sugere um cenário propício a este paradigma. Infelizmente, segundo

Zorgios et al. (2009), embora evidências em vários ambientes de produção indiquem que modelos de aprendizado são capazes de refletir as melhorias de produtividade observadas, abordagens baseadas no chamado capital intelectual (como o desenvolvimento de software) têm dado pouca importância aos benefícios que o uso de tal paradigma pode trazer. De fato, percebe-se na literatura uma escassez de trabalhos de software que se atentem à questão.

Visando superar esse fato e aproveitar-se dos supracitados benefícios, a abordagem proposta incorpora um modelo de aprendizado no cálculo de produtividade das equipes. É utilizado um tradicional modelo log-linear de custo por unidade, o qual estipula que o esforço de produção diminui uma dada porcentagem a cada vez que a produção é dobrada. Esse modelo é expresso na equação (3.1):

$$C_x = C_1 \times x^b \tag{3.1}$$

Na equação acima, C_x é o custo de implementação da x-ésima unidade, C_1 é o custo de implementação da primeira unidade e b é um expoente constante. A equação (3.2) mostra como calcular b em função da razão r de produtividade entre uma unidade $2x$ e uma unidade x.

$$r = \frac{C_{2x}}{C_x} = \frac{C_1 \times (2x)^b}{C_1 \times (x)^b} = 2^b \;\Rightarrow\; b = log_2(r) \tag{3.2}$$

Para exemplificar as equações acima, consideremos a seguinte situação: uma dada unidade qualquer é produzida com esforço 100 de homens–hora. Considerando um aprendizado que reduza esse esforço para 90% a cada vez que se dobre o número de vezes que a unidade foi produzida, o esforço estimado para produzir a décima unidade é calculado da seguinte maneira:

$$\begin{aligned} \text{utilizando (3.1):} \quad & C_{10} = 100 \times 10^b \\ \text{e (3.2):} \quad & b = log_2(0.9) = -0.15 \\ \text{tem-se que:} \quad & C_{10} = 70.795 \text{ homens–hora} \end{aligned}$$

Para calcular o custo acumulado de se produzir x unidades, soma-se os valores de C_1 a C_x, ou seja, $\sum_{i=1}^{x} C_i$. Para processos contínuos, esse cálculo é realizado com o uso de integral, conforme a equação (3.3):

$$C_{acc} = \int_0^x (C_z)\, dz = \int_0^x (C_1 \times z^b)\, dz = \frac{C_1 \times x^{(b+1)}}{b+1} \tag{3.3}$$

Usando (3.3), calcula-se o custo médio por unidade dividindo o esforço acumulado

pela quantidade de unidades, conforme a equação (3.4):

$$C_{med} = \frac{\frac{C_1 \times x^{(b+1)}}{b+1}}{x} = \frac{C_1 \times x^b}{b+1} \tag{3.4}$$

A equação (3.1) descreve como calcular o custo de produção da x-ésima unidade, enquanto a equação (3.3) expressa o custo acumulado para produção de x unidades e a equação (3.4) o custo médio. Para expressar tempo de produção em vez de custo, pode-se substituir o conceito de custo pelo conceito de tempo. Porém, diferentemente de processos de produção industrial, o desenvolvimento de software não se baseia em produções de unidades, mas sim no acumulo de esforço intelectual dos desenvolvedores, de forma que se deseja expressar não o custo ou tempo de produção de unidades, mas sim a produtividade de equipes em função de seu aprendizado. Para isso, esta abordagem utiliza o conceito de produtividade no lugar de custo e, consequentemente, o conceito de intervalos de tempo no lugar de unidades produzidas. Dessa forma, a equação (3.4) pode ser traduzida para expressar a produtividade média de uma equipe conforme a equação (3.5):

$$P_{med} = \frac{P_1 \times x^b}{b+1} \tag{3.5}$$

Na equação (3.5), P_{med} representa a produtividade média da equipe durante o período de execução da tarefa, P_1 representa a produtividade no primeiro instante de tempo, isto é, quando ainda não existe ganho aprendizado, e x representa a quantidade de instantes de tempo para execução da tarefa, ou seja, sua duração. Além disso, o cálculo do expoente b passa a ser feito em função do inverso da razão r, uma vez que ele agora representa o crescimento da produtividade e não mais o decréscimo do custo. Assim, utilizando a equação (3.2) e a propriedade logarítmica $log(x) = -log(1/x)$, temos que $b = -log_2(r)$.

Dado que a produtividade é igual ao esforço dividido pelo tempo, tem-se que P_1 e P_{med} podem ser calculados, respectivamente, em função da duração d e da duração ajustada pelo aprendizado d_{learn}, conforme expresso nas equações abaixo:

$$P_1 = \frac{effort}{d} \tag{3.6}$$

$$P_{med} = \frac{effort}{d_{learn}} \tag{3.7}$$

Substituindo P_1 e P_{med} da equação (3.5) por (3.6) e (3.7) e uma vez que x é igual à duração ajustada d_{learn}, tem-se a equação final para cálculo da duração em função do

aprendizado, descrita abaixo:

$$\frac{effort}{d_{learn}} = \frac{\frac{effort}{d} \times x^b}{b+1} \Rightarrow d_{learn} = \sqrt[b+1]{d \times (b+1)} \tag{3.8}$$

Utilizando a equação (3.8), a figura 3.3 ilustra um gráfico da duração da tarefa de um projeto em função do tempo. Como esperado, o gráfico apresenta um comportamento logarítmico para o ganho de produtividade obtido durante a execução da tarefa. O grau de curvatura da curva depende da razão de aprendizado utilizada. Uma razão bem escolhida é aquela que traduz com o máximo de exatidão a porcentagem de produtividade ganha a cada vez que se dobra a duração da tarefa. Naturalmente, para uma razão zero, isto é, sem aprendizado algum, a curva assume um comportamento linear.

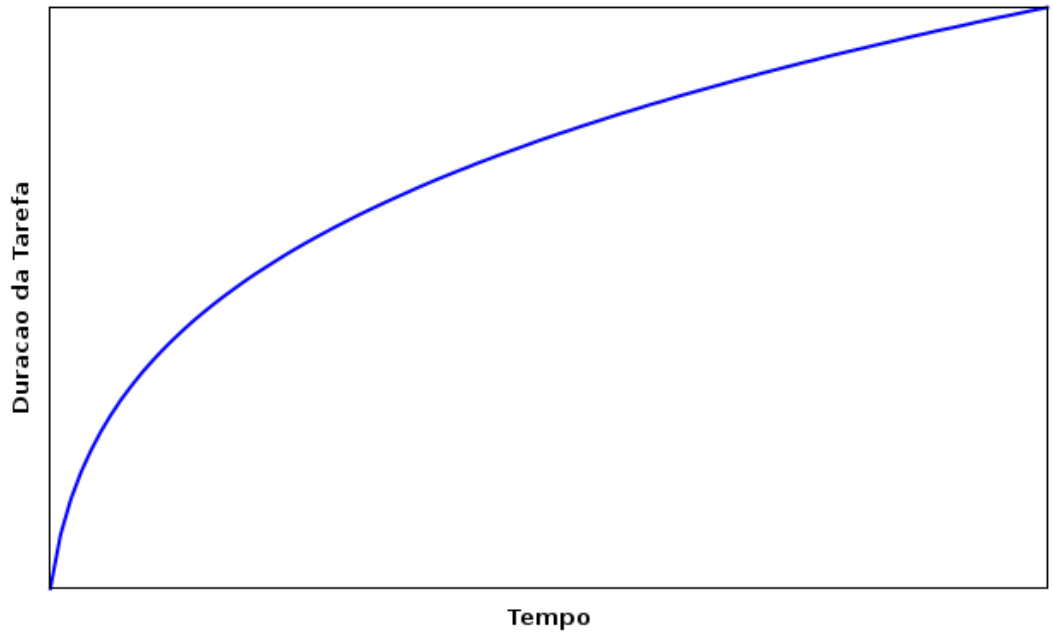

Figura 3.3. Aprendizado: tempo × duração das tarefas

Vale ressaltar que, além do modelo log-linear utilizado, a abordagem permite o uso de quaisquer outros modelos para modelagem de aprendizado. Dois exemplos são os modelo Stanford-B e DeJong, nos quais o primeiro assume que o aprendizado demora um certo tempo para se manifestar e o segundo considera um fator de incompressibilidade relativo aos aspectos do processo que não são sujeitos ao aprendizado, como o gasto de tempo relativo ao ambiente de trabalho, por exemplo. Entre outros, há ainda o modelo S-Curve, que combina os modelos Stanford-B e DeJong.

Mais discussões sobre curvas de aprendizado podem ser encontradas. Badiru (1992)

apresenta e avalia diversos modelos de aprendizado, Kerzner (2009) discute mais profundamente sobre curvas de aprendizado no âmbito da gerência de projetos em geral, enquanto Raccoon (1996) e Zorgios et al. (2009) as relacionam especificamente com o desenvolvimento de software.

3.2.9 Overhead de Comunicação

Conhecido princípio da engenharia de software, a lei de Brooks (1995) diz que "acrescentar mão de obra a um projeto de software atrasado torna-o ainda mais atrasado". Brooks afirma que métricas como homens-hora, homens-dias ou homens-meses, constantemente utilizadas para quantificar o esforço empregado em tarefas de software, são equivocadas devido ao *overhead* de comunicação interpessoal. Em outras palavras, a não ser que a tarefa possa ser particionada entre desenvolvedores sem que seja necessário qualquer tipo de comunicação entre eles, o que geralmente não ocorre na prática, é preciso considerar o tempo gasto na comunicação entre aqueles que a desenvolvem.

Brooks identifica dois tipos de comunicação ao se dividir uma tarefa. A primeira ocorre quando a tarefa pode ser subdividida sem que haja necessidade de comunicação entre as subtarefas. Nesse caso, o overhead se resume ao tempo gasto para integração das partes. O outro caso é a situação contrária, quando existe a necessidade de comunicação entre os membros da equipe. Nesse caso, numa equipe com n desenvolvedores existem $\frac{n\times(n-1)}{2}$ canais de comunicação pessoa a pessoa. Para incorporar mais este conceito na modelagem da produtividade das equipes, o trabalho utiliza o segundo caso, pois o autor considera que a necessidade de comunicação entre os desenvolvedores é o que mais comumente ocorre em projetos software. Em especial, esse pensamento se justifica para a abordagem proposta porque nela a alocação das equipes é um fator variável e, por consequência, não existe uma subdivisão de tarefas bem definida.

Conforme supracitado, numa equipe com n desenvolvedores existem $\frac{n\times(n-1)}{2}$ canais de comunicação. Esta expressão é utilizada para calcular a duração ajustada d_{comm} sobre a duração d em função do número n de integrantes da equipe e de um certo coeficiente de comunicação cf, que indica a proporção média de tempo gasto por um indivíduo na comunicação para com cada membro de sua equipe. Esta relação é matematicamente expressa na equação (3.9):

$$d_{comm} = d \times \left(1 + cf \times \frac{n \times (n-1)}{2}\right) \tag{3.9}$$

A equação acima representa o aumento no tempo, isto é, a perda de produtividade, devido ao *overhead* de comunicação interpessoal. O gráfico apresentado na figura 3.4 ilustra a curva do tempo de desenvolvimento em função do tamanho da equipe. Nele,

percebe-se que até um certo ponto o número de desenvolvedores faz com que decresça o tempo de desenvolvimento. Porém, o tempo tende a aumentar após um certo valor devido à necessidade de comunicação interpessoal. O coeficiente de comunicação usado é determinante para o ponto de inflexão da curva. Experimentos apresentados no capítulo 5 estudam e discutem esse comportamento com mais detalhes.

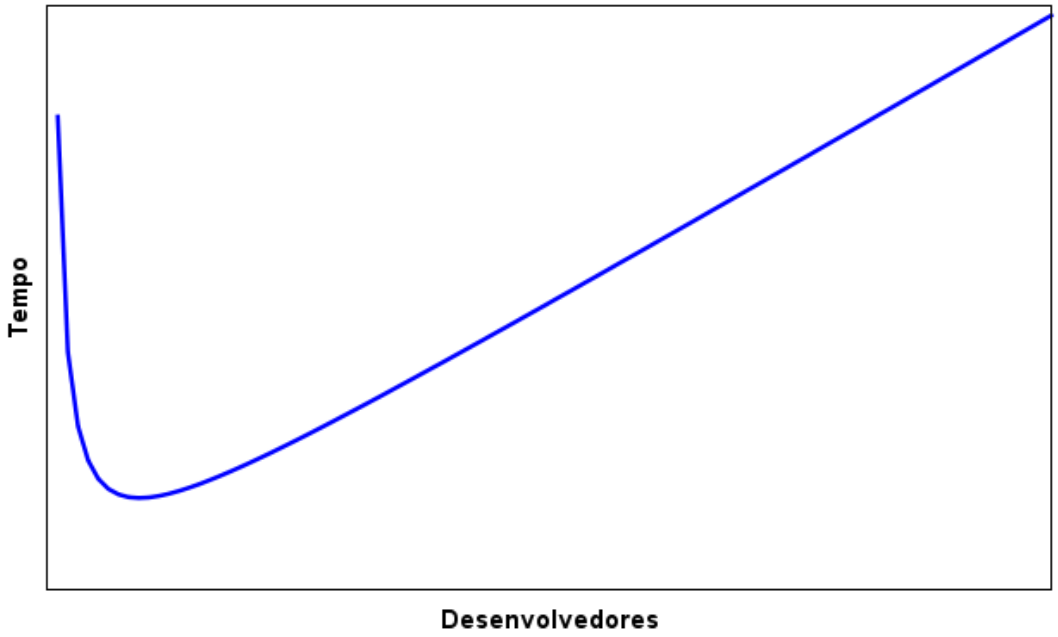

Figura 3.4. Overhead de comunicação: tempo × número de desenvolvedores

Embora a lei de Brooks seja largamente aceita pela comunidade, poucos foram os que tentaram quantificá-la ou modelá-la. Uma exceção é o trabalho de Di Penta et al. (2007), que utiliza SBSE para estudar diferentes formulações para o *overhead* de comunicação. Além do modelo quadrático acima discutido, o trabalho considera um modelo polinomial, mais complexo, que considera comunicação em subgrupos, e modelos linear e logarítmico, mais simples, que representam casos onde uma rede de comunicação é previamente estabelecida pelo gerente em vista de evitar o crescimento polinomial da comunicação. Embora esse trabalho utilize em sua formulação o modelo quadrático conforme descrito por Brooks (1995), vale salientar que qualquer outro modelo poderia ser utilizado em seu lugar, como algum dos utilizados por Di Penta et al. (2007).

3.2.10 Participação do Gerente

Embora apresente uma abordagem que vise automatizar a tarefa de alocação de equipe e desenvolvimento de cronograma, o autor reconhece a importância do julgamento

humano, que provavelmente nunca será completamente suprido por uma máquina. Dessa forma, a abordagem é aberta ao julgamento humano em vários sentidos, conforme discutido nos próximos parágrafos.

Primeiramente, o uso de uma técnica multiobjetivo oferece ao gerente de projetos uma ampla gama de soluções ao problema, cada qual com suas vantagens e desvantagens em relação a cada objetivo considerado. Assim, o gerente tem a possibilidade de analisar e discernir acerca de cada uma, a fim de escolher qual delas mais se enquadra em seu pensamento, podendo ainda modificá-la conforme seu pensamento e/ou necessidade ou mesmo combiná-la com outras soluções. Outra maneira de incorporar as intuição e experiência humanas na abordagem se dá com a possibilidade do uso de soluções previamente obtidas pelo especialista como soluções iniciais do algoritmo. Isso permite que o algoritmo seja guiado a partir de tais soluções em busca de aprimorá-las e de oferecer novas soluções, que podem apresentar perspectivas diferentes não atentadas inicialmente. Todo o algoritmo, incluindo o uso de soluções iniciais predefinidas, é discutido no capítulo 4.

O uso de um modelo matemático para formular o problema permite ainda a sua extensão, como a definição de restrições ao planejamento e a incorporação de novos objetivos. A extensibilidade da formulação é discutida na subseção 3.6.2. Finalmente, mas não menos importante, vale ressaltar que é o gerente o responsável por definir os parâmetros da abordagem, como os recursos e as tarefas, além de seus atributos. Dessa forma, é dada a liberdade necessária para que o problema seja tratado conforme sua visão e necessidade.

Conforme mostrado nas questões supracitadas, a abordagem proposta não elimina a indispensável presença do gerente, mas faz com que ele assuma um diferente papel. Nesse novo cenário, o gerente passa a ser o responsável por guiar a resolução da tarefa em vez de executá-la. Essa nova perspectiva elimina as desvantagens de se basear uma tarefa de tamanha importância exclusivamente no *feeling* de um ser humano, que, por mais competente que seja, é naturalmente sujeito a erros.

3.2.11 Planejamento multiprojetos

Embora organizações de software geralmente trabalhem em ambientes multiprojetos, são escassos os estudos relacionadas ao gerenciamento de múltiplos projetos de software, e é pequena a participação de comunidade desenvolvedora de software em pesquisa acerca de multiprojetos, como apontam Dong et al. (2008)[2]. Turner (2009) afirma

[2]Um levantamento na base de dados do IEEE apontou que a parcipação da comunidade de software em trabalhos sobre o tema é de 20% do total. No mesmo levantamento, restringindo-se o universo de pesquisa ao periódico *Internation Journal of Project Management* (IJPM), importante referência ao

em seu livro que, embora a literatura relacionada seja voltada a projetos isolados, essa situação o ocorre em menos de 10% casos. Segundo ele, a grande maioria dos projetos acontece como parte de um programa ou de um portfólio de projetos, os quais constituem os dois tipos comuns de associação entre múltiplos projetos. Um programa de projetos é descrito por Turner como "um grupo de projetos que contribuem para um objetivo comum de ordem superior", enquanto portfólios de projetos são "conjuntos de projetos que compartilham recursos comuns".

Turner defende haver pouca diferença entre programa e projeto, por definição. Um programa pode ser visto como um grande projeto que, por exemplo, envolve diferentes áreas de uma organização. Em geral, a principal diferença entre eles é a dimensão (monetária, temporal, etc), que num programa tende a ter maiores proporções. Por outro lado, portfólios de projetos apresentam maiores diferenças com projetos individuais, como: priorização de objetivos, balanceamento dos recursos e interface entre projetos.

Em suma, programas e portfólios são uma realidade nas organizações e, portanto, precisam de atenção. O mesmo Turner apresenta, em linhas gerais, três maiores diferenças entre multiprojetos e projetos individuais:

- Projetos em paralelo apresentam objetivos mutuamente independentes, onde um benefício maior é obtido com a interseção destes;
- O compartilhamento de recursos em situações de multiprojetos; e
- As interdependências entre os vários projetos.

Este trabalho considera as três questões em sua abordagem. Uma vez que ela se baseia em uma formulação multiobjetivo para o problema, os objetivos de cada projeto podem ser separadamente considerados, de forma o algoritmo se encarrega de apresentar soluções que os balanceiem de diferentes maneiras. Em relação ao compartilhamento de recursos, basta que se considere o conjunto de tarefas como a união das tarefas dos vários projetos, em vez de tarefas de um só projeto. Assim, o algoritmo se encarrega de realizar a alocação compartilhando os recursos entre os projetos. Por fim, a interdependência de projetos pode ser formulada da mesma maneira que a interdependência entre tarefas, com a diferença que as tarefas podem fazer parte de projetos diferentes.

O capítulo 5 apresenta testes que mostram a viabilidade do uso da abordagem para o planejamento multiprojetos. Assim, embora o trabalho utilize constantemente o termo "projeto", deve estar subentendido que o mesmo vale para planejamentos multiprojetos. Mais informações sobre alocação de recursos e definição de cronogramas

gerenciamento de projetos, esse percentual cai para 4%.

para múltiplos projetos de software, incluindo um *survey* da literatura relacionada, podem ser encontradas no trabalho de Dong et al. (2008).

3.2.12 Replanejamentos

Infelizmente, o planejamento inicial dificilmente pode ser seguido sem modificações até o término do projeto. Projetos de software, principalmente, sofrem com constantes transformações, tais como mudanças nos requisitos e retrabalho. O monitoramento e controle de projetos é uma importante tarefa da gerência, a qual, entre outros, é responsável por determinar replanejamentos corretivos ou preventivos quando necessário.

Uma vez que a abordagem proposta é direcionada ao planejamento de projetos, replanejamentos podem ser realizados sempre que preciso, ajustando-se as entradas (tarefas e recursos) para que um novo plano seja obtido conforme o necessário. Uma abordagem mais completa envolveria um acompanhamento dinâmico do projeto, onde, durante o seu decorrer, informações sobre o seu andamento seriam atualizadas de forma que o cronograma fosse dinamicamente refeito. A abordagem proposta poderia ser expandida nesse sentindo, porém seu escopo atual é o planejamento, de forma que o monitoramento e controle dinâmico não são considerados.

3.3 Formulação do Problema

Considerando os aspectos discutidos na seção anterior, uma formulação para o problema foi criada conforme apresentado nesta seção. São considerados os seguintes conjuntos:

- **Tarefas**: $T = \{t_1, t_2, \ldots, t_{|T|}\}$ é o conjunto de tarefas a serem executadas. Cada tarefa tem os seguintes atributos: nível de importância, esforço estimado e, se existirem, datas de início e término (*deadline*).

- **Recursos**: $R = \{r_1, r_2, \ldots, r_{|R|}\}$ são os recursos que executam as tarefas. Conforme explicado anteriormente, existem dois tipos de recursos: empregado e contratado. O primeiro, que possui salário e dedicação diária fixos e que pode realizar horas extras, possui os seguintes atributos: salário, dedicação diária, custo de hora extra e tempo máximo de hora extra. O segundo, que recebe por hora trabalhada, possui como atributos: custo da hora trabalhada e máxima dedicação diária. Além desses atributos, ambos os tipos possuem o atributo disponibilidade, que representa o calendário de cada recurso, ou seja, em que períodos estão disponíveis.

- **Skills**: $S = \{s_1, s_2, \ldots, s_{|S|}\}$ representam as habilidades consideradas, conforme discutido na subseção 3.2.4. Também como já discutido, cada habilidade, ou *skill*, pode ser de um tipo específico, tal como de análise, de programação, de suporte, psicológico e etc.

Especificados os "atores" da formulação, as próximas subseções discutem como eles se relacionam entre si e com os demais aspectos já discutidos.

3.3.1 Tarefas × Skills

Cada tarefa possui requisitos pessoais para ser executada. Por exemplo, implementar um dado caso de uso requerer conhecimento em uma linguagem de programação específica, enquanto a análise de requisitos requer a habilidade de análise. Conforme já discutido, o grau de especificidade dos *skills* é uma decisão do gerente.

3.3.2 Recursos × Skills

Uma vez que cada tarefa possui uma lista de *skills* como requisitos à sua execução, naturalmente a equipe alocada a uma tarefa deve atender a esses requisitos. Para isso, da mesma forma que para as tarefas, cada recurso possui uma lista de *skills* que representam as habilidades que o recurso possui. Além disso, essa lista contém também o nível de proficiência de cada recurso em relação a cada *skills*. A proficiência não é um pré-requisito para a execução da tarefa, mas é utilizada para o cálculo da produtividade da equipe na sua execução.

3.3.3 Recursos × Tarefas

O relacionamento entre recursos e *skills* está para as habilidades individuais assim como o relacionamento entre recursos e tarefas está para o conceito de experiência. Em outras palavras, cada recurso possui um conjunto de tarefas e um nível de experiência em cada. Da mesma forma que ocorre com o nível de proficiência nos *skills*, o nível de experiência nas tarefas também é um dado que influencia na produtividade da equipe.

3.3.4 Atribuição de Tarefas e Formação das Equipes

Uma equipe é a união de todos os recursos alocados a uma tarefa qualquer. Em outras palavras, é a atribuição dos recursos às tarefas que define a formação das equipes.

Cada recurso é alocado às tarefa em valores, representados em porcentagem, que variam de zero ao máximo permitido para o recurso. Obviamente, 0% significa que

o recurso não está alocado à tarefa, enquanto demais valores indicam a proporção de esforço, dentro de sua dedicação diária regular, que o recurso deverá empregar na execução da tarefa. Para evitar atribuições de valores irrealísticos, a atribuição é feita com valores discretos, que diferem 25% entre si, ou seja, 0%, 25%, 75%, 100%, 125% e etc. Valores acima de 100% indicam a atribuição de horas extras.

Para exemplificar, suponhamos que um empregado cuja dedicação regular seja de 8 horas diárias tenha sido alocado 50% a uma tarefa "A" e 75% a outra tarefa "B". Nesse caso, o empregado deverá empregar 4 horas de seu trabalho diário à primeira e 6 horas à segunda. Além disso, considerando que as tarefas sejam executadas simultaneamente, o empregado estará realizando 25% de hora extra, ou seja, 2 horas por dia.

3.3.5 Produtividade

Até aqui, muito foi discutido sobre as influências de diversos fatores sobre a produtividade das equipes e, consequentemente, sobre o cálculo da duração de cada tarefa. Finalmente, as equações (3.10) e (3.11) expressam matematicamente tudo o que foi levantado. As variáveis utilizadas nas equações são descritas na tabela 3.1.

$$t.duration = \frac{t.effort}{\sum_{r \in R} prod_{r,t}} \times learning \times communication \quad , \forall\, t \in T \tag{3.10}$$

$$prod_{r,t} = x_{r,t} \times r.effort \times \left(\prod_{s \in (S^r \cap S^t)} r.proficiency(s) \right) \times r.experience(t) \tag{3.11}$$

As tabelas 3.2 e 3.3 apresentam, respectivamente, os valores utilizados como fatores de ajustes em relação às habilidades e à experiência. Os valores utilizados foram baseados nos valores dos multiplicadores de esforço contidos no manual do método COCOMO-II (Boehm, 2000). Também com base no 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*, que inclui os tipos relativos a análise, desenho e atributos psicológicos, como comunicação e cooperação; *implementação*, que compreende habilidades de desenvolvimento e programação; e *outros*, que são relativos aos demais tipos, como habilidades em bancos de dados, sistemas operacionais, redes e etc.

[3] A abordagem não considera a falta de experiência como uma restrição à execução de tarefas, porém é considerado um fator de ajuste consideravelmente baixo de forma a evitar esse tipo de alocação. Para se definir a experiência como uma restrição, bastaria modificar para 0 o fator de *extra*

Variável	Descrição
$x_{r,t}$	Variáveis de decisão do problema, que expressam a proporção de esforço do recurso r na tarefa t, conforme discutido na subseção 3.3.4.
$r.effort$	Dedicação diária do recurso r, em horas.
S^r	Conjunto de *skills* que o recurso r possui, ou seja, suas habilidades.
S^t	Conjunto de *skills* requeridos pela tarefa t para sua execução.
$r.proficiency(s)$	Fator de ajuste devido à proficiência do recurso r no *skill* s. Seus valores são apresentados na tabela 3.2.
$r.experience(t)$	Fator de ajuste devido à experiência do recurso r na tarefa t. Seus valores são apresentados na tabela 3.3.
$learning$	Multiplicador para ajuste da duração devido ao aprendizado, conforme descrito na equação (3.8).
$communication$	Multiplicador para ajuste da duração devido ao *overhead* de comunicação, conforme descrito na equação (3.9).
$t.duration$	É o valor que se deseja calcular. Representa o tempo total para execução da tarefa t, em dias.

Tabela 3.1. Descrição das variáveis das equações (3.10) e (3.11)

	very low	*low*	*normal*	*high*	*very high*
Descrição	15º percentil	35º perc.	55º perc.	75º perc.	90º perc.
Análise	0.70	0.84	1.0	1.18	1.41
Implementação	0.75	0.87	1.0	1.14	1.32
Outros	0.84	0.92	1.0	1.10	1.18

Tabela 3.2. Fatores de ajuste de proficiência

	extra low[3]	*very low*	*low*	*normal*	*high*	*very high*
Descrição	$= 0$	$\leq$ 2 meses	$\leq$ 6 meses	$\leq$ 1 ano	$\leq$ 3 anos	$>$ 3 anos
Valor	0.63	0.82	0.91	1.0	1.14	1.23

Tabela 3.3. Fatores de ajuste de experiência

3.3.6 Cronograma

Uma vez calculado o tempo estimado para execução de cada tarefa, o cronograma do projeto pode ser definido. Utilizando a duração $t.duration$ de cada tarefa e considerando as interdependências entre tarefas e eventuais restrições do tipo "não iniciar antes de", o início $t.start$ e término $t.finish$ de cada tarefa pode ser calculado. Além disso, tarefas que não fazem parte do caminho crítico podem ter seu início reajustadas pelo algoritmo de forma a se evitar superalocações e obter uma melhor distribuição de esforços.

3.3.7 Custos

Uma vez que os recursos humanos geram custos, é desejável que a alocação de equipes seja realizada de forma a minimizá-los. Como já discutido, a abordagem diferencia os recursos em dois tipos: empregados e contratados. Em relação ao primeiro, além de um salário fixo, existe a possibilidade de realização de horas extras, o que acarreta despesas adicionais. Já para o segundo, cada hora trabalhada gera um custo.

3.3.8 Formulação Geral

Diante das questões supradescritas, o problema proposto pode ser resumido com da seguinte maneira:

É desejado:

- Minimizar o tempo total do projeto;
- Minimizar o custo total;
- Minimizar o atraso nas tarefas; e
- Minimizar as horas extras.

Respeitando:

- As habilidades requeridas por cada tarefa;
- O tempo máximo de trabalho de cada recurso;
- As interdependência entre tarefas;
- O cronograma de cada recurso, ou seja, sua disponibilidade;
- O prazo para término do projeto, se existir; e
- Limite de custos, se existir.

low, assim tornando nula a produtividade dos recursos sem experiência na tarefa.

3.4 Modelo Matemático

Com objetivo de formalizar o problema, um modelo matemático foi proposto. O modelo expressa o problema de alocação de equipes e desenvolvimento de cronogramas considerando de forma genérica todas as questões supracitadas na seção anterior.

3.4.1 Dados e Variáveis

Seja:

- $I = \{1, \ldots, i, \ldots, |I|\}$ – conjunto de tarefas a serem executadas;
- $J = \{1, \ldots, j, \ldots, |J|\}$ – conjunto de equipes de recursos humanos;
- $T = \{1, \ldots, t, \ldots, |T|\}$ – conjunto de períodos (espaços de tempo);
- $J^i = \{j_1, j_2, \ldots\}$ – conjunto de equipes que podem executar a tarefa i;
- $D^i = \{i_1, i_2, \ldots\}$ – conjunto de dependências da tarefa i, isto é, tarefas que precisam ser executadas antes da tarefa i.

Dados:

- t_{ij} – tempo de execução da tarefa i pela equipe j, ou seja, sua duração;
- c_{ij} – custo de desenvolvimento da tarefa i pela equipe j;
- d_i – *deadline* de entrega da tarefa i;
- w_i – custo pelo atraso da tarefa i.

Variáveis:

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

- a_i – atraso na entrega da tarefa i;
- s_j – término das tarefas da equipe j;
- s – makespan (tempo de término de todas as tarefas).

3.4.2 Objetivos

Minimizar:

$$A = \sum_{i \in I} (w_i * a_i)$$

$$C = \sum_{i \in I} \sum_{j \in J^i} \sum_{t=1}^{|T|-t_{ij}} \left(c_{ij} * x_{ij}^t\right)$$

$$S = s$$

3.4.3 Restrições

Todas as tarefas têm de ser executadas:

$$\sum_{j \in J} \sum_{t=1}^{|T|-t_{ij}} \left(x_{ij}^{t}\right) = 1 \ , \ \forall\, i \in I$$

Uma equipe pode executar somente uma tarefa no tempo t:

$$\sum_{i \in I} \left(x_{ij}^{t}\right) \leq 1 \ , \ \forall\, j \in J \ ; \ \forall\, t \in T$$

Nenhuma outra tarefa pode ser alocada à equipe j enquanto esta estiver executando uma tarefa i:

$$x_{ij}^{t} + \sum_{k=t}^{t+t_{ij}-1} \left(x_{i^{'}j}^{k}\right) \leq 1 \ , \ \forall\, i, i^{'} \in I \ , \ i \neq i^{'} \ ; \ \forall\, j \in J \ ; \ \forall\, t \ \mid \ 0 \leq t \leq (|T| - t_{ij})$$

Atraso na execução da tarefa i:

$$a_i \geq \sum_{j \in J} \sum_{t=0}^{|T|-t_{ij}} \left((t + t_{ij}) * x_{ij}^{t}\right) - d_i \ , \ \forall\, i \in I$$

Término das tarefas executadas por j:

$$s_j \geq (t + t_{ij}) * x_{ij}^{t} \ , \ \forall\, i \in I \ ; \ \forall\, j \in J \ ; \ \forall\, t \ \mid \ 0 \leq t \leq (|T| - t_{ij})$$

Makespan (término de todas as tarefas):

$$s \geq s_j \ , \ \forall\, j \in J$$

Interdependências entre as tarefas (tarefa i' é pre-requisito para a tarefa i)

$$x_{ij}^{t} \leq x_{i^{'}j^{'}}^{t+t_{ij}} \ , \ \forall\, i \in I \ ; \ \forall\, i^{'} \in D^{i} \ ; \ \forall\, j \in J \ ; \ \forall\, j^{'} \in J^{i^{'}} \ ; \ \forall\, t \ \mid \ 0 \leq t \leq t_{ij}$$

Integralidade e não negatividade:

$$x_{ij}^{t} \in \{0, 1\} \ , \ a_i \geq 0 \, , \ s_i \geq 0 \, , \ s_j \geq 0 \ , \ \forall\, i \in I \ ; \ \forall\, j \in J \ ; \ \forall\, t \ \mid \ 0 \leq t \leq (|T| - t_{ij})$$

3.4.4 Considerações

Acima, estão listados três objetivos já discutidos anteriormente. São eles, respectivamente: minimização de atrasos, minimização de custos e minimização do tempo. Além disso, são descritas restrições que devem ser respeitadas pelas variáveis do problema de forma a se obter soluções válidas. Tais soluções são constituídas pelas variáveis de decisão x_{ij}^t.

Como anteriormente salientado, o modelo acima descrito considera os aspectos e as questões discutidos nas seções 3.2 e 3.3. Aqui, vale a pena discutir como o modelo se aplica a tais questões.

O conceito de equipes, representado pelo conjunto J, é utilizado de forma genérica. Equipes pré-definidas podem ser formadas por um ou mais recursos, cada qual com uma certa dedicação, lembrando que, conforme discutido em 3.3.4, a abordagem limitou as porções de dedicação a valores que variam 25% entre si. Por exemplo, uma equipe poderia ser formada por três recursos, dos quais o primeiro tem uma dedicação de 100%, o segundo de 75% e o terceiro de 50%. Diante dessa configuração prévia da equipe, é possível, conforme discutido na subseção 3.3.5, obter a produtividade da equipes para cada tarefa e, consequentemente, a duração desta, representada no modelo pelo dado t_{ij}. A partir do tempo de execução, e tendo em mãos os custos relativos à cada recurso, obtém-se o custo de execução da tarefa i pela equipe j, o qual é representado pelo dado c_{ij}.

Já a questão das habilidades necessárias para a execução das tarefas é tratada com o conjunto J^i, que representam as equipes j que podem executar a tarefa i. Assim, obviamente só devem fazer parte do conjunto J^i as equipes que satisfaçam as habilidades necessárias para execução daquela tarefa.

3.4.4.1 Horas Extras

Ao analisar o modelo proposto, o leitor pode questionar que, embora horas extras tenham sido um aspecto discutido na seção 3.2.7, o modelo apresentado não contempla essa questão. Isso deve-se simplesmente para efeito de simplificação e melhor entendimento, e, como será mostrado a seguir, o modelo pode ser estendido para o uso de horas extras. Para incorporar ao modelo esta questão, bastam as seguintes modificações:

1. Definir a variável he_j, que representa a quantidade de horas extras da equipe j.
2. Definir o novo dado c_j^{he}, que é o custo de hora extra da equipe j.
3. Substituir todo t_{ij} por $t_{ij} - he_j$, de forma que o tempo de execução fica diminuído pela quantidade de horas extras realizadas.

4. Incluir o custo de horas extras na função objetivo de custo, conforme a seguinte equação:

$$C = \left[\sum_{i \in I} \sum_{j \in J^i} \sum_{t=1}^{|T|-t_{ij}} \left(c_{ij} * x_{ij}^{t} \right) \right] + \left[\sum_{j \in J} \left(c_{j}^{he} * he_{j} \right) \right]$$

3.4.4.2 Treinamentos

Embora não tenha sido inteiramente considerada na abordagem a possibilidade de treinamento de pessoal (e por isso não discutida esta questão nas seções anteriores), diante da importância deste aspecto, a questão já foi considerada de antemão no modelo proposto[4]. De forma semelhante à questão de horas extras, algumas modificações no modelo apresentado são suficientes para isso:

1. Definir a variável ht_{ij}, que representa a redução no tempo de execução t_{ij} conforme treinamento para elevação da qualificação de j.

2. Definir a variável $y_{ij}^{q} \in \{0, 1\}$, representando a realização ou não realização do bloco de treinamento q para a equipe j na tarefa i.

3. Definir o dado $c^{ht^{q}}_{ij}$, que representa o custo do bloco de treinamento q para a equipe j na tarefa i.

4. Substituir todo t_{ij} por $t_{ij} - ht_{ij}$, de forma que o tempo de execução fica diminuído pela redução de tempo relativo ao aumento da qualificação de j.

5. Incluir uma restrição relativa à unicidade dos bloco de treinamento para cada combinação $i \times j$, conforme a seguinte equação:

$$\sum_{q=1}^{|Q|} y_{ij}^{q} \leq 1 \ , \ \forall\, i \in I \ ; \ \forall\, j \in J$$

6. Incluir uma restrição relativa à variável ht_{ij}^{q}, representando a redução no tempo de execução t_{ij} conforme o bloco de treinamento realizado:

$$ht_{ij} = ht_{ij}^{q} * y_{ij}^{q} \ , \ \forall\, i \in I \ ; \ \forall\, j \in J \ ; \ \forall\, q \ \mid \ 1 \leq q \leq |Q|$$

7. Incluir o custo de treinamento na função objetivo de custo, conforme a equação:

$$C = \left[\sum_{i \in I} \sum_{j \in J^i} \sum_{t=1}^{|T|-t_{ij}} \left(c_{ij} * x_{ij}^{t} \right) \right] + \left[\sum_{i \in I} \sum_{j \in J} \sum_{q=1}^{|Q|} \left(c^{ht^{q}}_{ij} * y_{ij}^{q} \right) \right]$$

[4] A seção 6.1, de trabalhos futuros, discute, entre outros, sobre a questão dos treinamentos

3.5 Análise de Complexidade

Inicialmente, será analisada a complexidade do espaço de soluções do problema, isto é, a quantidade de soluções possíveis em função dos parâmetros de entrada do problema. Para um projeto com $n = |I|$, $m = |J|$ e $p = |T|$, tem-se um número de variáveis ${x^t}_{ij}$ igual a nmp. Como cada variável pode assumir 2 valores, 0 ou 1, existem 2^{nmp} diferentes configurações, o que significa um espaço de soluções exponencial no número de tarefas, no número de equipes e no número de períodos, mais especificamente $O\left(2^{nmp}\right)$. A notação O é utilizada em vez de θ devido às restrições do problema, as quais reduzem a quantidade de soluções válidas.

A complexidade acima é expressa em função dos dados de entrada do modelo matemático. Para expressar o espaço de soluções em função dos dados do problema em si, tarefas e recursos, basta considerar a quantidade de equipes em função da quantidade de recursos em um cenário hipotético de pior caso, onde todas as combinações possíveis de recursos formam o conjunto de equipes, o que implica em $m = C_{r,1}+C_{r,2}+\ldots+C_{r,r} = 2^r$ possíveis combinações de recursos, onde $r = |R|$ é a cardinalidade do conjunto de recursos e $C_{r,x}$ é a combinação de r recursos tomados em equipes de tamanho x.

Além disso, ainda existe a questão da quatidade de esforço empregado por cada recurso. Como discutido na subseção 3.3.4, cada recurso deve ser alocado a cada tarefa em valores que variam de 0% ao valor máximo permitido para o recurso, com uma variação de 25% entre cada valor possível. Sem perda da generalidade, consideremos o caso em que todos os recursos possuam o mesmo valor de dedicação máxima (geralmente algo entre 100% e 150%), o que significa um número constante k de opções para cada alocação recurso–tarefa (algo entre 5 e 7, considerando o máximo entre 100% e 150%). Assim, para considerar também esta questão, r passa a ser não somente a quantidade de recursos disponíveis, mas as $k^{|R|}$ combinações de valores de dedicação, resultando em até $m = 2^{k^{|R|}}$ possíveis equipes.

Finalmente, o espaço de soluções em função das variáveis do problema é $O\left(2^{np2^{k^r}}\right)$, onde $n = |I|$ é o número de tarefas, $r = |R|$ é o número de recursos e $p = |T|$ é o número de períodos – ou espaços de tempo – definidos. Vale ressaltar que tal complexidade aparentemente absurda representa o pior caso contendo todas as combinações de equipes possíveis, embora, na prática, essa quantidade será muito menor. De qualquer forma, fica mostrado que o espaço de soluções do problema é exponencial em relação aos seus dados de entrada.

Investigado o espaço de soluções do problema, outra forma de evidenciar seu grau de complexidade é assemelhando-o com algum problema bem conhecido. Para isso, vamos considerar a seguinte simplificação do problema proposto: dado um conjunto de recursos, todos do tipo empregado e sem possibilidade de hora extra, ou seja, com

dedicação máxima de 100%; dado um conjunto de *skills* com somente um elemento, ou seja, somente um tipo de habilidade; e dado um conjunto de tarefas a serem executadas, todas as quais de igual importância e sem *deadline*. Consideremos também que cada tarefa tem como requisito o *skill* supracitado e todos os recursos possuam nível de proficiência "normal" em relação a ele, além de nível de experiência também "normal" em relação a cada uma das tarefas. E, finalmente, vamos supor que os recursos estejam disponíveis durante todo o projeto e que não exista *deadline* para o projeto.

O problema descrito no parágrafo anterior, uma simplificação do problema formulado neste capítulo, é uma variação do problema RPCSP, onde um conjunto de tarefas precisa ser distribuídas a um conjunto de recursos com capacidade máxima, com objetivo de minimização do tempo total de execução. Dado que o RPCSP é um conhecido problema pertencente à classe dos problemas NP-difíceis (Herroelen et al., 1998), pode-se concluir que o problema proposto, mesmo numa versão bem simplificada, é um problema NP-difícil.

Argumentações sobre a complexidade acerca do desenvolvimento de cronogramas para projetos de software, tarefa comum no cotidiano de gerentes de projetos, já haviam sido apresentadas. Esta seção comprova aqueles argumentos demonstrando a complexidade da formulação proposta para o problema, fato que evidencia o que ocorre na maior parte das abordagens baseadas em SBSE: a necessidade do uso de técnicas aproximadas devido à complexidade do problema.

3.6 Questões Relevantes

Após discutir, formular e analisar o problema proposto, essa seção apresenta algumas questões que merecem ser discutidas.

3.6.1 Aplicabilidade

Apresentado o problema proposto, a primeira pergunta que surge é sobre sua aplicabilidade em projetos reais. Como discutido anteriormente, a abordagem proposta tem o intuito de apoiar o gerente de projetos, oferecendo todas as vantagens advindas de uma abordagem automatizada. Para isso, durante o processo de desenvolvimento da abordagem, principalmente no que tange a formulação do problema, entrevistas com gerentes de projetos e conversas formais com especialistas foram realizadas com o intuito de tornar a abordagem realmente aplicável a situações reais da gerência de projetos.

A abordagem pode ser aplicada em diferentes contextos de planejamento, desde uma visão macro, como um planejamento geral de todo o projeto, a uma visão micro,

na qual um planejamento mais detalhado é realizado para um conjunto menor de tarefas, como no planejamento de uma iteração do produto. Nesse sentido, basta que instâncias do problema sejam formuladas de acordo com as necessidades de cada caso. Num planejamento macro, por exemplo, os processos relativos ao desenvolvimento podem ser vistos com as tarefas, enquanto as habilidades podem ser generalizadas por funções (analista, implementador, etc) em vez de *skills* mais detalhados. Já num planejamento micro, atividades mais específicas podem ser alocadas a um conjunto de recursos pré-selecionado, com habilidades específicas àquelas tarefas.

A fim de demonstrar a aplicabilidade da abordagem, foi conduzido um estudo de caso utilizando dados de uma empresa de desenvolvimento de software. A descrição, condução e análise desse estudo são descritas no capítulo 5.

3.6.2 Extensibilidade

Uma vantagem de uma formulação voltada à aplicação de técnicas de otimização é a facilidade em estendê-la. De acordo com as necessidades da organização ou o desejo do gerente, é possível reformular o problema de forma que ele agregue novos aspectos não considerados na atual formulação.

Uma vez que a abordagem baseia-se em uma técnica multiobjetivo, novos objetivos podem ser considerados. Um exemplo seria considerar o gerenciamento de riscos no planejamento de cronogramas, incluindo um objetivo de minimização de riscos no qual tarefas de maior risco têm sua execução priorizada. Colares et al. (2009) utilizam uma função objetivo nesse sentido para planejar liberações de um produto de software. O desejo do gerente de projetos também podem ser incorporado à formulação inserindo-se novas restrições ao problema. Esse artifício possibilita ao gerente restringir, por exemplo, datas, tamanho de equipes, duração de tarefas e pré-alocações de recursos específicos a tarefas consideradas críticas.

É importante ressaltar que os artifícios discutidos nesta seção representam apenas a *possibilidade* de extensão do problema, não tendo sido essas extensões incorporadas à abordagem. Isto é, a abordagem proposta trabalha apenas com a formulação já apresentada nas seções deste capítulo.

3.6.3 Limitações

Naturalmente, o problema formulado também possui suas limitações. Uma delas é a não preempção de tarefas, isto é, a impossibilidade de interrompê-las num dado momento e continuá-las posteriormente. O mesmo vale para alterações nas equipes no decorrer do projeto. Embora essas restrições pareçam incompatíveis com o mundo

real, é importante ressaltar que a abordagem proposta é voltada ao *planejamento* de projetos, e que, embora preempção de tarefas e alterações nas equipes sejam comuns no andamento de um projeto de software, essas situações não costumam ser previstas nos cronogramas, pois geralmente elas ocorrem devido a fatores imprevisíveis.

Outra limitação que pode ser apontada é a inexistência de margens de erros nas estimações relativas ao esforço e à duração das tarefas. Embora tenham sido estudadas formas de se trabalhar com margens de erros, o autor concluiu que a imprevisibilidade do tema impossibilita a formulação de um modelo que considere esta questão com uma margem aceitável de exatidão.

Erros de estimativa, mudanças de pessoal e alterações no cronograma fazem parte de uma gama fatores imprevisíveis relativos à natureza dinâmica do desenvolvimento de software, para a qual existem técnicas de monitoramento e controle. Em outras palavras, as questões discutidas nos parágrafos acima, além de boa parte de outras que podem surgir, não são relativas ao planejamento, mas sim ao controle de projetos. Uma vez que o escopo deste trabalho é o planejamento, a abordagem aqui proposta é uma abordagem estática, de forma que aspectos dinâmicos do problema podem ser tratados somente como replanejamentos, conforme discutido na subseção 3.2.12.

Capítulo 4

Um Algoritmo Baseado em Otimização Genética Multiobjetivo

Apresentada a primeira parte da abordagem, relativa ao problema proposto e à sua formulação, o próximo passo é apresentar uma forma de resolvê-lo. Conforme discutido anteriormente, este não é um problema trivial. De fato, a seção 3.5 mostra que, além do problema possuir um espaço de soluções de ordem exponencial, ele pertence à classe dos problemas NP-difícil mesmo considerando uma variação simplificada. Felizmente, como ocorre na maioria dos problemas da engenharia de software, a busca por soluções para o problema não se restringe somente a encontrar a melhor solução (solução ótima), mas também em encontrar soluções "suficientemente boas", isto é, soluções que satisfaçam as necessidades da organização.

Ambos os fatores supracitados, a complexidade do problema e a não limitação a soluções ótimas, são argumentos suficientes para justificar o uso de uma técnica aproximada, ou seja, uma abordagem que consiga obter boas soluções em tempo razoável de execução. Este capítulo descreve um algoritmo nesse sentido, proposto para resolver o problema formulado no capítulo 3.

Inicialmente, a seção 4.1 contextualiza o leitor acerca de técnicas de otimização e de otimização multiobjetivo. Em seguida, a seção 4.2 apresentada e discute a metaheurística utilizada como base para o implementação do algoritmo, enquanto a implementação em si é discutida na seção 4.3. Finalmente, a seção 4.4 analisa a complexidade do algoritmo.

4.1 Técnicas de Busca e a Otimização Multiobjetivo

Técnicas de busca são assim chamadas pois costumam resolver um problema percorrendo o seu espaço de soluções, ou seja, buscando soluções interessantes em um conjunto

de possíveis soluções para o problema. Em geral, essas técnicas procuram uma combinação de variáveis que consigam otimizar um ou mais objetivos do problema e que respeitem as eventuais restrições impostas por ele.

Técnicas aplicadas a um problema de otimização podem ser classificadas em dois tipos: técnicas exatas e técnicas aproximadas. Técnicas exatas são utilizadas para encontrar soluções ótimas, isto é, a melhor solução possível para o problema. Infelizmente, devido à complexidade intrínseca a grande parte dos problemas desta natureza, a eficiência computacional é uma barreira para o uso de tais técnicas. Surgem como alternativa as técnicas aproximadas, que podem ser eficientemente aplicadas a problemas de maior complexidade e ainda assim conseguirem soluções suficientemente boas, ou seja, soluções que geralmente se aproximam do ótimo.

Entre as técnicas exatas, pode-se destacar técnicas clássicas como *branch and bound*, algoritmos gulosos[1] e o método simplex[2]. Entre as técnicas aproximadas, destacam-se as meta-heurísticas, que podem ser descritas como técnicas genéricas para resolução eficiente de problemas complexos. Devido à complexidade da área, a grande maioria dos autores prefere abordar os problemas de engenharia de software utilizando meta-heurísticas, conforme mostra o *survey* realizado por Harman (2007b). Entre as meta-heurísticas mais tradicionais estão inclusas a busca local, a busca tabu e o algoritmo genético. Mais informações sobre técnicas de busca e otimização em geral podem ser encontradas no livro de Burke e Kendall (2005).

4.1.1 Otimização Multiobjetivo

Uma vez que muitos problemas podem apresentar vários possíveis objetivos a serem otimizados, como é o caso neste trabalho, um paradigma que vem ganhando bastante atenção da comunidade acadêmica é a chamada otimização multiobjetivo. Diferentemente de técnicas clássicas de otimização, a otimização multiobjetivo, conforme seu nome sugere, permite a manipulação de vários objetivos em vez de somente um, o que pode trazer diversas vantagens de uso.

Em geral, múltiplos objetivos constantemente traduzem-se como objetivos contrários. Neste trabalho, por exemplo, os objetivos de minimizar o custo e de minimizar as horas extras geralmente vão de encontro à minimização de tempo e de atrasos. Por esse motivo, problemas multiobjetivo não costumam apresentar uma única solução ótima, pois otimizar um objetivo geralmente implica na piora de outros. Assim, surge o conceito de *pareto*, conceito do campo das ciências econômicas cunhado por Joseph Juran,

[1] A abordagem gulosa pode obter soluções ótimas para problemas que estejam de acordo com certas propriedades.

[2] O método simplex é aplicado a problemas de Programação Linear (Luenberger e Ye, 2008)

que adotou uma observação feita por Vilfredo Pareto, o qual observara que 20% da população concentrava 80% dos bens, na Itália do início do século XX. A observação, generalizada por Juran para situações de negócios, ficou conhecida como princípio 80–20. A idéia de que não se pode beneficiar elementos de um sistema sem prejudicar os demais foi estendida a áreas matemáticas, incluindo à otimização multiobjetivo.

Dentro deste contexto, eficiência de pareto, também conhecida como ótimo de pareto, é o conjunto ótimo de soluções para um problema multiobjetivo. O conjunto ótimo é aquele que contém as soluções não dominadas, conforme o conceito de dominância entre soluções. A dominância de uma de uma solução x sobre uma solução y para um problema com o objetivos é tal que sejam satisfeitas as seguinte desigualdades:

$$\forall o \mid f_o(x) \geq f_o(y) \quad e \quad \exists o \mid f_o(x) > f_o(y)$$

Traduzindo o acima exposto, uma solução domina outra se a primeira é melhor em pelo menos um objetivo e não é pior em todos os demais. Portanto, uma solução está no conjunto ótimo de soluções, conjunto também denominado *frente de pareto* ótima, se não existem outras soluções que melhorem algum objetivo sem que se piore outros. A figura 4.1 ilustra o conceito de frente de pareto. Nela, $f1$ e $f2$ são objetivos de minimização e as soluções que não são dominadas estão destacadas na frente de pareto. É mostrado que as soluções A e B não dominam uma à outra, porém ambas dominam a solução C.

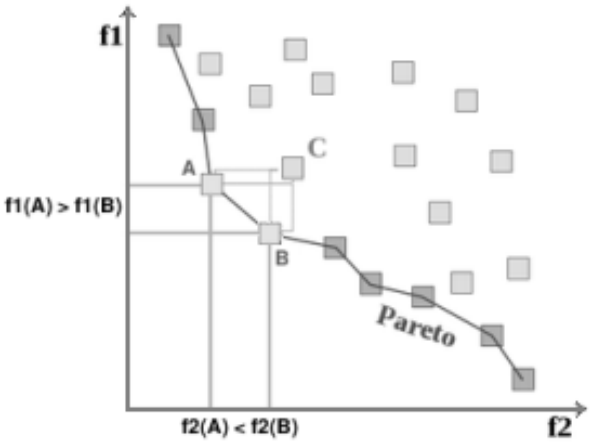

Figura 4.1. Frente de Pareto – (figura adaptada de Wikipedia (2010))

Além de oferecer uma maneira de solucionar problemas que apresentam múltiplos objetivos, a otimização multiobjetivo fornece através da frente de pareto uma maneira de se estruturar a noção de *trade-off*, ou seja, de se realizar escolhas em situações de conflito, onde é preciso considerar o balanceamento de interesses (objetivos). Em

outras palavras, além de prover soluções balanceadas para o problema, a otimização multiobjetivo também oferece uma introspecção acerca daquilo que o envolve.

Várias técnicas para otimização multiobjetivo têm sido propostas, muitas delas baseadas em técnicas tradicionais da otimização clássica. Entre elas está a meta-heurística utilizada como base para o algoritmo aqui proposto, a qual é apresentada e discutida na próxima seção.

4.2 Non-dominated Sorting Genetic Algorithm II

O NSGA-II, técnica proposta por Deb et al. (2002), é provavelmente a mais utilizada meta-heurística para otimização multiobjetivo. Conforme sugere seu nome, ela é baseada na tradicional técnica de algoritmos genéticos, incluindo em suas características conceitos como seleção, cruzamento (*crossover*) e mutação. Além de sua grande aceitação na comunidade científica, alguns dos fatores que levaram à escolha desta meta-heurística são sua eficiência computacional e o bom uso de conceitos como elitismo e espaçamento (granularidade) das soluções.

Em resumo, o algoritmo parte de uma população inicial (conjunto inicial de soluções) e, a cada iteração, novos indivíduos (soluções) são obtidos através dos operadores de cruzamento e mutação e agregados à população anterior (conceito de elitismo). As melhores soluções formam a nova geração, a qual é utilizada na iteração seguinte. Para classificação e seleção de indivíduos, o conceito de frente de pareto, discutido na subseção 4.1.1, é utilizado juntamente com o conceito de granularidade das soluções. Além das tradicionais operações de seleção, cruzamento e mutação, são três as principais operações do NSGA-II: ordenação pela não dominância, atribuições de valores de granularidade e ordenação pela granularidade. A ordenação pela não dominância consiste em ordenar frentes de pareto, enquanto a granularidade é utilizada com intuito de se obter uma boa diversidade de soluções.

A figura 4.2 ilustra uma iteração do algoritmo NSGA-II. A união de soluções anteriores P_t com novas soluções Q_t são ordenadas pelo critério de não dominância e em seguida pela granularidade, formando-se uma nova geração P_{t+1} que passa pelo mesmo processo até que seja satisfeita uma dada condição de parada (geralmente o número de gerações). As operações realizadas a cada iteração do algoritmo têm complexidade de execução de $O(mn^2)$, onde m é o número de objetivos e n é o tamanho da população, isto é, o número de soluções presentes em cada geração. A análise detalhada de sua complexidade e outras informações sobre o algoritmo NSGA-II podem ser encontradas no trabalho de Deb et al. (2002).

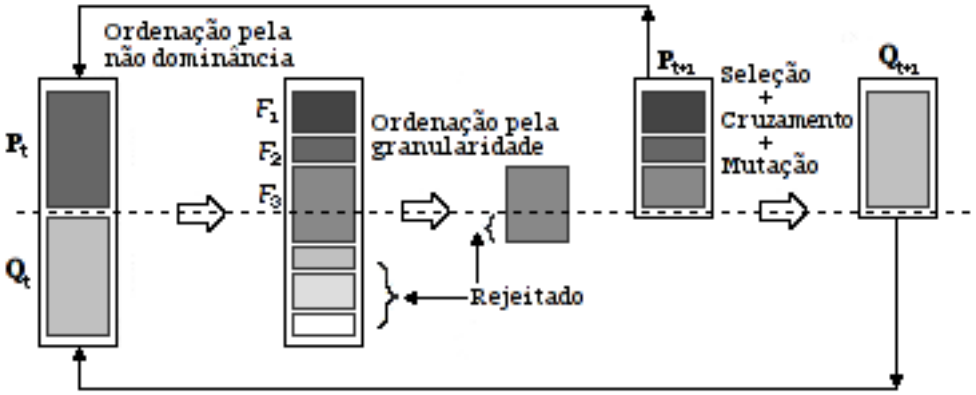

Figura 4.2. Algoritmo NSGA-II

4.3 Implementação

Essa seção descreve detalhes de implementação do algoritmo desenvolvido. Como discutido na seção anterior, sua base é a meta-heurística multiobjetivo NSGA-II. Para a implementação foi utilizado o *framework* jMetal (Durillo et al., 2006), que oferece ferramentas para implementação de diversas meta-heurísticas multiobjetivo utilizando Java. Embora o algoritmo desenvolvido seja baseado no NSGA-II, o uso do jMetal permite uma fácil adaptação para qualquer meta-heurística oferecida pelo *framework*.

Os algoritmos 1 e 2 apresentam pseudocódigos do algoritmo desenvolvido. O primeiro é o procedimento principal, cuja principal função é executar o segundo, que é a modificação do algoritmo NSGA-II. Em seguida, as próximas subseções discutem particularidades da implementação.

Algoritmo 1: Procedimento principal

Data: O projeto de software a ser planejado
Result: Soluções com a alocação dos recursos e o cronograma do projeto
begin

```
// Inicializar variáveis, instanciar classes, etc
inicialização;

// carregar parâmetros e operadores da meta-heurística
carregar parâmentros;
carregar operadores;

// adaptação da meta-heurística (algoritmo 2)
executar NSGA-II modificado;

// Apresentar resultado
escrever soluções;
```

end

Algoritmo 2: Adaptação do algoritmo NSGA-II

Data: n – tamanho da população
Data: k – número de gerações
Data: r – porcentagem de inidivíduos randômicos na população inicial
Data: S[ ] – soluções iniciais pré-informadas
Result: Frente de pareto com as melhores soluções obtidas

```
1  begin
       // Gerando população inicial
2      foreach s ∈ S do
3          avaliar_indivíduo(s);
4          adicionar_indivíduo_à_população(s);
5      while população.tamanho < n do
6          if random() ≤ r then
7              s ← gerar_indivíduo_aleatório;
8          else
9              s ← criar_indivíduo;
10         avaliar_indivíduo(s);
11         adicionar_indivíduo_à_população(s);

       // Evoluindo população
12     while cont < k do
           // Gerar novos indivíduos
13         for i ← 1 to n/2 do
14             p[1,2] ← selecionar_pais;
15             s[1,2] ← cruzamento(p1,p2);
16             mutação(s1);
17             mutação(s2);
18             avaliar_indivíduo(s1);
19             avaliar_indivíduo(s2);
20             aplicar_heurísticas(s1);
21             aplicar_heurísticas(s2);
22             adicionar_indivíduo_à_população(s1);
23             adicionar_indivíduo_à_população(s2);
           // Procedimentos do NSGA-II
24         ordenar_pela_não_dominância;
25         atribuir_valores_de_granularidade;
26         ordenar_pela_granularidade;
           // Incrementar contador
27         cont ← cont + 1;
28     return primeira_frente_de_pareto;
29 end
```

4.3.1 Indivíduos

Cada indivíduo da população representa uma solução para o problema. No decorrer do algoritmo, novos indivíduos são gerados de forma a evoluir a população em busca de obter-se a frente de pareto ótima ou uma frente considerada suficientemente boa. Cada indivíduo é representado por uma matriz de duas dimensões denominada matriz de alocação, na qual uma das dimensões representa os recursos e a outra as tarefas às quais os recursos devem ser alocados. A figura 4.3 apresenta esse conceito graficamente. Na matriz de alocação, cada variável $x_{r,t}$ recebe um valor inteiro entre 0 e 8, o qual é interpretado dividindo-o por 4 de forma a se obter porcentagens de 0% a 200%, que representam o esforço empregado pelo recurso r à tarefa t.

		Tarefas						
		t_1	t_2	t_3		. . .		$t_{\|T\|}$
Recursos	r_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$				$x_{1,\|T\|}$
	r_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$				$x_{2,\|T\|}$
	r_3	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$				$x_{3,\|T\|}$
	$\vdots$							
	$r_{\|R\|}$	$x_{\|R\|,1}$	$x_{\|R\|,2}$	$x_{\|R\|,3}$				$x_{\|R\|,\|T\|}$

Figura 4.3. Representação de uma solução (indivíduo)

4.3.1.1 Análise da Complexidade Espacial

Embora tenha sido implementado em um ambiente sem grande limitação de memória, além de ter sido utilizada uma linguagem que não prima pela economia espacial, pode ser interessante ao leitor uma análise da complexidade espacial da representação das soluções. O próximo parágrafo apresenta essa análise, ressaltando que ela é válida caso cada indivíduo seja representado como um vetor binário, o que aumenta em alguns aspectos a complexidade de implementação do algoritmo.

Uma vez que cada variável $x_{r,t}$ pode assumir 9 valores diferentes, são necessários $\lceil log_2(9) \rceil = 4$ bits para representá-la. Considerando $r = |R|$, $t = |T|$ e n o tamanho da população, cada indivíduo é armazenado com $4 \times r \times t$ bits, e toda a população com $4 \times r \times t \times n$. Portanto, o espaço de armazenamento cresce na ordem de $\theta(nrt)$, ou seja, linearmente em relação ao tamanho da população e à quantidade recursos e de tarefas.

4.3.2 Subproblemas: Alocação e Sequenciamento

A implementação do algoritmo considerou a divisão do problema em dois subproblemas: alocação dos recursos às tarefas e o posterior sequenciamento dessas. Esta divisão deveu-se ao fato de que, de outra forma, questões como a interdependência entre tarefas e a alocação paralela de recursos mostraram-se altamente complexas ou mesmo inviáveis de se manipular diretamente. Uma alternativa considerada fora utilizar uma terceira dimensão – além de recursos e tarefas – relativa ao tempo, abordagem semelhante à de Chang et al. (2008). Porém, ficou claro que o uso de uma dimensão temporal tende a tornar inviável o algoritmo para problemas maiores, exatamente onde o uso de SBSE pode trazer mais vantagens. Uma possível solução para isso seria considerar períodos de tempos maiores, porém essa alternativa também não se mostra atrativa pelo fato de que períodos maiores impossibilitam planejamento mais detalhados e são incompatíveis com tarefas de menor duração.

Pelas razões supracitadas, a divisão do problema mostrou-se atrativa e, como é apresentado no capítulo 5, obteve resultados bastante satisfatórios. Assim, o algoritmo consiste em duas etapas: inicialmente, realiza-se a distribuição das tarefas aos recursos, atribuindo-se valores às variáveis $x_{r,t}$, discutidas na subseção 4.3.1. Em seguida, a partir da alocação realizada, o sequenciamento das tarefas é definido de forma determinística, ou seja, para cada combinação possível de variáveis, o algoritmo produz um único sequenciamento de tarefas, de forma que essa relação 1:1 permita à meta-heurística NSGA-II sempre evoluir em busca da melhor solução. Dessa forma, o primeiro passo, o de alocação, é realizado pela própria meta-heurística NSGA-II ao buscar a melhor combinação de valores para as variáveis $x_{r,t}$ através das operações descritas na seção 4.2 e dos operadores discutidos na subseção 4.3.5, enquanto a segunda etapa, de sequenciar as tarefas, é realizada na função *avaliar_indivíduo*, onde a solução é avaliada em função dos objetivos e restrições.

O sequenciamento de tarefas nada mais é que a definição do cronograma do projeto, onde é determinado o início e fim de cada tarefa. Como supracitado, para cada possível combinação de variáveis, isto é, para cada possível alocação recurso $\times$ tarefa, o sequenciamento das tarefas é calculado deterministicamente. Esse cálculo consiste em alguns passos, descritos no algoritmo 3.

4.3.3 Avaliação de Indivíduos

A função *avaliar_indivíduo* tem o importante papel de avaliar a qualidade de uma solução em relação tanto ao respeito às restrições impostas quanto aos valores das funções objetivos especificadas. É a partir desta avaliação que a meta-heurística NSGA-

Algoritmo 3: Sequenciamento de tarefas

Data: Matriz de alocação $|R| \times |T|$
Result: Tarefas sequenciadas

```
begin
    // Calculando duração de cada tarefa (ver eq.  3.10)
    foreach t ∈ T do
        calcular_duração(t);

    // Início e término de acordo com as interdependências
    foreach t ∈ T do
        ajustar_interdependências(t);

    // Rearranjar tarefas fora do caminho crítico
    foreach t ∈ T do
        reajustar_tarefas(t);

    return início_e_término_das_tarefas;
end
```

II se baseia para classificar, ordenar e evoluir a população de forma a se buscar uma solução ótima para o problema.

Inicialmente, a avaliação das soluções consiste em sequenciar as tarefas conforme descrito no algoritmo 3, descrito na seção anterior. Além disso, também são calculados os esforços diário e total dos recursos, informações que serão utilizadas para cálculo dos objetivos e restrições e também para a aplicação de heurísticas. O cálculo da dedicação diária de cada recurso é realizado por períodos, conforme ilustra a figura 4.4.

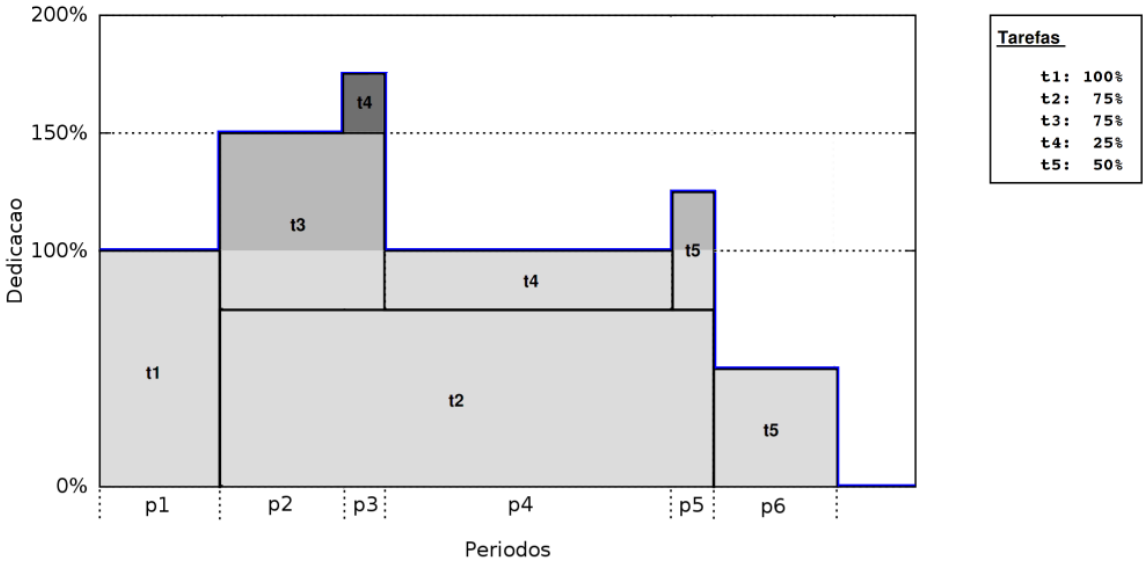

Figura 4.4. Exemplo de cálculo de dedicação e horas extras de um recurso

A figura 4.4 apresenta um exemplo de cálculo da dedicação e das horas extras de um recurso qualquer. O eixo X representa a dimensão temporal, que é dividida em períodos de acordo com os inícios e términos das tarefas. O eixo Y representa a dedicação do recurso em cada período. No exemplo ilustrado, o recurso é alocado a cinco tarefas conforme os valores contidos no quadro ao lado do gráfico. Supondo que o limite de dedicação diária do recurso seja de 150%, sendo 100% de trabalho regular e até 50% de horas extras, então o recurso estaria realizando horas extras nos períodos $p2$ e $p5$ e a ainda estaria sendo quebrada a restrição de dedicação máxima diária no período $p3$, o que faria desta alocação uma solução inválida para o problema. Observa-se também que o recurso é subutilizado no período $p6$.

4.3.4 Inicialização

A inicialização do algoritmo consiste em configurar o registro de anotações (*logger*), inicializar variáveis, instanciar classes e carregar o projeto. Os três primeiros são tarefas triviais, já o segundo merece ser discutido.

Além da possibilidade de criar um projeto fazendo-o objeto por objeto (projeto, recursos, tarefas, *skills*, e etc), existem também formas de automatizar esse processo. Uma delas é defini-lo em documentos texto utilizando uma sintaxe predefinida. Outra forma, que visa integrar a abordagem com ferramentas tradicionais da gerência de projetos, é instanciar um projeto a partir de arquivos do MS Project[3] (Microsoft Corporation, 2010) e do Gnome Planner[4] (GNOME Project, 2010). Para isso, utiliza-se a biblioteca MPXJ (Packwood Software, 2010), a qual permite ainda a integração com o software Primavera[5] (Oracle Corporation, 2010) e com bases de dados do Microsoft Project Server[6] e do Primavera[7].

Após a inicialização, são carregados os parâmetros do algoritmo NSGA-II, além dos operadores. Entre os parâmetros, estão o tamanho da população, a quantidade de gerações e soluções pré-geradas que podem ser utilizadas como indivíduos da população inicial. As duas subseções seguintes discutem sobre os operadores utilizados e sobre as soluções iniciais do algoritmo.

[3] Arquivos MPP, MPX e XML (MSPDI)
[4] Arquivos PLANNER
[5] Arquivos XER
[6] Base da dados Microsoft Access
[7] Banco de dados Oracle

4.3.5 Operadores

São três os operadores utilizados no algoritmo: seleção, cruzamento e mutação. Para seleção e mutação foram utilizados, respectivamente, Torneio Binário e *Bit Flip*, operadores clássicos providos pelo *framework* jMetal. Já o operador de cruzamento foi implementado especificamente para o problema.

O Torneio Binário seleciona um indivíduo escolhendo o melhor entre dois indivíduos aleatórios. A escolha do melhor é realizada considerando o critério da não dominância e da granularidade. A mutação *Bit Flip* consiste em alterar aleatoriamente o valor das variáveis $x_{r,t}$ com uma dada probabilidade (geralmente pequena). Na implementação, o valor da probabilidade é inverso à quantidade de variáveis da solução, o que significa que a mutação aplicada a um indivíduo irá alterar, em média, uma de suas $|R| \times |T|$ variáveis.

A implementação do operador de cruzamento se deve ao fato do indivíduo ser representado numa matriz em vez de um vetor, de forma que operadores unidimensionais tradicionais fariam menos sentido que um *crossover* multidimensional. Foi então implementado um cruzamento bidimensional (que pode ser multidimensionalmente estendido) baseado no cruzamento tradicional *Single Point*. Para isso, é selecionado um ponto aleatório (x, y) e, a partir dele, invertidas as informações genéticas dos indivíduos pais, formando-se dois novos indivíduos filhos. A figura 4.5 ilustra o cruzamento multidimensional, onde dois indivíduos são combinados a partir do ponto $(2, 2)$. É bom ressaltar que a inversão genética é realizada com uma dada probabilidade (geralmente alta), que nesta implementação foi de 90%.

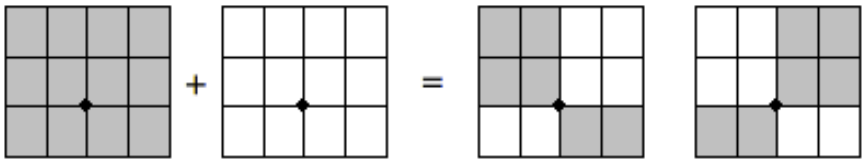

Figura 4.5. Exemplo de cruzamento bidimensional

4.3.6 Soluções Iniciais

O população inicial é crucial para a eficácia do algoritmo, pois é a partir de sua evolução que novos indivíduos são gerados até serem obtidas as soluções que comporão a frente de pareto final. Uma prática comum é utilizar uma população inicial de variáveis aleatórias. Porém, diante da complexidade do problema em questão, uma abordagem

puramente aleatória pode resultar em soluções finais aquém do desejado. Assim, foi desenvolvida uma maneira de se melhorar a população inicial, discutida abaixo.

As linhas 2 a 11 do algoritmo 2 mostram a geração da população inicial. Inicialmente, o algoritmo aceita como entrada soluções pré-geradas, seja pelo próprio gerente ou mesmo por outra abordagem qualquer. Essa possibilidade é interessante pois permite ao gerente utilizar seu ponto de vista como um ponto inicial do algoritmo, além de permitir a integração com outras abordagens. As soluções iniciais são passadas como arquivos do MS Project ou do Gnome Planner.

Somada à utilização de soluções previamente informadas, que é uma alternativa opcional para enriquecer a população inicial, a criação artificial de indivíduos é outra questão considerada. Para isso, além da geração de indivíduos aleatórios, alguns indivíduos são criados utilizando uma heurística gulosa, que realiza alocações por ordem de prioridade, respeitando a seguinte ordem: tarefas que fazem parte caminho crítico, tarefas de menor *deadline*, tarefas que possuem maior nível de importância e tarefas de maior duração. Obviamente que isso é feito respeitando sempre que possível as restrições de habilidades, dedicação máxima e disponibilidade.

4.3.7 Heurísticas

A geração contínua de novos indivíduos, mostrada nas linhas 13 a 23 do algoritmo 2, baseia-se nas conhecidas e supracitadas etapas de seleção, cruzamento e mutação, além da etapa de avaliação dos indivíduos gerados, que consiste verificar as restrições e em calcular os valores das funções objetivos, onde incluim-se o cálculo de produtividade, o ajuste do cronograma de acordo com as interdependências e etc. Porém, em meio a essas foi acrescida uma nova etapa que consiste na aplicação de heurísticas aos indivíduos gerados. O objetivo das heurísticas é diminuir a quantidade de soluções inválidas, isto é, soluções que não respeitam as restrições do problema. O algoritmo 4 mostra o pseudocódigo relativo à aplicação das heurísticas.

No algoritmo 4, são aplicadas aleatoriamente quatro heurísticas: adicionar *skills*, remover *skills*, superalocação e subalocação. A primeira consiste em, para as tarefas cujas habilidades não foram satisfeitas, adicionar recursos em vista de satisfazê-las; a segunda é o caso inverso, onde são desalocados os recursos que não possuem habilidades requeridas pela tarefas; a terceira consiste em diminuir a carga de dedicação de recursos superalocados, isto é, que estão com trabalho acima da dedicação máxima; e, finalmente, a quarta é a situação oposta, onde recursos cuja dedicação está abaixo do máximo têm sua carga de trabalho aumentada.

As heurísticas não são aplicadas a todas as soluções. Existe uma probabilidade para que isso aconteça, a qual é definida na implementação como 80%, o que significa que, em

Algoritmo 4: Aplicação de Heurísticas

Data: s – solução (indivíduo) a serem aplicadas as heurísticas
Data: p – probabilidade
Result: Solução modificada pelas heurísticas

```
1  begin
2      if random() < p then
3          heurística[1] ← "Adicionar Skills";
4          heurística[2] ← "Remover Skills";
5          heurística[3] ← "Superalocação";
6          heurística[4] ← "Subalocação";
7          permutação ← criar_permutação_aleatória(4);
8          for i ← 1 to 4 do
9              n ← permutacao[i];
10             heurística[n](s);
11     if s foi modificada then
12         avaliar_indivíduo(s);
13     return s;
14 end
```

média, 80% dos indivíduos gerados passam pela aplicação de heurísticas. No entanto, após toda a população tornar-se viável, ou seja, quando todos os indivíduos satisfazem as restrições do problema, a probabilidade da aplicação de heurísticas diminui para um valor consideravelmente menor, que na implementação foi de 5%. Ao final do processo, caso a solução tenha sido modificada por alguma das heurísticas, é necessário reavaliá-la.

4.3.8 Tratamento de Soluções Inválidas

Em todo algoritmo de busca, a validez das soluções é um fato que deve ser considerado. Embora o ideal seja sempre trabalhar com soluções válidas, isto é, soluções que satisfaçam as restrições apresentadas pelo problema, tal abordagem pode ser inviável ou mesmo impossível dependendo do problema em questão. É o caso do problema abordado neste trabalho, devido à complexidade de variáveis e restrições nele envolvidas. Assim sendo, a abordagem utilizada no algoritmo para tratar soluções inválidas consiste em penalizar todas as soluções que apresentem quebra de restrições, de forma que soluções válidas sejam sempre priorizadas. No pseudocódigo apresentado no algoritmo 2, essa priorização é feita na função *avaliar_indivíduo*.

Embora simples, a penalização de soluções inválidas, abordagem utilizada no al-

goritmo, produz bons resultados. Ao analisar a execução do algoritmo em diferentes instâncias do problema, observa-se que o conjunto de soluções, mesmo se partindo de uma população de indivíduos inviáveis, tende sempre a evoluir para a viabilidade através das operações da meta-heurística NSGA-II e das heurísticas discutidas na subseção anterior. Uma vez que a população se torna composta somente por soluções viáveis, a aplicação das heurísticas é diminuída, de forma a melhorar o desempenho computacional e a também de permitir uma maior diversidade na busca.

4.3.9 Apresentação dos Resultados

Embora a implementação por ora não contemple uma ferramenta com interface gráfica voltada ao usuário final (uma vez que a abordagem ainda encontra-se no estado de pesquisa), a forma de apresentação dos resultados deve ser considerada e discutida, pois ela é necessária tanto para a análise de resultados quanto para estudo da interação da abordagem com o usuário, que neste caso é o gerente de projetos.

Para a apresentação dos resultados, a abordagem utiliza mais uma vez a integração com conhecidas ferramentas de gerência, que são: MS Project (Microsoft Corporation, 2010), OpenProj (Serena Software, 2010) e Gnome Planner (GNOME Project, 2010). Para isso, as soluções resultantes (aquelas que compõem a frente de pareto obtida) são traduzidas para arquivos relativos a essas ferramentas. Os resultados também são apresentados em um relatório textual, que inclui as alocações, o cronograma e valores obtidos para objetivos e restrições. Além disso, existe ainda uma interação com o programa de geração de gráficos gnuplot (Williams e Kelley, 2007), onde gráficos com os valores dos objetivos são gerados e exibidos dinamicamente durante a execução do algoritmo, além de serem disponibilizados ao seu final.

4.4 Análise da Complexidade

Esta seção analisa a complexidade dos algoritmos apresentados neste capítulo. O núcleo da complexidade está no algoritmo 2, relativo à execução da meta-heurística NSGA-II, que compreende a geração contínua de soluções e a sua classificação em frentes de pareto ordenadas. Considerando n o tamanho da população, k o número de gerações, r a quantidade de recursos, t a quantidade de tarefas e s a quantidade de *skills* (habilidades), os próximos parágrafos apresentam a análise da complexidade do algoritmo.

Para avaliar uma solução (função *avaliar_indivíduo*), é necessário calcular o início e o término de cada tarefa, calculando sua duração e, consequentemente, a produtividade das equipes. Assim, para cada tarefa é necessário verificar cada recurso e cada possível *skill*, de forma que são necessárias até $O(rts)$ operações. Além da duração da tarefa,

outro fator que interfere nos início e término das tarefas é a possibilidade de adiamento de tarefas que não fazem parte do caminho crítico do projeto, em vista de evitar o acúmulo de trabalho em certos períodos, em especial no início do projeto. Aqui, o caminho crítico não é calculado em relação ao esforço estimado para cada tarefa, mas sim em relação à sua duração, obtida através da produtividade calculada, de forma que esse cálculo precisa ser feito para cada solução avaliada. O custo desse cálculo e do rearranjo de tarefas é de $\theta(t^2)$ operações.

Durante a avaliação de soluções também são calculadas as quantidades de esforço total e esforço individual em cada período de tempo. Conforme ilustrado na figura 4.4, cada início e término de uma tarefa formam uma divisão entre dois períodos, de forma que, no pior caso (quando os inícios e términos não se coincidem vez alguma), há $2t-1$ períodos. Uma vez que o cálculo de esforço em cada período precisa ser para feito cada recurso em relação a todas as tarefa, a complexidade deste cálculo é de $O(rt^2)$.

Os cálculos das funções objetivo são realizados em seguida. Para o objetivo relativo ao tempo total do projeto, cada tarefa precisa ser analisada em relação ao seu término, o que resulta em $\theta(t)$ operações. O mesmo vale para o objetivo de minimização de atrasos. Em relação aos custos e às horas extras, como as quantidades de esforços de cada recurso já foram calculados, basta que cada recurso tenha seu custo ou hora extra calculado de acordo com seu tipo, implicando em $\theta(r)$ operações.

Por fim, há a avaliação das restrições. As habilidades de cada tarefa são verificadas para cada recurso, resultando na complexidade de $O(rts)$. Em relação à dedicação máxima, em cada período de tempo são analisados todos os recursos, o que resulta em $O(rt)$ operações. Finalmente, a não alocação dos recursos indisponíveis é verificada para cada tarefa em cada período, o que significa uma quantidade de operações na ordem de $O(rt^2)$.

Como na maioria dos casos a ordem de t é maior que a de r, e ambos são bem maiores que a ordem de s, a complexidade dominante entre todas as supracitadas é $O(rt^2)$, que é, pois, a complexidade da função *avaliar_indivíduo*.

Para a geração aleatória de soluções (função *gerar_indivíduo_aleatório*) gera-se valores aleatórios para as $r \times t$ variáveis, realizando-se assim $\theta(rt)$ operações. Para a geração artificial de indivíduos (função *criar_indivíduo*) a heurística utilizada requer o cálculo do caminho crítico do projeto (desta vez em relação ao esforço estimado), de forma que a complexidade torna-se $\theta(rt + t^2)$. Conforme supracitado, a ordem de t costuma ser maior que a de r, assim prevalecendo a complexidade $\theta(t^2)$.

As complexidades apresentadas nos parágrafos acima, de avaliar uma solução e de criar soluções, ajudam a calcular o custo total de geração de uma população inicial de tamanho n, que é $O(nrt^2)$. A evolução dos indivíduos a partir da população inicial

tem sua complexidade analisada abaixo.

A geração de novos indivíduos envolve as operações de seleção ($\theta(1)$), cruzamento ($O(rt)$) e mutação ($\theta(1)$). Em seguida, os indivíduos são avaliados ($O(rt^2)$) e as heurísticas são aplicadas. O algoritmo 4 descreve a aplicação de quatro heurísticas. Nas duas primeiras, relativas às habilidades, para cada *skill* de cada tarefa é verificada a alocação de cada recurso, o que implica em até $O(rts)$ operações. As duas outras, relativas ao esforço alocado, consistem em avaliar os esforços dos recursos em cada período de tempo (já calculados) e eventualmente modificá-los. Para isso, são analisados cada período e cada recurso e, em caso de modificação de valores, são buscadas tarefas candidatas a terem seu esforço aumentado ou reduzido, resultando em um têm custo de ordem $O(rt^2)$. Novamente devido à ordem de s ser geralmente bem menor que t e r, a complexidade dominante na geração de novos indivíduos é de $O(rt^2)$. Como são gerados n indivíduos a cada geração, a complexidade torna-se $O(nrt^2)$.

Uma vez obtidos os indivíduos da próxima geração, as frentes de pareto são calculadas e ordenadas utilizando os procedimentos da meta-heurística NSGA-II. Conforme mostrado por Deb et al. (2002), a complexidade dessas operações é de $O(mn^2)$, onde m é o número de objetivos. Dado que para esta abordagem $m = 4$, ou seja, m é um número constante, a complexidade das três operações do NSGA-II nesta implementação é de $O(n^2)$.

Temos, então, que a complexidade de execução relativa à evolução da população é de $O(nrt^2 + n^2)$. Considerando um número k de gerações, temos $\boxed{O\left(k \times (nrt^2 + n^2)\right)}$ como complexidade total do algoritmo, uma vez que esse valor domina a complexidade de geração da população inicial. Esse valor demonstra a escalabilidade do algoritmo para projetos de grande porte, dado que a sua complexidade de execução é linear à quantidade de recursos e quadrática à quantidade de tarefas, ou seja, é um algoritmo com baixa ordem de magnitude polinomial em relação às entradas do problema.

Capítulo 5

Estudos Experimentais

Senso comum, intuição e especulação não são fontes confiáveis de conhecimento. Ao contrário, em qualquer disciplina, avanços científicos estão relacionados à construção de modelos que possam ser testados, através de estudos empíricos, no intuito de verificar se o entendimento atual do campo está correto, diferenciando o que é realmente verdadeiro do que apenas acredita-se ser um fato científico. É o que afirmam Basili et al. (2000). Infelizmente, porém, embora tenham valor científico altamente reconhecido, avaliações experimentais constituem um universo não tão bem explorado por engenheiros de software, pelo menos não tanto quanto o necessário. Juristo e Moreno (2001) discutem, entre outros tópicos relacionadas à experimentação em engenharia de software, sobre a escassez de experimentos na área e suas razões. Visando melhorar esse cenário, Wohlin et al. (2000) apresentam uma boa introdução à realização de experimentos em pesquisas de engenharia de software, tendo sido a principal referência utilizada para avaliar e validar esta pesquisa através de estudos empíricos, os quais são apresentados neste capítulo.

A abordagem proposta foi avaliada em diferentes cenários, cada qual com seus objetivos e especificidades. As próximas seções apresentam os cenário considerados, analisando e discutindo os resultados obtidos em cada, sendo antes apresentada uma introdução aos conceitos de experimentação em engenharia de software, na seção 5.1. Posteriormente, a seção 5.2 apresenta um estudo de caso no qual a abordagem foi aplicada a um projeto de uma organização desenvolvedora de software. A seção 5.3 apresenta um experimento realizado com profissionais da área em busca de se analisar quantitativamente os possíveis ganhos oferecidos pela abordagem. E, finalmente, a seção 5.4 apresenta testes realizados com o intuito de verificar aspectos da abordagem proposta e do próprio problema abordado.

5.1 Experimentação em Engenharia de Software

Esta seção discute brevemente sobre a realização de experimentos em pesquisas de engenharia de software e apresenta alguns conceitos utilizados neste capítulo.

Avaliações experimentais geralmente são tarefas complexas e dispendiosas, especialmente se consideradas especificidades da engenharia de software. Diferentemente de outros setores industriais, o desenvolvimento de software não se baseia na construção contínua de um mesmo produto, de forma que as variações tendem a dificultar análises estatísticas, por exemplo. Além disso, a engenharia de software baseia-se principalmente em valores intelectuais, ou seja, em seres humanos, o que implica em maior complexidade e muitas vezes em mais gastos.

Apesar das dificuldades supracitadas, a experimentação não pode ser desconsiderada do método científico, conforme discutido na introdução deste capítulo. Glass (1994) sintetiza quatro métodos de pesquisa: científico, aplicado, empírico e analítico. Entre esses, o método empírico, comumente aplicado às ciências sociais e à psicologia, é o que melhor se aplica à engenharia de software. É utilizando esse método que foram conduzidos os estudos apresentados neste capítulo. Wohlin et al. (2000), principal referência utilizada para este capítulo, apresentam três tipos de estratégias de pesquisa empírica:

- **Survey**: Consiste na obtenção de dados quantitativos e qualitativos através da introspecção de indivíduos que representem a população alvo para o objeto de estudo (técnicas, métodos, ferramentas, etc). Geralmente é realizado através de entrevistas e questionários.

- **Estudo de Caso**: Baseia-se em situações práticas, onde o objeto de estudo é avaliado aplicando-o a um caso real. Restringe-se a uma situação específica e tem um baixo nível de controle em relação aos experimentos, estratégia discutida a seguir.

- **Experimento**: O objeto é estudado em um ambiente controlado, geralmente um laboratório, com regras impostas e específicas. Os participantes geralmente são tratados aleatoriamente, de forma a melhor se representar o universo analisado. Experimentos permitem a manipulação e o controle das variáveis envolvidas.

As estratégias de estudo de caso e experimento foram utilizados nos estudos realizados em vista de avaliar e validar a abordagem proposta no trabalho. Eles são apresentados nas próximas seções.

5.2 Estudo de Caso: Aplicabilidade da Abordagem

O primeiro cenário tem o intuito de validar a abordagem proposta, focando naquela que o autor entende ser a mais importante questão a ser considerada para isso: a aplicabilidade da abordagem em projetos reais de software. Para isso, foi considerado um estudo de caso no qual a abordagem foi aplicada a um projeto de uma organização desenvolvedora de software. Infelizmente, devido a restrições de tempo e principalmente de recursos, foi inviável aplicar a abordagem a um projeto durante a sua execução. Assim, foi considerada a alternativa de utilizar um projeto passado com base nos dados históricos da organização.

A organização utilizada neste cenário foi um laboratório de engenharia de software da Universidade Federal de Minas Gerais, o qual há mais de 10 anos oferece serviços e parcerias para o desenvolvimento de sistemas, implantação de processos e consultorias em TI. Baseando-se em entrevistas com um de seus gerente de projetos e em documentos da organização, como o gabarito do processo de desenvolvimento da empresa e dados de um determinado projeto de desenvolvimento, foi definido um projeto e a ele aplicada a abordagem proposta no trabalho.

5.2.1 Definição do Cenário

A iteração do desenvolvimento da organização foi sintetizada em 7 atividades, especificadas na tabela 5.1 e cujas interdependências são ilustradas na figura 5.1[1]. Para cada caso de uso a ser desenvolvido todas as atividades precisam ser realizadas. Neste cenário, foi considerado o planejamento de uma iteração que incluía 10 casos de usos, totalizando um total de 70 tarefas a serem executadas, às quais foram disponibilizados 22 recursos do tipo empregado. A relação dos recursos disponíveis é especificada na tabela 5.2.

Seguindo o especificado pelo processo de desenvolvimento da organização, as habilidades (ou *skills*) foram sintetizadas em funções individuais, onde cada recurso possui uma função específica, e cada tarefa necessita de recursos que executem certas funções. As tabelas 5.1 e 5.2 expressam mais claramente essa relação recursos × tarefas. Como os dados utilizados não especificavam proficiência e experiência dos recursos utilizados, foi considerado sempre o valor "normal" em vista desses fatores não influenciarem no cálculo de produtividade das equipes. Também foram considerados nulos os fatores de comunicação e de aprendizado, uma vez que a organização não possuía dados que permitisse quantificá-los.

[1]Poderiam ser consideradas também interdependências entre casos de uso, mas, como o projeto utilizado não se identificou restrições desse tipo, o desenvolvimento de cada caso de uso foi considerado independente dos demais.

Tarefa	Esforço por PF	Recursos Necessários
Levantar de Requisitos	1,50 homens-hora	Analista de Requisitos Arquiteto de Requisitos Engenheiro de Usabilidade Desenhista de Usabilidade
Detalhar Requisitos	1,52 homens-hora	Analista de Requisitos Arquiteto de Requisitos
Desenho Externo	4,62 homens-hora	Arquiteto de Requisitos Arquiteto de Software Engenheiro de Usabilidade Desenhista de Usabilidade Implementador
Especificar Testes	1,63 homens-hora	Analista de Testes
Implementar	6,88 homens-hora	Implementador
Verificar	0,37 homens-hora	Analista de Testes
Executar Testes	0,81 homens-hora	Analista de Testes

Tabela 5.1. Atividades do processo de desenvolvimento (por caso de uso)

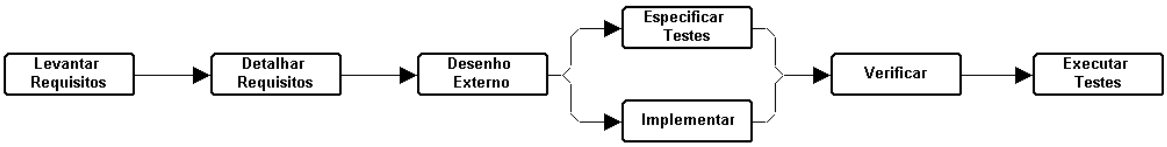

Figura 5.1. Interdependências entre as atividades do processo

Recurso	Quantidade	Custo (salário mensal)[2]
Analista de Requisitos	2	$ 3000
Arquiteto de Requisitos	3	$ 3500
Analista de Usabilidade	2	$ 3000
Desenhista de Usabilidade	2	$ 3500
Arquiteto de Software	1	$ 4000
Implementador	8	$ 2500
Analista de Testes	3	$ 3500

Tabela 5.2. Recursos disponíveis no projeto

[2]Foram utilizados valores fictícios para expressar os salários dos empregados.

5.2.2 Aplicação da Abordagem

Ao projeto descrito na subseção anterior foi aplicada a abordagem proposta neste trabalho. Como parâmetros da meta-heurística NSGA-II foram utilizados, após alguns testes de calibração, 200 para o tamanho da população e também 200 para o número de gerações. A população inicial foi constituída 20% de soluções randômicas e 80% de soluções geradas pela heurística de criação de indivíduos discutida em 4.3.6. O tempo de execução do algoritmo em um notebook comum, com processador Intel Core 2 Duo T5750 e 2 GB de memória RAM, foi de aproximadamente 3 minutos.

A figura 5.2 apresenta um gráfico com as soluções da frente de pareto obtida pelo algoritmo. O fato dos recursos do projeto analisado serem todos do tipo empregado faz com que os objetivos de custo e de horas extras tornem-se redundantes, enquanto o objetivo de atrasos é sempre nulo pelo fato das tarefas do projeto não possuírem *deadline*. Por esses motivos, são ilustrados no gráfico somente os objetivos de custo e de tempo.

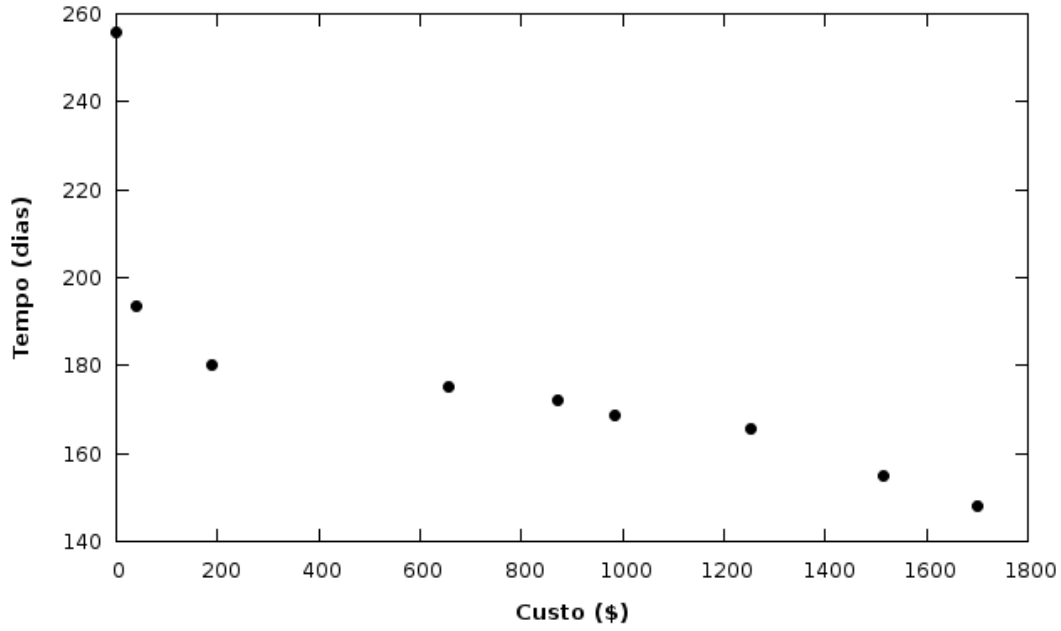

Figura 5.2. Soluções obtidas pelo algoritmo

O gráfico apresenta no eixo Y o tempo total do cronograma, e no eixo X o custo financeiro extra (custo além dos salários fixos). São apresentadas 9 soluções, cada qual com diferentes valores de custo e tempo. Com essas informações em mãos, o gerente de projetos pode analisar e discernir acerca do espaço de soluções do problema, escolher uma solução que melhor atenda aos interesses da organização, ou mesmo realizar um novo planejamento baseado nos resultados obtidos. A figura 5.3 apresenta o gráfico Gantt de uma das soluções apresentadas.

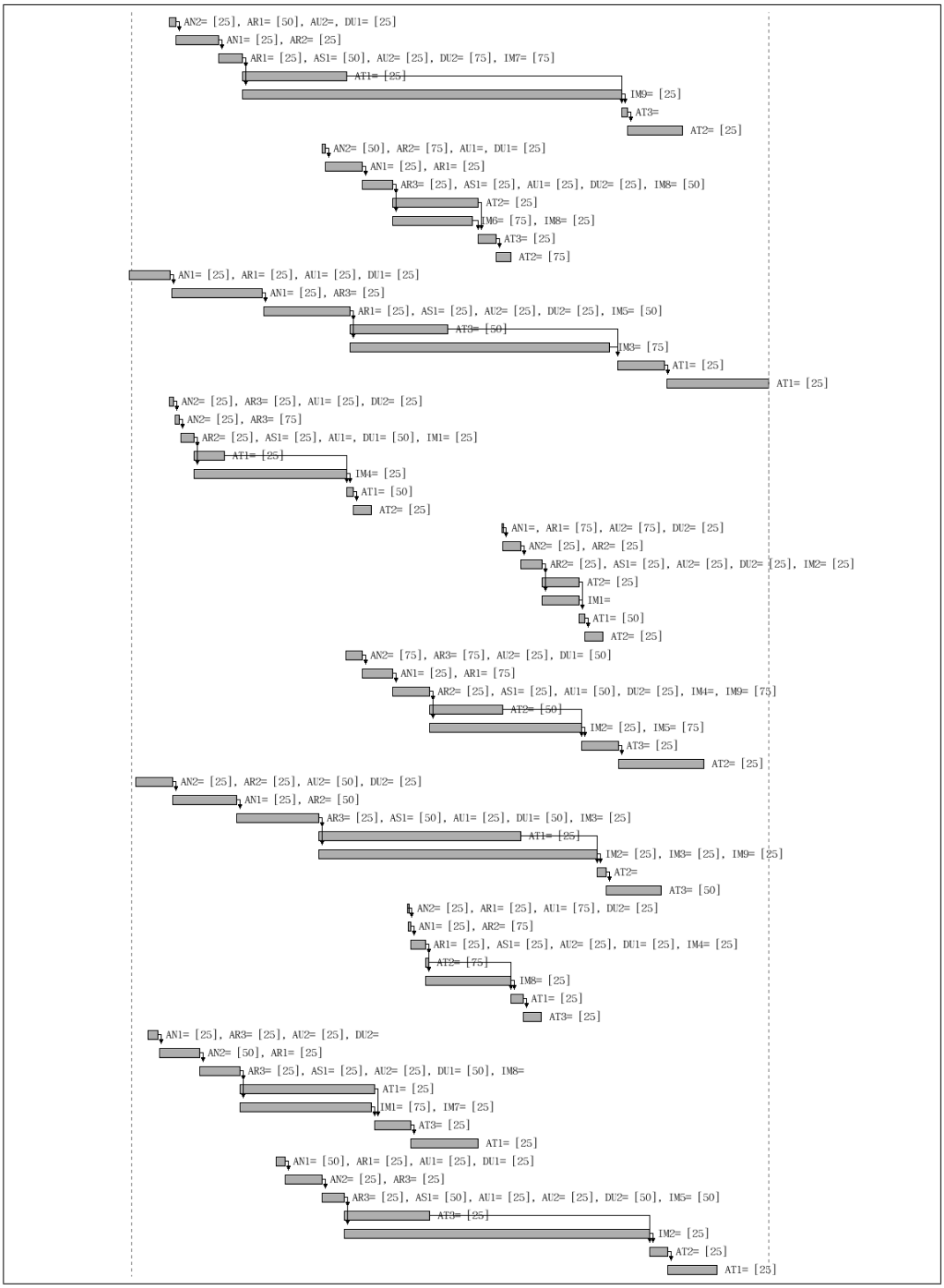

Figura 5.3. Gráfico de Gantt de uma das soluções apresentadas

5.2.3 Análise dos Resultados e Conclusões

Além do gráfico de soluções e do diagrama de Gantt apresentados, o trabalho não apresenta detalhadamente cada solução em vista de não prejudicar o entendimento do leitor diante da quantidade e do detalhamento dos dados, lembrando ainda que

o objetivo deste cenário de teste é avaliar a aplicabilidade da abordagem a projetos reais, e não a qualidade das soluções obtidas[3]. Além do diagrama de Gantt, que já permite uma boa noção das soluções obtidas, o autor apresenta sua análise pessoal dos resultados obtidos, expressa a seguir.

Ao analisar o resultado apresentado, percebe-se a obtenção de soluções bem diversificadas e de aparente qualidade, embora algumas questões possam ser negativamente apontadas, como a subalocação de recursos. Porém, no geral pode-se dizer que a abordagem apresentou boas soluções, principalmente se considerando o fato da participação do gerente em uma natural etapa posterior, de análise e refinamento das soluções.

Em relação à aplicabilidade da abordagem a projetos de desenvolvimento de software, principal objetivo de análise deste cenário de teste, tem-se como atingido o resultado esperado. Com sucesso, um projeto real de desenvolvimento de software foi adaptado à abordagem e esta aplicada a ele, obtendo-se soluções para o planejamento de uma iteração do projeto em relação à alocação de equipes e ao desenvolvimento de cronograma. É importante ressaltar, obviamente, que o estudo de caso envolveu uma situação particular de uma organização específica, de forma que sua generalização deve ser cuidadosa. De qualquer forma, o sucesso no estudo proposto é um indício da aplicabilidade em projetos de software em geral.

5.3 Experimento Empírico: Competitividade das Soluções

Demonstrados indícios da aplicabilidade da abordagem proposta, um experimento foi conduzido com o objetivo de avaliar a qualidade das soluções produzidas. Para isso, resultados obtidos pelo algoritmo foram avaliados em relação à sua competitividade para com resultados de seres humanos, especialistas da área avaliados em um ambiente controlado. Esta subseção apresenta a condução do experimento e a análise de seus resultados, além de discutir brevemente sobre experimentação em SBSE e sobre conceitos acerca de experimentos empíricos.

Existe um grande interesse da comunidade de otimização em avaliar resultados como *human competitive*, no intuito de avaliar a qualidade de soluções alcançadas. Como a engenharia de software geralmente não se enquadra em contextos onde soluções ótimas para o problema são previamente conhecidas, a comparação de resultados de SBSE com resultados obtidos através do julgamento humano torna-se a mais viável e interessante forma de quantificar e julgar como competitivos os resultados de uma abordagem automatizada. Infelizmente, conforme afirma Harman (2007b), experimentos nesse sentido

[3] A análise da qualidade das soluções é apresentada na seção 5.3.

não tem sido realizados por pesquisadores de SBSE. Visando superar essa questão e assim oferecer mais uma contribuição à comunidade, além de obviamente avaliar da qualidade da abordagem proposta, este experimento foi conduzindo.

Uma exceção à supracitada escassez de trabalhos relativos ao tema é o recente trabalho de Souza et al. (2010), o qual apresenta um extensivo estudo empírico no intuito de avaliar a competitividade de SBSE em relação a soluções obtidas por seres humanos. Esse trabalho apresentou fortes indícios de que SBSE é capaz de produzir soluções ditas *human competitive*.

5.3.1 Condução do Experimento

Wohlin et al. (2000) dividem um experimento empírico em cinco fases: definição, planejamento, operação, análise e apresentação. O experimento seguiu todas as etapas conforme recomendado, com o objetivo de atingir a seguinte meta definida:

> *Analisar* ***a abordagem proposta*** *com o propósito de* ***avaliá-la*** *em respeito à* ***sua eficácia e eficiência*** *no ponto de vista da* ***gerência de projetos*** *no contexto do* ***planejamento de projetos de software****.*

Para isso, foram formuladas hipóteses a serem investigadas. O objetivo disso é analisar e quantificar, através do experimento, possíveis e diferentes ganhos que a abordagem pode trazer à gerência de projetos. Abaixo, as hipóteses formuladas:

1. **Competitividade**: a abordagem proposta produz resultados melhores ou equiparáveis aos de seres humanos especialistas.

2. **Apoio à qualidade**: a abordagem proposta é capaz de ajudar especialistas a obterem melhores resultados que sem utilizá-la.

3. **Apoio à produtividade**: a abordagem proposta é capaz de ajudar especialistas a obterem resultados em menor tempo que sem utilizá-la.

4. **NULA**[4]: a abordagem proposta não é capaz de obter melhoramentos à atividade.

Participaram do experimento 10 indivíduos, estudantes e profissionais de ciência da computação. Desses, a maior parte foi formada por especialista em engenharia de software e por indivíduos com experiência no desenvolvimento de software e em gerência de projetos. Naturalmente, um conjunto ideal de participantes seria todo constituído por gerentes de projetos experientes na atividade em questão. Infelizmente, tal situação

[4]A hipótese NULA é a negação das demais hipóteses

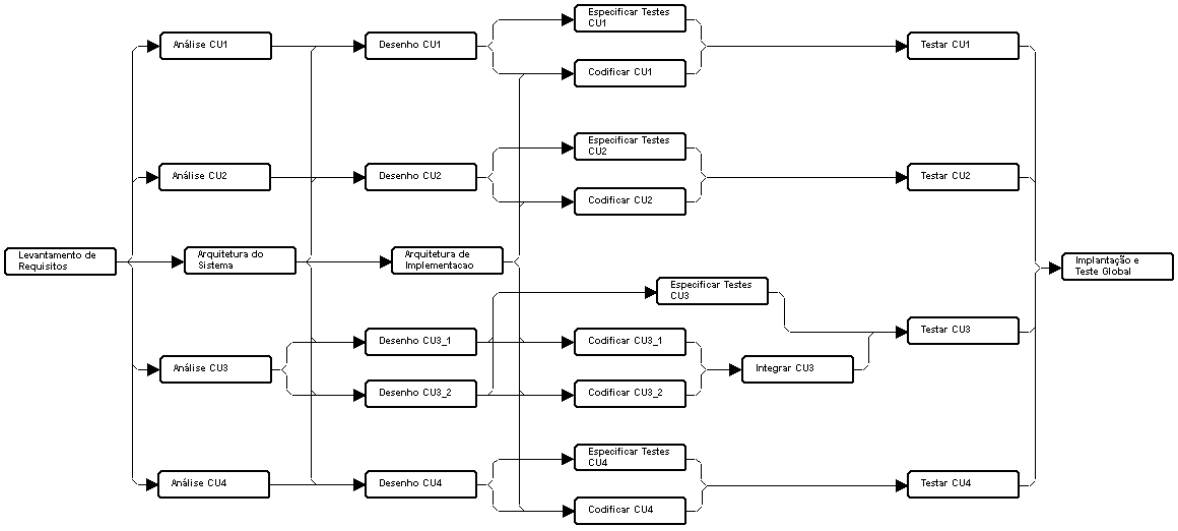

Figura 5.4. Diagrama de rede do projeto proposto

mostrou-se inviável diante dos recursos disponíveis. Apesar dessa limitação, diante da experiência de alguns como gerentes de projetos e especialmente pelo fato de seus resultados terem entoado com os dos demais, o autor considera que os participantes selecionados representaram de forma satisfatória o universo de pesquisa.

Aos participantes foi especificado um projeto de software a ser planejado em relação à alocação de equipes e ao desenvolvimento do cronograma. O projeto constitui-se de 27 tarefas a serem executadas por 18 recursos, dos quais 13 eram empregados da organização e 5 eram prestadores de serviços. O projeto envolvia tarefas de requisitos, arquitetura, análise, desenho, codificação, testes e implantação. A figura 5.4 apresenta o diagrama de rede das tarefas. Um total de 12 habilidades individuais foram consideradas.

Os participantes foram divididos igualmente e aleatoriamente em dois grupos: grupo de controle e grupo experimental. A diferença entre os grupos é que no experimental os participantes receberam soluções previamente geradas pela abordagem e puderam se basear nelas para realização da tarefa, enquanto os participantes do grupo de controle utilizaram somente seu próprio julgamento. Para apoio à tarefa foi utilizado por ambos os grupos o software MS Project 2007. Uma vez que essa ferramenta desconsidera fatores de proficiência, experiência, comunicação interpessoal e aprendizado, não foi possível considerar esses aspectos.

As soluções obtidas foram coletadas e confrontadas com resultados do algoritmo, em vista de se verificar as hipóteses do experimento supracitadas. A próxima subseção apresenta e analisa os resultados obtidos.

5.3.2 Análise dos Resultados

Inicialmente, a figura 5.5, apresenta um gráfico confrontando resultados dos especialistas com os obtidos por três execuções da abordagem. Para simplificação da análise, são ilustrados, sem perda da generalidade, somente os objetivos de tempo e custo. Algumas diferenças podem ser notadas, como o fato de, no comparativo das médias, os especialistas terem sido levemente melhores na minimização do tempo enquanto o algoritmo obteve melhores soluções para o objetivo de custo.

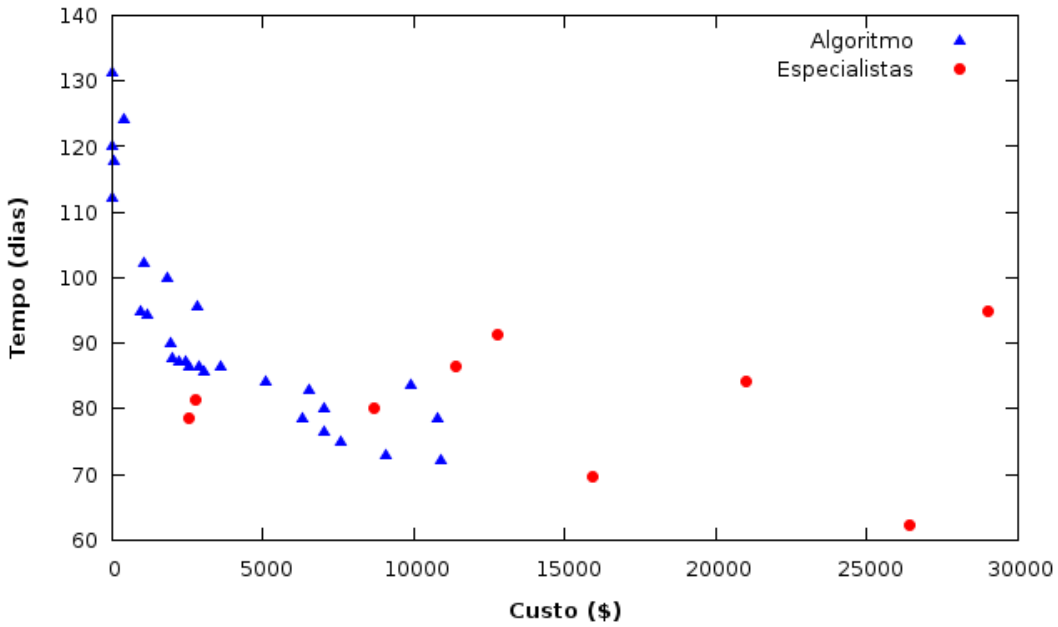

Figura 5.5. Resultado geral: algoritmo × especialistas

Numa análise mais detalhada, pode-se verificar que a abordagem obteve, no geral, resultados melhores que os dos especialistas, diante de dois aspectos: pelo número de soluções pertencentes à frente de pareto relativa às soluções obtidas e devido à viabilidade dessas soluções, conceito relativo ao respeito às restrições impostas pelo problema. A figura 5.6 mostra que grande parte dos especialistas apresentou soluções inviáveis, isto é, soluções que quebraram pelo menos uma das restrições do problema, geralmente por superalocação ou por habilidades não satisfeitas. A figura também ilustra as soluções que fazem parte da frente de pareto. É importante salientar que, embora a figura ilustre soluções inválidas como pertencentes à frente de pareto, isso é feito somente para facilitar o entendimento do leitor e facilitar análises comparativas. Porém, tais soluções obviamente não fazem parte da frente de pareto de fato uma vez que não são tidas como válidas.

A partir das observações acima discutidas, infere-se a hipótese 1 como verdadeira, uma vez que foi mostrado que a abordagem produziu soluções comparáveis aos dos

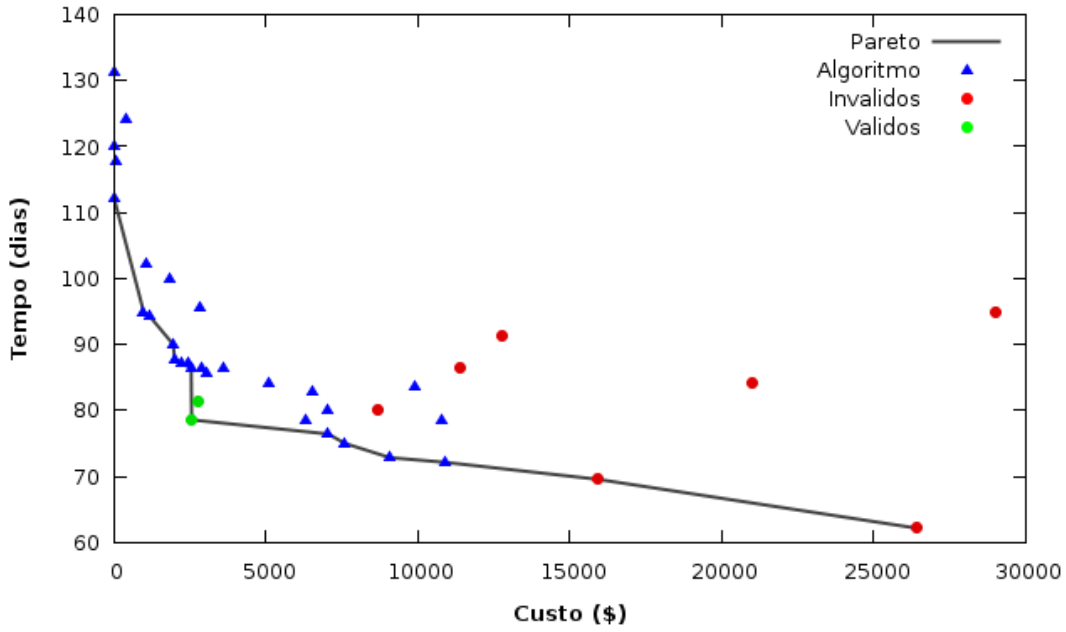

Figura 5.6. Viabilidade das soluções e a frente de pareto

especialistas, inclusive melhores no aspecto geral. A validação dessa hipótese é de extrema importância, pois mostra que a abordagem pode ser utilizada eficaz e eficientemente para obtenção de soluções para a alocação de equipes e o desenvolvimento de cronogramas.

Para verificar a capacidade da abordagem em apoiar o gerente de projetos para obtenção de soluções de maior qualidade, análise relativa à hipótese número 2, duas estratégia de interação entre especialista e algoritmo foram consideradas: utilização *a priori* e *a posteriori*[5]. Para analisar a primeira estratégia, foram confrontados resultados dos grupos experimental e de controle no intuito de verificar se o uso de soluções previamente geradas pela abordagem resultou em soluções melhores que sem utilizá-las. Já para a estratégia *a posteriori*, resultados obtidos por especialistas de ambos os grupos foram utilizados como soluções iniciais para o algoritmo, em vista de verificar se a abordagem seria capaz de obter melhoramentos às soluções dos especialistas.

O gráfico ilustrado na figura 5.7 diferencia as soluções dos grupos de controle e experimental, onde os dois pontos destacados representam as médias cartesianas das soluções de cada grupo. Considerando a média como dado de análise, é possível perceber que os valores relativos ao objetivo de tempo assemelharam-se para ambos os grupos, porém o grupo de controle conseguiu resultados melhores para o objetivo de custo, de forma que, no geral, o grupo de controle obteve melhores resultados. Além disso, duas das quatro soluções obtidas pelos especialistas do grupo de controle foram

[5]A hipótese 2 é verdadeira para o caso de pelo menos uma das duas estratégias de interação mostrar-se verdadeira.

soluções viáveis, enquanto os cinco especialistas do grupo experimental apresentaram soluções inviáveis, ou seja, com quebra de restrições. Os dois fatos mostram que o uso *a priori* da abordagem pode ajudar o gerente a obter soluções de melhor qualidade.

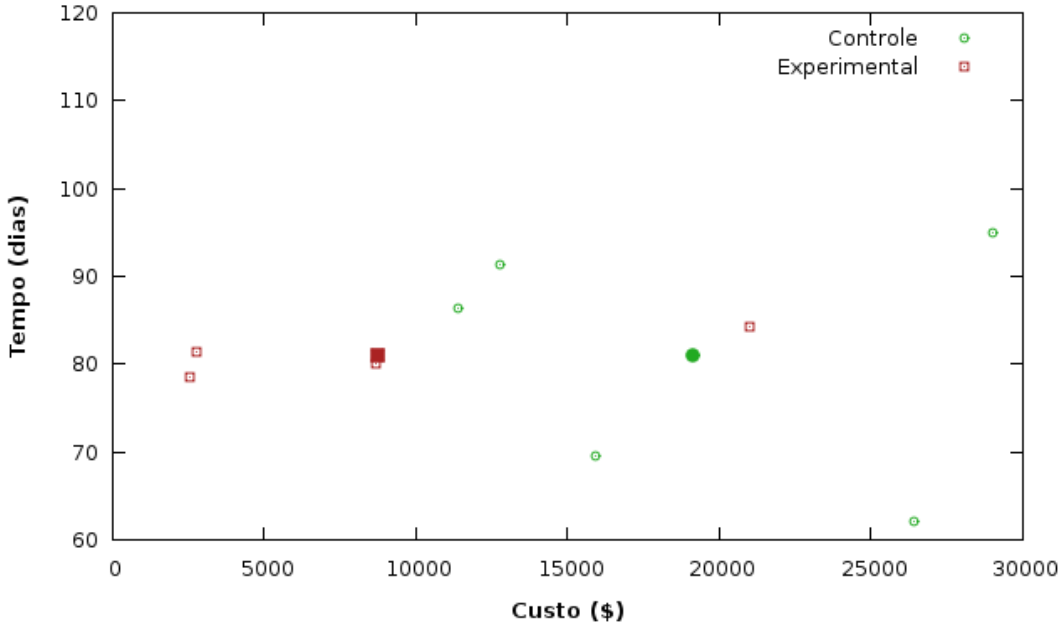

Figura 5.7. Uso *a priori*: grupo de controle × grupo experimental

Para análise da utilização *a posteriori*, três soluções dos grupo experimental e de controle foram aleatoriamente selecionadas e utilizadas como solução inicial para o algoritmo. O processo foi realizado duas vezes para cada solução. As figuras 5.8, 5.10 e 5.11 apresentam os resultados obtidos nos três casos.

O gráfico da figura 5.8 mostra a aplicação *a posteriori* da abordagem a uma solução obtida por um especialista, uma solução inválida e que não fez parte da frente de pareto ilustrada na figura 5.6. É clara a percepção de melhoria nos objetivos, além do algoritmo apresentar soluções viáveis em lugar da solução inviável utilizada, fatos que demonstram o ganho de qualidade nesse caso *a posteriori*. É interessante relevar que, embora a solução utilizada tenha sido uma solução inválida e que não fazia parte da frente de pareto já obtida, ela ajudou o algoritmo a obter soluções melhores que sem utilizá-la, conforme ilustra a figura 5.9.

As figuras 5.10 e 5.11 apresentam outras duas aplicações *a posteriori*. Diferentemente do solução anterior, as soluções desses dois casos fizeram parte da frente de pareto da 5.6, sendo a primeira uma solução inválida e a segunda uma solução válida. No primeiro gráfico, percebe-se a obtenção de uma diversidade de boas soluções, principalmente no que se refere à melhoria do objetivo de custo, objetivo para o qual a solução do especialista obteve um valor ruim. Nota-se também que as novas soluções

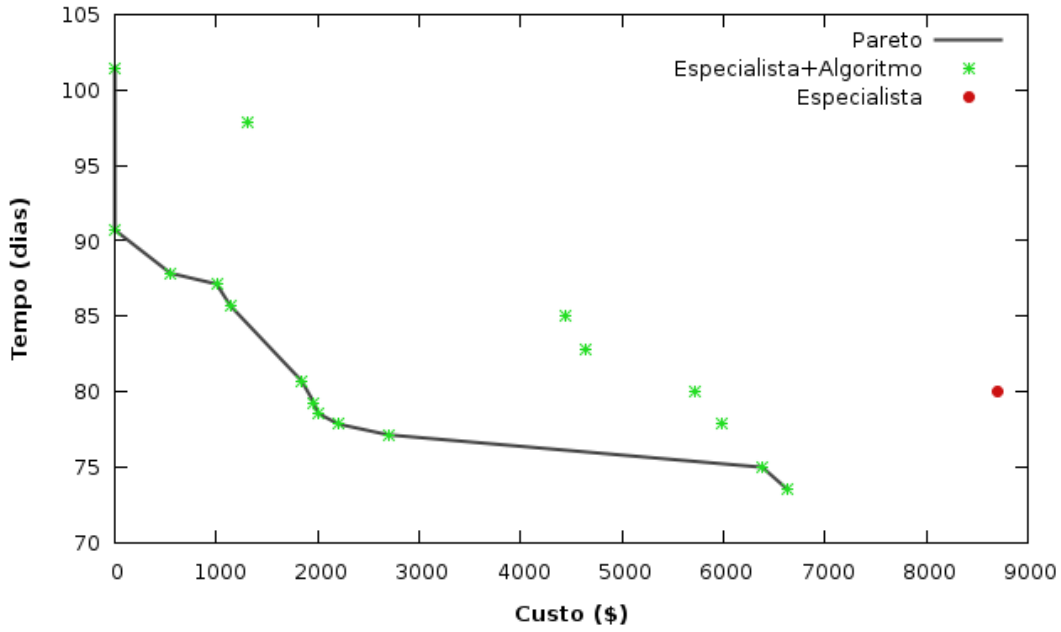

Figura 5.8. Uso *a posteriori* (solução 1)

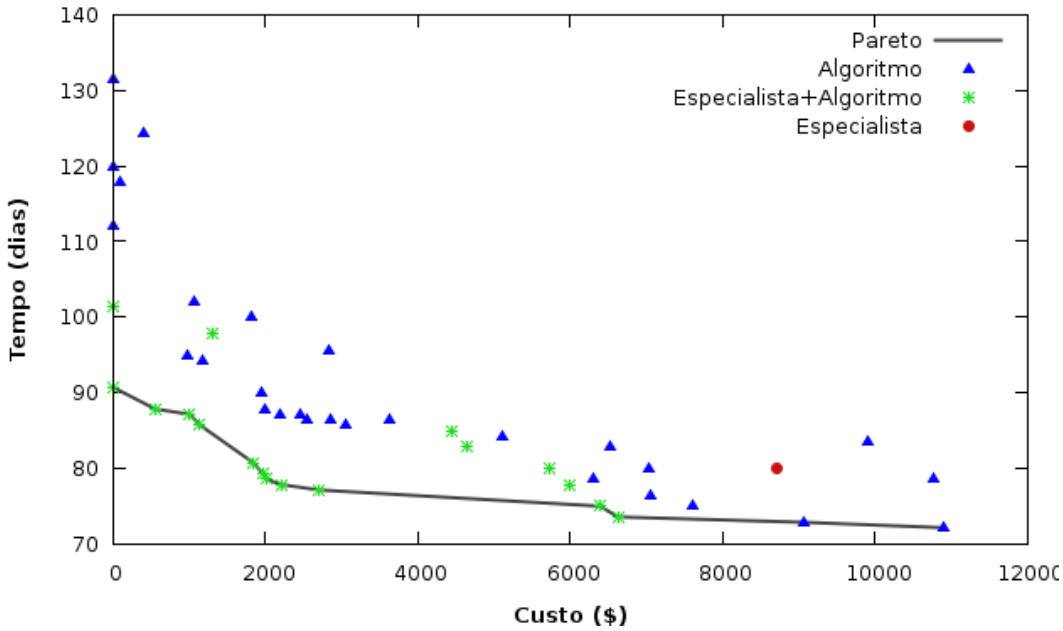

Figura 5.9. Uso *a posteriori* (solução 1): melhoria nos resultados do algoritmo

formaram uma frente de pareto que não incluiu a solução do especialista, o que demonstra a melhora na qualidade. Além disso, é importante ressaltar que mais uma vez o algoritmo obteve soluções viáveis a partir de uma solução inviável. Já no gráfico da figura 5.11, observa-se a obtenção de um número menor de soluções e também que a solução do especialista continuou a fazer parte da frente de pareto. Isso indica que foi utilizada uma solução de alta qualidade, o que pode ser reforçado comparando-a

com as demais soluções dos especialistas. Ainda que a solução utilizada tenha sido das melhores, o algoritmo foi capaz de trazer ganhos em relação à diversidade, uma vez que apresentou novos resultados com melhores valores para um dos objetivos analisados, tempo ou custo.

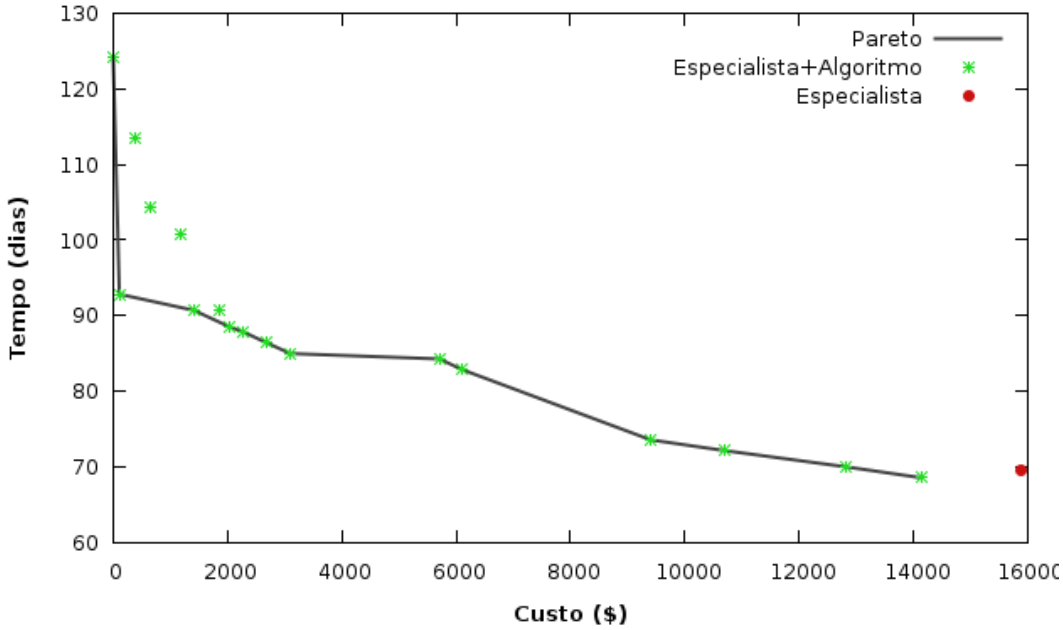

Figura 5.10. Uso *a posteriori* (solução 2)

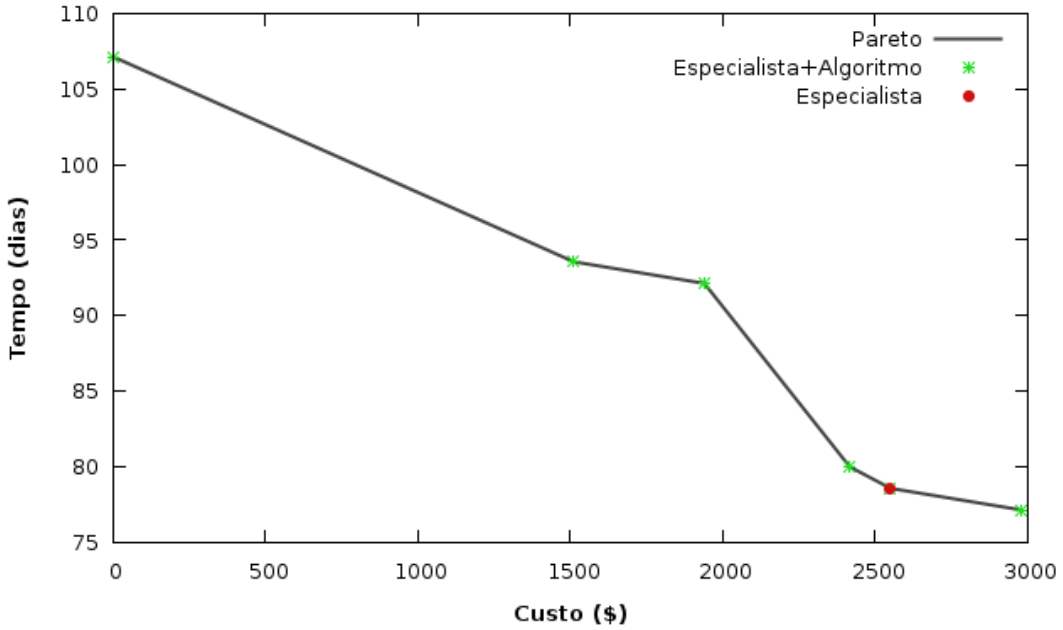

Figura 5.11. Uso *a posteriori* (solução 3)

A análise dos três casos de utilização *a posteriori* deixa claro o ganho de qua-

lidade obtido. Somando-se isso ao também demonstrado ganho com a utilização *a priori*, fica provada como verdadeira a hipótese 2 para ambas as estratégias, fato que demonstra que a abordagem proposta pode potencialmente auxiliar o gerente de projetos a obter melhores soluções de planejamento de cronogramas e alocação de equipes, mostrando-se, assim, como uma interessante ferramenta para melhoria do processo de desenvolvimento, objetivo principal deste trabalho.

Finalmente, a hipótese 3 também foi analisada, porém infelizmente a análise mostrou-se inconclusiva diante do fato das diferenças temporais entre os especialistas não terem seguindo padrões que pudessem ser atribuídos ao grupo ao qual pertenciam, seja experimental ou de controle. Assim, não se pôde concluir a hipótese como verdadeira, uma vez que não foi possível demonstrar ganho na produtividade ao utilizar a abordagem. Porém, vale salientar que também não foi demonstrada qualquer perda de produtividade e que, diante da escalabilidade da abordagem, ainda pode-se acreditar na possibilidade de ganhos de produtividade em projetos maiores e de maior complexidade, fato que precisa ser verificado a partir de novos experimentos.

5.3.3 Análise de Validez e Conclusão Geral

Antes de qualquer conclusão final acerca do experimento, é preciso analisar a validez dos resultados obtidos. Wohlin et al. (2000) enumera quatro tipos de ameaças à validez dos resultados, os quais precisam ser prevenidos durante o planejamento do experimento e verificados ao seu final. Os próximos parágrafos discutem tais ameaças e as analisam no contexto do experimento, de forma a demonstrar a validez dos resultados obtidos.

Inicialmente, a *validez da conclusão* é analisada em busca de garantir como verdadeira as pressupostas relações entre as variáveis independentes, aquelas que são controladas durante do experimento, e as variáveis dependentes, aquelas que constituem os resultados obtidos e sobre as quais não se tem controle. Neste experimento, o projeto a ser planejando é um exemplo de variável independente, enquanto os valores de tempo e custo do cronograma fazem parte das variáveis dependentes. Entre as ameaças enumeradas por Wohlin et al. (2000), como a garantia de heterogeneidade aleatória dos participantes e a confiabilidade das medições, o que mais merece ser discutido é o volume estatístico de dados, uma vez que a quantidade de participantes pode ser questionada. Embora um universo com 10 participantes não represente uma grande massa de dados para aplicação de técnicas estatísticas mais refinadas, esse número foi considerado satisfatório, principalmente pelo fato dos resultados não terem apresentado situações de fugas ao padrão das soluções.

Ameaças à *validez interna* são aquelas que podem gerar questionamentos acerca da relação de causa e efeito sugerida no experimento, ou seja, dúvidas no sentido dos

resultados obtidos terem sido causados por fatores desconsiderados ou que tenham fugido ao controle do experimento, como a interação indevida entre participantes ou uma distribuição de grupos mal feita. No experimento, que foi conduzido em duas sessões, foi garantido não ter havido a interação prejudicial entre os participantes e qualquer seleção interna de participantes foi feita de forma aleatória, além de terem sido observadas todas às ameaças enumeradas por Wohlin et al. (2000).

A *validez da construção* está relacionada à relação entre teoria e observação, isto é, à generalização daquilo que foi concretamente obtido no experimento aos conhecimentos teóricos por trás dele. Neste aspecto, pode ser discutido sobre o projeto utilizado para o planejamento. Uma vez inviável a utilização de um projeto de grande escala e de maior complexidade, foi utilizado um projeto fictício de tamanho limitado. Porém, dado que a abordagem proposta se baseia no conceito de automatização e para isso utiliza técnicas de grande escalabilidade, a utilização de um projeto simplificado não tende a prejudicar a generalização dos resultados. Ao contrário, quanto maior a complexidade da tarefa, mais sujeita a erros e perda de qualidade é a abordagem *ad hoc* tradicional, utilizada pelos especialistas. Outra questão relativa à validez da construção, essa sempre passível de discussão, é a análise conclusiva dos resultados. Como recomendado, as conclusões tomadas neste experimento foram sempre baseada em dados estatísticos quantitativos, evitando-se ao máximo o julgamento subjetivo na análise das hipóteses apresentadas.

Finalmente, a *validez da externa* refere-se à generalização dos resultados a situações práticas, considerando três tipos de ameaças: de tempo, de lugar e de pessoal. Em relação ao primeiro, a questão temporal não afeta o problema analisado. Em relação ao segundo, o ambiente foi reproduzido conforme as situações práticas da indústria, por exemplo com a utilização de uma ferramenta altamente aceita na indústria. Finalmente, em relação aos participantes, vale ressaltar o que foi discutido sobre o conjunto ideal formado por gerentes de projetos e o conjunto utilizado, constituído parte por especialistas com experiência e conhecimento em gerência de projetos mas também por estudantes com pouca ou nenhuma experiência na área. Como mencionado, diante da qualificação daqueles e pelo fato de seus resultados terem entoado com os destes, foi considerada satisfatória a representatividade da população alvo.

Analisada a validez dos resultados obtidos no experimento, finalmente pode-se inferir como fato tudo o que foi demonstrado como verdadeiro nas hipóteses analisadas. Dessa forma, conclui-se, por fim, que este experimento agregou grande valor ao trabalho, demonstrando de forma fundamentada a eficácia da abordagem em atingir os objetivos propostos, no que se refere à qualidade dos resultados e aos ganhos gerenciais obtidos com a sua utilização.

5.4 Expandindo o Estudo: Análises de Sensibilidade

Mostrada a aplicabilidade da abordagem e comprovada a qualidade de seus resultados, respectivamente no estudo de caso da seção 5.2 e no experimento empírico da seção 5.3, esta seção apresenta estudos aprofundando questões acerca da abordagem em si e do problema por ela abordado, a alocação de equipes e o desenvolvimento de cronogramas em projetos de software. Inicialmente, investigações sobre a abordagem são apresentadas nas subseções 5.4.1, a qual discute resultados no ponto de vista dos múltiplos objetivos propostos na formulação do problema, e 5.4.2, que analisa o planejamento multiprojetos. Posteriormente, são realizadas análises de sensibilidade[6] relativas a três parâmetros do problema: a disponibilidade de recursos, na subseção 5.4.3, a comunicação interpessoal, na subseção 5.4.4, e o aprendizado, na subseção 5.4.5.

5.4.1 Análise de Múltiplos Objetivos

Embora a formulação do problema apresentada na seção 3.3 apresente quatro objetivos a serem otimizados, os resultados expressos nas seções anteriores deste capítulo não apresentaram análises para todos os quatro objetivos, seja por alguns não se aplicarem às análises ou simplesmente por alguns terem sido ocultados por questão de clareza. Em vista de melhor explorar essa questão, foram realizados testes com o objetivo de mostrar e discutir a análise simultânea dos quatro objetivos da formulação, testes esses que são apresentados nesta subseção.

A abordagem foi aplicada a um projeto semelhante ao utilizado no experimento da seção 5.3, com algumas diferenças como a presença de tarefas com *deadline*, em vista de possibilitar que o objetivo de minimizar atrasos também pudesse ser considerado. Outra mudança, esta com o objetivo de investigar um outro aspecto também pouco analisado até então, foi em relação aos valores de proficiência e experiência dos recursos, que não se limitaram somente ao valor "normal".

A análise de resultados de múltiplos objetivos é uma questão bem discutida no meio acadêmico em geral, principalmente em relação à apresentação das soluções. A apresentação de dois objetivos é trivial, porém, quanto maior a quantidade de objetivos, maior a dificuldade de ilustrá-los com clareza. Deb (2001) discute essa questão, apresentando alternativas de representação. Uma delas, apresentada na figura 5.12, é o chamado *scatter plot matrix method*, que consiste em dividir os objetivos dois a dois, apresentando $\frac{n \times (n-1)}{2}$ gráficos bidimensionais.

[6] Análises de sensibilidade consistem em investigar aspectos do problema em questão, geralmente com o objetivo de estudar influências de parâmetros específicos na resolução do problema. Como afirma Harman (2007b), o uso de SBSE facilita a realização de análises de sensibilidade.

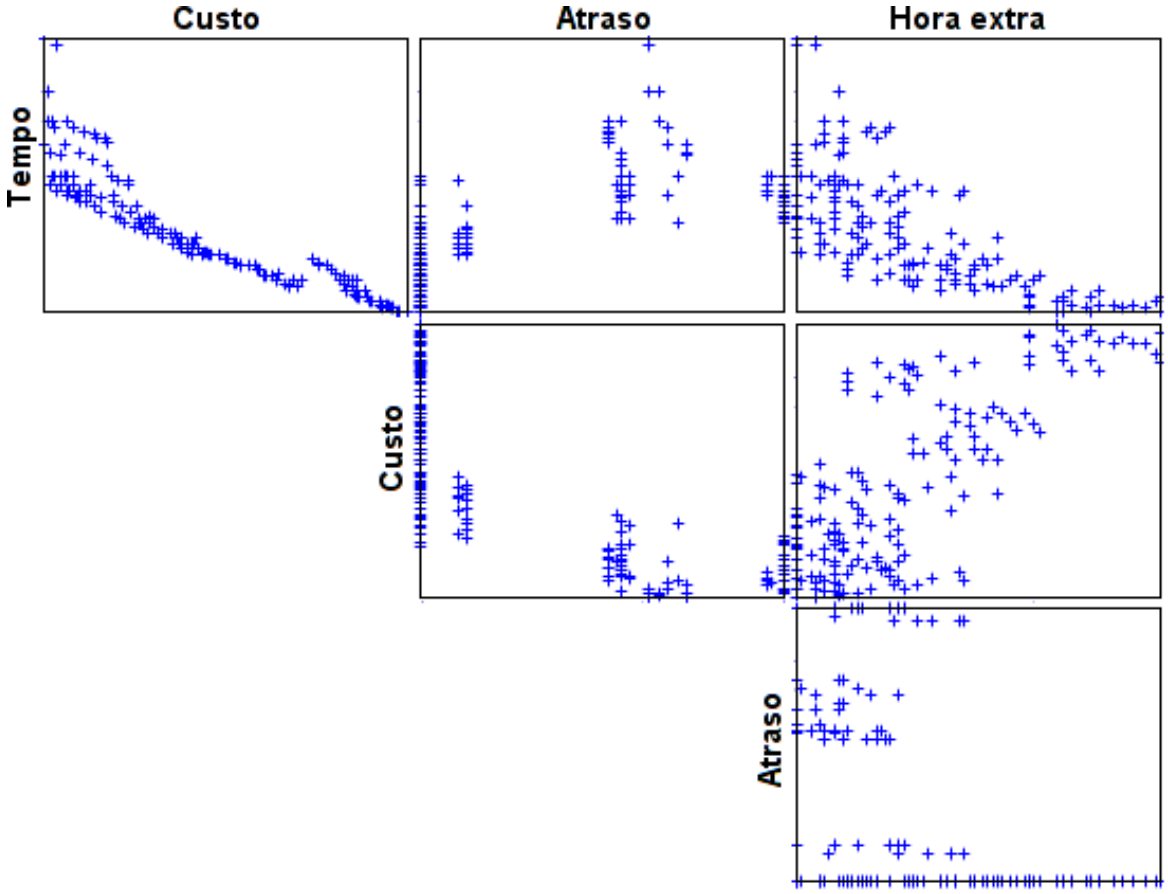

Figura 5.12. Matriz de gráficos bidimensionais

Inicialmente, é possível perceber que a utilização de mais objetivos aumenta a quantidade de soluções da frente de pareto. Isso ocorre porque quanto maior a quantidade de objetivos mais difícil é uma solução dominar outra, de forma que aumenta a quantidade de soluções não dominadas, as quais formam a frente de pareto resultante.

Em relação aos objetivos do problema, os gráficos ilustram certos comportamentos esperados. A gráfico tempo × custo, já analisado em situações anteriores, mantém a relação de inversão entre objetivos, onde menor tempo implica em maior custo, e vice versa. Esse comportamento também pode ser observado, embora de forma menos rígida, no gráfico tempo × hora extra. Já na comparação dos objetivos de tempo e atraso, existe uma relação, porém não sempre verdadeira, de que os menores tempos implicam em menos atrasos. Em resumo, os gráficos da primeira linha, dos três objetivos em função do tempo, mostram relação em todos os objetivos, sendo com o objetivo de custo a relação mais evidente.

A análise do gráfico custo × atraso mostra que as soluções de menores custos apresentaram um certo nível de atraso, sendo observadas soluções com atraso nulo a partir de um certo custo mediano. Como esperado, o gráfico custo × hora extra

mostra uma relação de proporcionalidade direta entre os dois objetivos, porém não muito rígida pelo fato da existência de recursos do tipo contratado, os quais também causam influência no custo do projeto. Finalmente, a relação entre atrasos e horas extras aparenta ser a mais fraca dentre todas, mas ainda pode ser identificada uma ligação entre a quantidade hora extras com um menor atraso nas tarefas.

Uma óbvia desvantagem em utilizar gráficos bidimensionais é a quantidade de gráficos necessários, que cresce de forma quadrática em função da quantidade de objetivos, além de que gráficos bidimensionais podem dificultar o entendimento global de todos os objetivos. Assim, outras abordagens de análise podem ser utilizadas, como a utilização de gráficos em três dimensões. A figura 5.13 apresenta um gráfico tridimensional de custo × tempo × atraso. Para maior clareza, a saturação de cores varia de acordo com o valor de custo. O mesmo gráfico é mapeado numa versão de bidimensional (tempo × atraso), onde a saturação de cor representa o terceiro objetivo (custo), na figura 5.14.

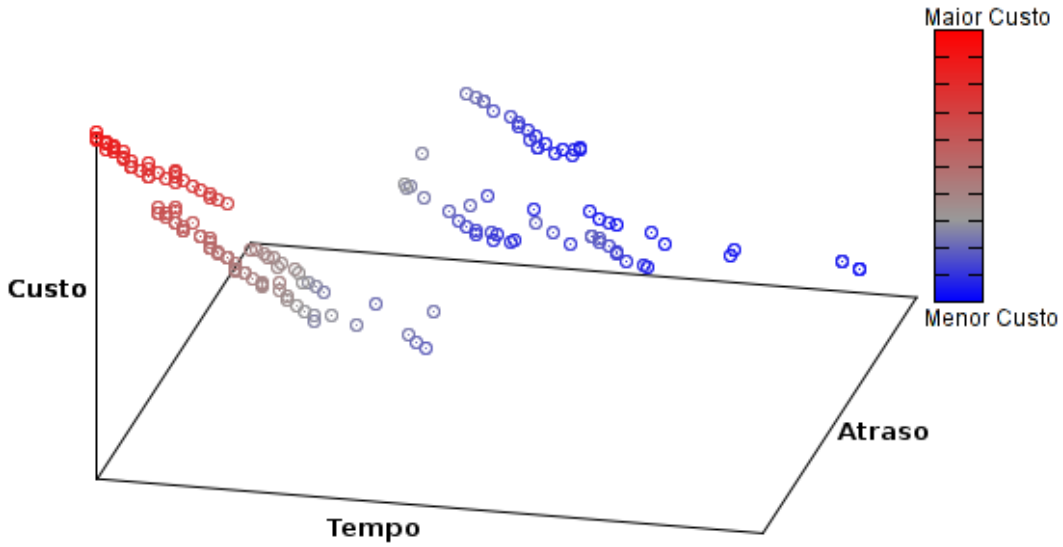

Figura 5.13. Gráfico tridimensional (custo × tempo × atraso)

A análise dos gráficos das figuras 5.13 e 5.14 mostram bem a relação mútua entre os três objetivos em questão. É possível perceber que, como esperado, as soluções de menor tempo e menor atraso são as que apresentam maior custo, o qual decresce à medida que aumentam tempo e atraso.

5.4.2 Planejamento Multiprojetos

A subseção 3.2.11 discute sobre planejamentos de múltiplos projetos utilizando a abordagem proposta neste trabalho. Como discutido, o planejamento multiprojetos consiste em uma situação especial em que recursos são compartilhados por tarefas de projetos

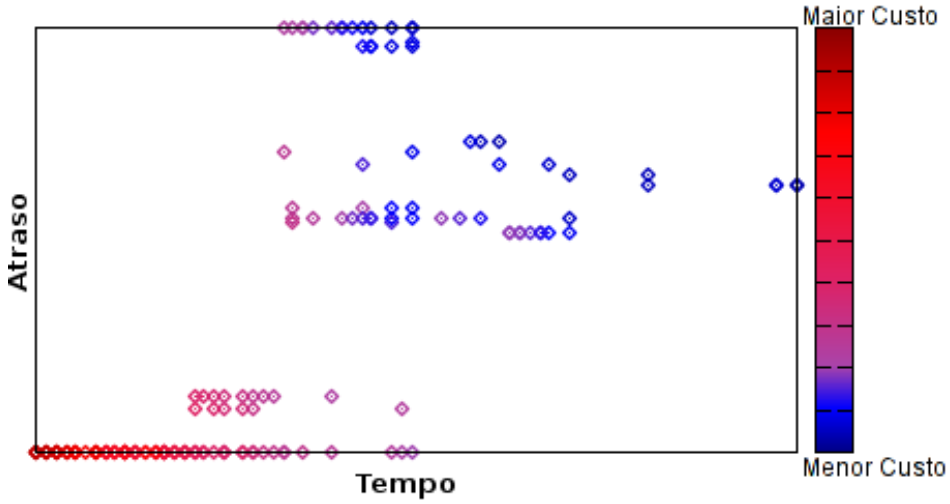

Figura 5.14. Visão tridimensional mapeada em duas dimensões

diferentes, onde os objetivos de cada projetos precisam ser considerados e suas fronteiras delimitadas. Esta subseção apresenta resultados de testes realizados com o intuito de comprovar a aplicabilidade da abordagem ao planejamento multiprojetos.

Para isso, a abordagem proposta foi utilizada para o planejamento de dois projetos simultâneos. Novamente, os projetos seguiram o mesmo padrão do projeto utilizado no experimento apresentado na seção 5.3, mais especificamente utilizando esse mesmo projeto acrescido de um novo projeto baseado em um processo de desenvolvimento semelhante, totalizando 54 tarefas (27 do primeiro projeto e 17 do segundo), que foram alocadas aos mesmos 18 recursos daquele projeto. Em relação aos objetivos, cada projeto teve separado seus objetivo de tempo e atraso, mantendo os objetivos de custo e horas extras independentes de projetos, uma vez que esses são aspectos relativos a toda a organização e não a cada projeto específico.

As figuras 5.15 e 5.16 apresentam as soluções apresentadas pelo algoritmo. Uma vez que as tarefas não continham *deadline*, o objetivo de atraso é sempre nulo, por isso não sendo apresentado, enquanto o objetivo de hora extra foi suprimido para facilitação da análise. Assim, são apresentados os objetivos de tempo para os dois projetos e de custo total. Os gráficos ilustrados mostram uma diversidade de soluções que otimizam cada um dos objetivos, sempre com comportamentos esperados. Por exemplo, é possível perceber soluções de custo razoável com bom valor de tempo para um dos dois projetos, ou ainda soluções que minimizem o tempo de ambos, porém apresentando maior custo. Ainda se observa que os menores custos foram obtidos com soluções que apresentaram maiores tempos para ambos os projetos.

Por efeito de clareza, não são apresentados detalhes de cada solução específica, uma vez elas seguiram o mesmo padrão de soluções já discutidas e ressaltando que o objetivo

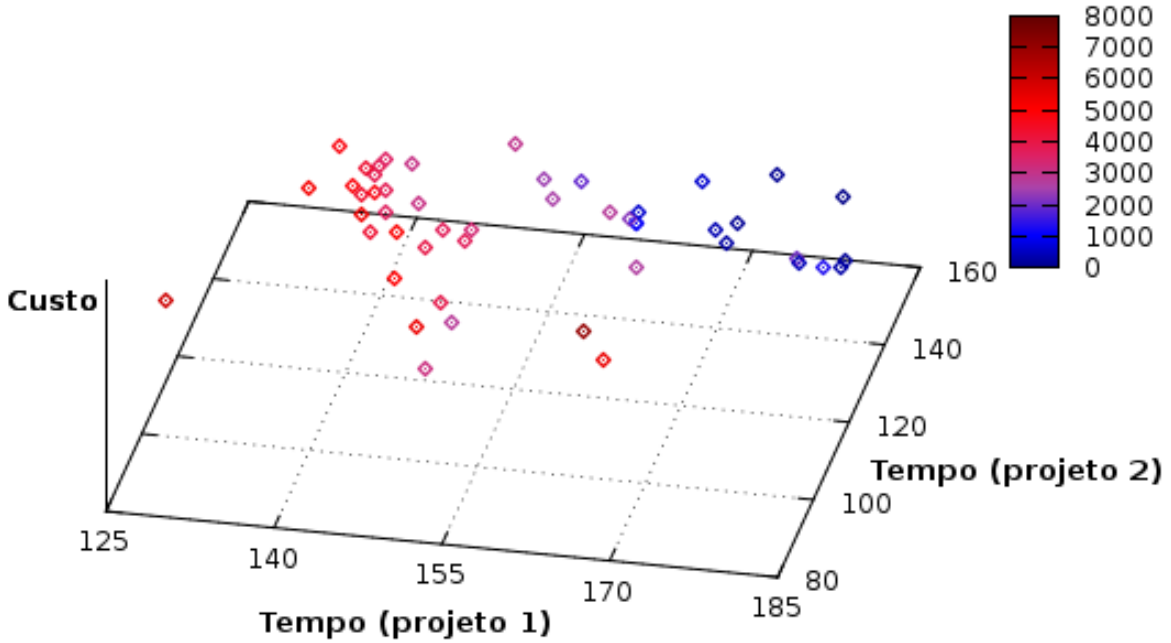

Figura 5.15. Relação custo × tempo × tempo

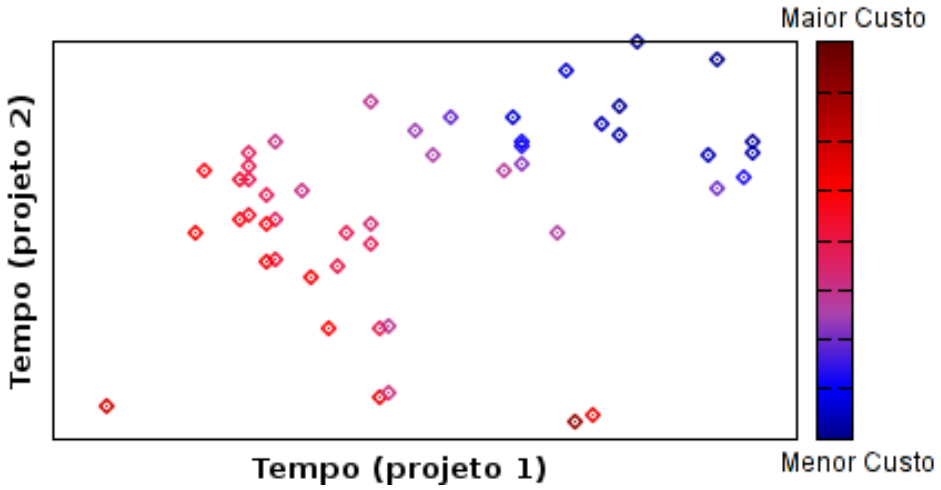

Figura 5.16. Visão mapeada em duas dimensões do custo em função dos tempos

deste cenário é demonstrar a viabilidade do planejamento multiprojetos utilizando a abordagem proposta, o que fica claro com os resultados apresentados.

5.4.3 A Influência da Quantidade de Recursos

A primeira análise de sensibilidade do problema é relativa à investigação da influência da quantidade de recursos sobre o cronograma do projeto, mais especificamente em sua duração. Utilizando um projeto semelhante aos já supracitados e variando a quantidade de recursos disponíveis, os resultados obtidos foram comparados em relação ao objetivo

de tempo. Para evitar peturbações no resultado, foram considerados somente recursos do tipo empregado e sem a possibilidade de realização de hora extra, de forma a manter-se fixo o custo do projeto e tornar o objetivo de tempo como o único a ser realmente otimizado pelo algoritmo. A figura 5.17 apresenta o gráfico obtido com 75 execuções do algoritmo para cinco diferentes configurações de recursos, totalizando 15 execuções para cada configuração.

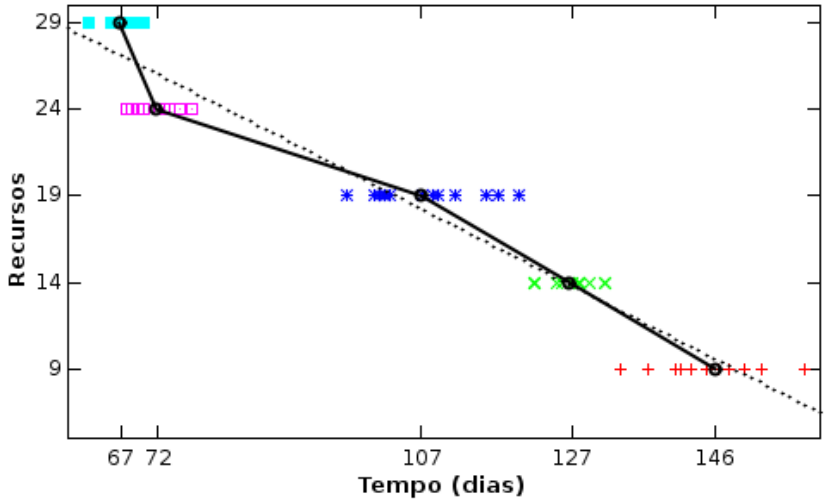

Figura 5.17. Variação de tempo em função da quantidade de recursos

Obviamente, o gráfico mostra que o tempo aumenta conforme decresce a quantidade de recursos disponíveis. Mais do que isso, ele mostra a curva resultante dessa relação (linha preta) e uma reta do comportamento geral aproximado (reta tracejada). O coeficiente angular dessa reta indica uma razão de 20% e 25% entre tempo e quantidade de recursos. É claro que há um momento em que essa razão sofre distorções, superior e inferiormente, uma vez que a quantidade crescente de recursos que executam uma tarefa é finita e também dado que existe uma quantidade mínima de recursos para execução de todas as tarefas. Vale ressaltar ainda não foi considerado o *overhead* de comunicação interpessoal, fator que também influencia diretamente esta relação analisada e que é analisado na próxima subseção.

5.4.4 A Influência da Comunicação

É sabido que a comunicação interpessoal causa distorções na produtividade das equipes em projetos de desenvolvimento de software. Porém, pouco se sabe sobe como e quão negativa é essa influência. Conforme discutido na subseção 3.2.9, este trabalho formulou o *overhead* de comunicação de acordo com a conhecida lei de Brooks (1995), utilizando uma equação de ordem quadrática que varia em função de um certo coeficiente de

comunicação, que representa a razão entre o tempo gasto na comunicação e o tempo realmente utilizado na execução das tarefas. Esta subseção apresenta uma análise de sensibilidade relativa à influência desse coeficiente de comunicação na alocação das equipe e no cronograma de um projeto. Uma vez que muitas das organizações de desenvolvimento de software podem não possuir dados suficientes para a escolha do coeficiente de comunicação, este estudo pode ser considerado um uma interessante referência de ajuda nesse sentido.

A figura 5.18 ilustra o gráfico resultante de 50 execuções do algoritmo, 10 para cada valor do coeficiente de comunicação, o qual variou de 0% a 20%. Como esperado, quanto maior o coeficiente utilizado, maior são os valores de tempo e custo. Uma análise mais detalhada mostrou que isso é influenciado por dois fatores: a menor produtividade e também a alocação de equipes menores, o que, de fato, é também uma consequência da perda de produtividade em equipes maiores. A curva apresentada no gráfico não possui um padrão bem definido, mas pode ser usada para algumas análises, como a percepção de que o tempo é mais influenciado nos valores menores do coeficiente, enquanto para valores maiores a influência no custo é mais visível.

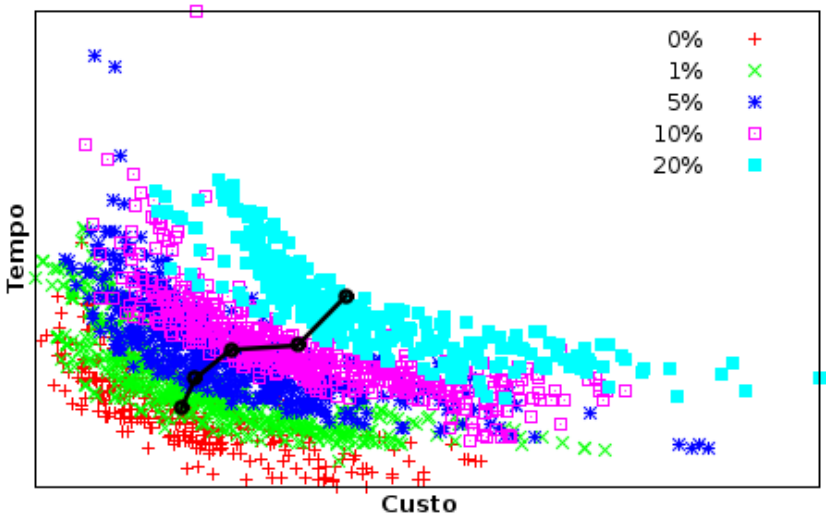

Figura 5.18. Tempo e custo em função da presença de comunicação interpessoal

5.4.5 A Influência do Aprendizado

Finalmente, a terceira e última análise de sensibilidade realizada investiga sobre a influência do aprendizado nos cronogramas de projetos. Semelhantemente à comunicação interpessoal, o modelo utilizado para formular a curva de aprendizado também depende de um fator de aprendizado, fator esse que representa a taxa de ganho na produtividade

durante a execução continuada de uma tarefa. Novamente, este estudo pode ser visto como uma boa referência para o entendimento do coeficiente de aprendizado.

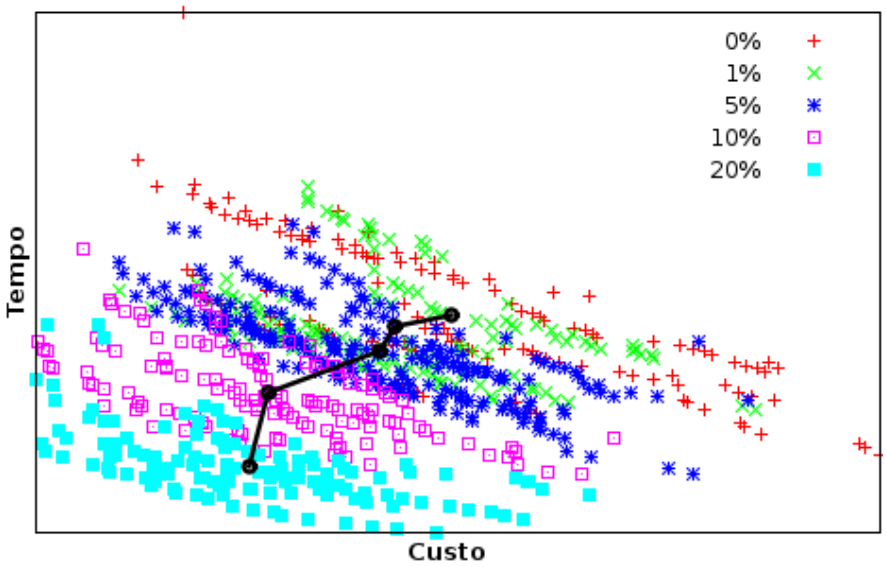

Figura 5.19. Tempo e custo em função do coeficiente de aprendizado

No estudo, foram analisados cinco valores entre 0% a 20% para o coeficiente de aprendizado. A figura 5.19 ilustra o gráfico resultante de 50 execuções do algoritmo, 10 para cada valor do coeficiente. Naturalmente, quanto maior o aprendizado melhor é a qualidade das soluções obtidas, tanto em relação ao tempo quanto ao custo, devido ao ganho de produtividade. A curva apresentada ainda mostra que a proporção de ganho é maior para coeficientes maiores, o que pode ser explicado devido às tarefas por vezes não terem duração considerável para acúmulos de aprendizado que resultem em grandes ganhos de produtividade.

Capítulo 6

Conclusão e Trabalhos Futuros

Este trabalho apresentou uma proposta de uma abordagem baseada em otimização computacional para o problema de alocação de equipes e desenvolvimento de cronogramas em projetos de software, problema que se enquadra no processo de desenvolvimento de cronograma da área de gerenciamento de tempo nas boas práticas de gestão de projetos. A abordagem apresentada, que se enquadra em um campo emergente da engenharia de software denominado *Search-Based Software Engineering* (SBSE), consiste em formular o problema como um problema de busca, visando a posterior aplicação de algoritmos de otimização. A formulação proposta considera vários aspectos condizentes com questões enfrentadas cotidianamente por gerentes de projetos de software, tais como habilidades individuais, experiência, horas extras, contratações e planejamentos multiprojetos. Diante da complexidade do problema abordado, a qual é formalmente analisada e comprovada no trabalho, um algoritmo aproximado foi desenvolvido com base na meta-heurística multiobjetivo NSGA-II.

Estudos empíricos foram conduzidos para validar e avaliar a abordagem proposta. Baseando-se em sólidas referências de experimentação em engenharia de software, foram apresentados um estudo de caso, que mostrou indícios da aplicabilidade da abordagem em situações reais, e um experimento supervisionado, o qual comprovou a eficácia da abordagem em obter resultados comparáveis e melhores com de especialistas da área. O experimento também demonstrou os ganhos que o uso da abordagem pode proporcionar à gerência de projetos de software. Finalmente, também foram apresentados testes aprofundando investigações acerca da abordagem em si e analisando sensibilidades do problema em relação à influência de parâmetros como a disponibilidade de recursos, a comunicação interpessoal e o aprendizado.

6.1 Trabalhos Futuros

Há uma gama de possibilidades para condução de estudos e trabalhos futuros. Inicialmente, espera-se investigar junto a gerentes de projetos novos aprimoramentos à formulação, tais como a possibilidade de adicionar características dinâmicas à abordagem, extendo o planejamento estático para planejamento e controle dinâmicos. Novos aspectos também podem ser incorporados à formulação, como, por exemplo, a realização de treinamentos, onde um treinamento pode ser considerado como uma tarefa diferenciada na qual os recursos aprimorem suas habilidades ou adquiram outras. De fato, o próprio modelo apresentado na seção 3.4 já aborda a questão dos treinamentos.

Em vista de melhorar a qualidade das soluções, aprimoramentos ao algoritmo podem ser conduzidos, como a utilização de outras técnicas de otimização e a criação de novas heurísticas a serem introduzidas no algoritmo. Também pode ser melhor investigada a eficiência e eficácia do algoritmo proposto em relação a outras técnicas de otimização, em especial a técnicas exatas.

Finalmente, a aplicação da abordagem a projetos reais para a condução de novos estudos de caso também é uma importante questão a ser realizada em vista de se analisar o comportamento da abordagem em diferentes cenários.

Referências Bibliográficas

Alba, E. e Chicano, J. F. (2007). Software project management with gas. *Information Science*, 177(11):2380–2401.

Ali, S.; Briand, L. C.; Hemmati, H. e Panesar-Walawege, R. K. (2009). A systematic review of the application and empirical investigation of search-based test-case generation. *IEEE Transactions on Software Engineering*, 99(PrePrints).

Amoui, M.; Mirarab, S.; Ansari, S. e Lucas, C. (2006). A genetic algorithm approach to design evolution using design pattern transformation. *International Journal of Information Technology and Intelligent Computing*, 1(2):235–244.

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

Badiru, A. (1992). Computational survey of univariate and multivariate learning curve models. *IEEE Transactions on Engineering Management*, 39(2).

Barreto, A.; Barros, M. d. O. e Werner, C. M. L. (2008). Staffing a software project: A constraint satisfaction and optimization-based approach. *Computers & Operations Research*, 35(10):3073–3089.

Basili, V. R.; Shull, F. e Lanubile, F. (2000). Using experiments to build a body of knowledge. In *Proceedings of the Third International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pp. 265–282. Springer.

Baudry, B.; Fleurey, F.; Jezequel, J.-M. e Traon, Y. L. (2005). Automatic test case optimization: A bacteriologic algorithm. *IEEE Software*, 22(2):76–82.

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

Boehm, B. W.; Clark; Horowitz; Brown; Reifer; Chulani; Madachy, R. e Steece, B. (2000). *Software Cost Estimation with Cocomo II with Cdrom*. Prentice Hall.

Bowman, M.; C.Briand, L. e Labiche, Y. (2007). Multi-objective genetic algorithm to support class responsibility assignment. In *Proceedings of IEEE International Conference on Software Maintenance*, pp. 124–133. IEEE Computer Society.

Brooks, F. P. (1995). *The mythical man-month : essays on software engineering*. Addison-Wesley, 2a edição.

Brucker, P.; Drexl, A.; Möhring, R.; Neumann, K. e Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41.

Burke, E. K. e Kendall, G., editores (2005). *Search Methodologies. Introductory Tutorials in Optimization and Decision Support Techniques*. Publisher.

Catal, C. e Diri, B. (2009). Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences*, 179(8):1040–1058.

Chang, C. K.; Jiang, H.-y.; Di, Y.; Zhu, D. e Ge, Y. (2008). Time-line based model for software project scheduling with genetic algorithms. *Information and Software Technology*, 50(11):1142–1154.

Cho, S.-B. e Ray, T. S. (1995). An evolutionary approach to program transformation and synthesis. *International Journal of Software Engineering and Knowledge Engineering*, 5(2):179–192.

Clarke, J.; Dolado, J. J.; Harman, M.; Jones, B.; Lumkin, M.; Mitchell, B.; Mancoridis, S.; Rees, K. e Roper, M. (2003). Reformulating software engineering as a search problem. *IEE Proceedings - Software*, 150(3):161–175.

Colak, S.; Agarwal, A. e Erenguc, S. (2006). *Perspectives in Modern Project Scheduling*, volume 92 of *International Series in Operations Research & Management Science*, chapter Resource Constrained Project Scheduling: A Hybrid Neural Approach, pp. 297–318. Springer.

Colares, F.; Souza, J.; Carmo, R.; Padua, C. e Mateus, G. R. (2009). A new approach to the software release planning. In *Proceedings of XXIII Brazilian Symposium on Software Engineering*, pp. 207–215. IEEE Computer Society.

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons.

Deb, K.; Pratap, A.; Agarwal, S. e Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transaction on Evolutionary Computation*, 6(2):181–197.

Desnos, N.; Huchard, M.; Tremblay, G.; Urtado, C. e Vauttier, S. (2008). Search-based many-to-one component substitution. *Journal of Software Maintenance and Evolution: Research and Practice (Special Issue Search Based Software Engineering)*, 20(5):321–344.

Di Penta, M.; Harman, M.; Antoniol, G. e Qureshi, F. (2007). The effect of communication overhead on software maintenance project staffing: a search-based approach. In *Proceedings of the 23rd IEEE International Conference on Software Maintenance*, pp. 315–324.

Dolado, J. J. (2000). A validation of the component-based method for software size estimation. *IEEE Transactions on Software Engineering*, 26(10):1006–1021.

Dolado, J. J. (2001). On the problem of the software cost function. *Information and Software Technology*, 43(1):61–72.

Dolado, J. J. e Fernandez, L. (1998). Genetic programming, neural networks and linear regression in software project estimation. In *Proceedings of International Conference on Software Process Improvement, Research, Education and Training*, pp. 157–171. British Computer Society.

Dong, F.; Li, M.; Zhao, Y.; Li, J. e Yang, Y. (2008). Software multi-project resource scheduling: A comparative analysis. In *Making Globally Distributed Software Development a Success Story (Proceedings of International Conference on Software Process)*, volume 5007, pp. 63–75. Springer.

Doval, D.; Mancoridis, S. e Mitchell, B. S. (1999). Automatic clustering of software systems using a genetic algorithm. In *Proceedings of International Conference on Software Tools and Engineering Practice*, pp. 73–81. IEEE Computer Society.

Durillo, J. J.; Nebro, A. J.; Luna, F.; Dorronsoro, B. e Alba, E. (2006). jMetal: A Java Framework for Developing Multi-Objective Optimization Metaheuristics. Technical Report ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos.

Engwall, M. e Jerbrant, A. (2003). The resource allocation syndrome: the prime challenge of multi-project management? *International Journal of Project Management*, 21(6):403–409.

Feather, M. S.; Cornford, S. L.; Kiper, J. D. e Menzies, T. (2006). Experiences using visualization techniques to present requirements, risks to them, and options for risk mitigation. In *Proceedings of the International Workshop on Requirements Engineering Visualization*. IEEE Computer Society.

Fleming, Q. W. e Koppelman, J. M. (2006). *Earned Value Project Management*. Project Management Institute, 3a edição.

Glass, R. L. (1994). The software-research crisis. *IEEE Software*, 11(6):42–47.

GNOME Project (2010). Gnome planner. `http://live.gnome.org/Planner`.

Goldratt, E. M. e Cox, J. (2004). *The Goal: a process of ongoing improvement*. Gower Publishing, 3a edição.

Golomb, S. W. e Baumert, L. D. (1965). Backtrack programming. *Journal of the ACM (JACM)*, 12(4):516–524.

Grunske, L. (2006). Identifying "good"architectural design alternatives with multi-objective optimization strategies. In *Proceedings of the 28th International Conference on Software Engineering*, pp. 849–852. ACM.

Harman, M. (2007a). 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. (2007b). The current state and future of search based software engineering. In *Proceedings of Future of Software Engineering 2007*, pp. 342–357. IEEE Computer Society.

Harman, M. (2010). *Fundamental Approaches to Software Engineering*, volume 6013, chapter Why the Virtual Nature of Software Makes It Ideal for Search Based Optimization, pp. 1–12. Springer.

Harman, M. e Jones, B. F. (2001). Search-based software engineering. *Information & Software Technology*, 43(14):833–839.

Harman, M.; Krinke, J.; Ren, J. e Yoo, S. (2009). Search based data sensitivity analysis applied to requirement engineering. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pp. 1681–1688. ACM.

Hartmann, J. e Robson, D. (1989). Revalidation during the software maintenance phase. In *Proceedings of the 1989 Conference on Software Maintenance*, pp. 70–80.

Heričko, M.; Živkovič, A. e Rozman, I. (2008). An approach to optimizing software development team size. *Information Processing Letter*, 108(3):101–106.

Herroelen, W.; De Reyck, B. e Demeulemeester, E. (1998). Resource-constrained project scheduling: a survey of recent developments. *Computers and Operations Research*, 25(4):279–302.

Hoste, K. e Eeckhout, L. (2008). Cole: Compiler optimization level exploration. In *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, pp. 165–174. ACM.

ISO (2003). *Quality management systems - Guidelines for quality management in projects.* International Organization for Standardization.

Jedrzejowicz, P. e Ratajczak-Ropel, E. (2007). Agent-based approach to solving the resource constrained project scheduling problem. In *Proceedings of the 8th international conference on Adaptive and Natural Computing Algorithms*, pp. 480–487. Springer.

Jones, B.; Sthamer, H.-H. e Eyres, D. (1996). Automatic structural testing using genetic algorithms. *Software Engineering Journal*, 11(5):299–306.

Jones, C. (1994). *Assessment and control of software risks.* Yourdon Press.

Jones, C. (1997). *Applied software measurement: assuring productivity and quality.* McGraw-Hill, 2a edição.

Juristo, N. e Moreno, A. M. (2001). *Basics of Software Engineering Experimentation.* Springer.

Katz, G. e Peled, D. (2010). Code mutation in verification and automatic code correction. In *Proceedings of 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Theory and Practice of Software*, volume 6015, pp. 435–450. Springer.

Kerzner, H. (2009). *Project Management: A Systems Approach to Planning, Scheduling, and Controlling.* John Wiley & Sons, 10a edição.

Kolisch, R. e Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23–37.

Kumar, V. (1992). Algorithms for constraint-satisfaction problems: a survey. *AI Maganize*, 13(1):32–44.

Lee, B. e Miller, J. (2004). Multi-project management in software engineering using simulation modelling. *Software Quality Control*, 12(1):59–82.

Li, H.; Love, P. E. D. e Drew, D. S. (2000). Effectos of overtiime work and additional resources on project cost and quality. *Engineering Construction and Architecural Management*, 7(3):211–220.

Li, Z.; Harman, M. e Hierons, R. M. (2007). Search algorithms for regression test case prioritization. *IEEE Transactions on Software Engineering*, 33(4):225–237.

Luenberger, D. e Ye, Y. (2008). *Linear and Nonlinear Programming*. Springer, 3a edição.

Lutz, R. (2001). Evolving good hierarchical decompositions of complex systems. *Journal of Systems Architecture*, 47(7):613–634.

McMinn, P. (2004). Search-based software test data generation: a survey. *Software Testing, Verification and Reliability*, 14(2):105–156.

Microsoft Corporation (2007-2010). Microsoft office project 2007. `http://office.microsoft.com/project`.

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

Mitchell, B. S. e Mancoridis, S. (2006). On the automatic modularization of software systems using the bunch tool. *IEEE Transactions on Software Engineering*, 32(3):193–208.

Ngo-The, A. e Ruhe, G. (2009). Optimized resource allocation for software release planning. *IEEE Transactions on Software Engineering*, 35(1):109–123.

Nishikitani, M.; Nakao, M. e Karita, K. (2005). Influence of overtime work, sleep duration, and perceived job characteristics on the physical and mental status of software engineers. *Industrial Health*, 43(4):623–629.

O'Keeffe, M. e Cinnéide, M. Ó. (2008). Search-based refactoring: An empirical study. *Journal of Software Maintenance and Evolution: Research and Practice (Special Issue Search Based Software Engineering)*, 20(5):345–364.

Oracle Corporation (2010). Primavera software. `http://www.oracle.com/primavera`.

Packwood Software (2010). Mpxj - microsoft project exchange. `http://mpxj.sourceforge.net`.

Padberg, F. (2006). A study on optimal scheduling for software projects. *Software Process: Improvement and Practice*, 11(1):77–91.

Penta, M. D.; Neteler, M.; Antoniol, G. e Merlo, E. (2005). A language-independent software renovation framework. *Journal of Systems and Software*, 77(3):225–240.

Pizzi, N. J. (2008). Software quality prediction using fuzzy integration: A case study. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 12(1):67–76.

Plekhanova, V. (1998). On project management scheduling where human resource is a critical variable. In *Proceedings of the 6th European Workshop on Software Process Technology*, pp. 116–121. Springer.

PMI (2008). *A Guide To The Project Management Body Of Knowledge (PMBOK Guides)*. Project Management Institute, 4a edição.

Pressman, R. (2005). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 6a edição.

Raccoon, L. B. (1996). A learning curve primer for software engineers. *ACM SIGSOFT Software Engineering Notes*, 21(1):77–86.

Ruhe, G. e Saliu, M. O. (2005). The art and science of software release planning. *IEEE Software*, 22(6):47–53.

Räihä, O. (2008). *Genetic Synthesis of Software Architecture*. PhD thesis, University of Tampere.

SBSE Repository (2010). Repository of publications on search based software engineering. `http://www.sebase.org/sbse/publications/`.

Schwaber, K. (2002). *Agile Software Development with SCRUM*. Prentice Hall, 1a edição.

Serena Software (2010). Openproj. `http://openproj.org`.

Sheta, A. F. (2006). Estimation of the cocomo model parameters using genetic algorithms for nasa software projects. *Journal of Computer Science*, 2(2):118–123.

Shevertalov, M.; Kothari, J.; Stehle, E. e Mancoridis, S. (2009). On the use of discretized source code metrics for author identification. In *Proceedings of the 1st International Symposium on Search Based Software Engineering*, pp. 69–78. IEEE Computer Society.

Silva, M. (2007). Auxílio à alocação de pessoas em projetos de software através de políticas. In *Anais do VI Simpósio Brasileiro de Qualidade de Software*.

Sommerville, I. e Rodden, T. (1996). *Trends in Software Process*, chapter Human, Social and Organizational Influences on the Software Process, pp. 89–110. John Wiley & Sons.

Souza, J.; Maia, C.; Freitas, F. e Coutinho, D. (2010). The human competitiveness of search based software engineering. In *Proceedings of the 2nd International Symposium on Search Based Software Engineering*, pp. 143–152. IEEE Computer Society.

Symons, C. R. (1988). Function point analysis: Difficulties and improvements. *IEEE Transactions on Software Engineering*, 14(1):2–11.

Taguchi, G. e Hsiang, E. A. E. T. C. (1988). *Quality Engineering in Production Systems*. Mcgraw-Hill, 1a edição.

Tsai, H.-T.; Moskowitz, H. e Lee, L.-H. (2003). Human resource selection for software development projects using taguchi's parameter design. *European Journal of Operational Research*, 151(1):167–180.

Turner, J. R. (2009). *The Handbook of Project-based Management*. McGraw-Hill, 3a edição.

Vijayalakshmi, K.; Ramaraj, N. e Amuthakkannan, R. (2008). Improvement of component selection process using genetic algorithm for component-based software development. *International Journal of Information Systems and Change Management*, 3(1):63–80.

Wei, C.-C.; Liu, P.-H. e Tsai, Y.-C. (2002). Resource-constrained project management using enhanced theory of constraint. *International Journal of Project Management*, 20:561–567.

Wikipedia (2010). Pareto efficiency. `http://en.wikipedia.org/wiki/Pareto_efficiency`.

Williams, T. e Kelley, C. (2007). *gnuplot: An Interactive Plotting Program*.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B. e Wesslén, A. (2000). *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers.

Xanthakis, S.; Ellis, C.; Skourlas, C.; Gall, A. L.; Katsikas, S. e Karapoulios, K. (1992). Application of genetic algorithms to software testing. In *Proceedings of the 5th International Conference on Software Engineering and Applications*, pp. 625–636.

Xiao, J.; Osterweil, L. J.; Wang1, Q. e Li, M. (2010). Dynamic resource scheduling in disruption-prone software development environments. In *Proceedings of 13th International Conference Fundamental Approaches to Software Engineering - Joint European Conferences on Theory and Practice of Software*, volume 6013, pp. 107–122. Springer.

Zhang, Y.; Finkelstein, A. e Harman, M. (2008). Search based requirements optimisation: Existing work challenges. In *Proceedings of the 14th International Working Conference, Requirements Engineering: Foundation for Software Quality*, volume 5025, pp. 88–94. Springer.

Zorgios, Y.; Vlismas, O. e Venieris, G. (2009). A learning curve explanatory theory for team learning valuation. *VINE: The Journal of Information and Knowledge Management Systems*, 39(1):20–39.