Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

RAFAEL MATOSO PEREIRA E SOUZA

Alocação de recursos em projetos de software através de técnicas de busca

Belo Horizonte
2012

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Especialização em Informática: Ênfase: Engenharia de Software

Alocação de recursos em projetos de software através de técnicas de busca

por

RAFAEL MATOSO PEREIRA E SOUZA

Monografia de Final de Curso

Profª. Gisele Lobo Pappa
Orientador(a)

Belo Horizonte
2012

RAFAEL MATOSO PEREIRA E SOUZA

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Especialista em Informática.

Área de concentração: Engenharia de Software
Orientador(a): Profª. Gisele Lobo Pappa

Belo Horizonte
2012

Folha aprovação

AGRADECIMENTOS

Agradeço primeiramente a Deus por sempre ter iluminado meus caminhos tornado-os, cada vez mais, menos tortuosos.

À minha mãe e a minha família, por terem me proporcionado alcançar mais essa conquista.

À professora Gisele Pappa, pela paciência e valorosa atenção que a mim concedeu durante todo o desenvolvimento deste trabalho.

Aos meus amigos, por sempre terem me estimulado e apoiado, contribuindo, direta ou indiretamente, na realização deste trabalho.

A todos que me acompanharam durante o curso, professores e colegas de classe, sem os quais não se teria alcançado mais esta vitória.

“Se vi mais longe,
foi porque me apoiei no ombro de gigantes”.
Isaac Newton (1642 – 1727)

RESUMO

No gerenciamento de projetos de software o planejamento é considerado fator chave para o sucesso. A alocação de recursos em projetos de software é reconhecida como uma atividade tão importante quanto complexa, sobretudo quando vários fatores são considerados nesta atividade. Este trabalho apresenta uma abordagem baseada em técnicas de otimização computacional, utilizando a *Search-Based Software Engineering* (SBSE), para resolução do problema de alocação de recursos em projetos de software. Na abordagem aqui apresentada o problema é definido como uma tarefa de otimização, na qual pretende-se otimizar a qualidade dos grupos formados para determinado projeto de software, com o objetivo de contornar limitações atuais através de uma abordagem automatizada. Espera-se com este trabalho encontrar a melhor configuração de equipes de trabalho levando em consideração a tarefa a ser executada, a habilidade de cada pessoa e a preferência por agrupamento de cada integrante da equipe. Ao final do trabalho, é apresentado o método proposto e simulações para aplicações de alocação de recursos a projetos de softwares. Uma análise mostra os resultados encontrados.

Palavras-chave: *Search-Based Software Engineering,* otimização, engenharia de software, alocação de recursos.

ABSTRACT

In software management projects, planning is considered a key factor for success. In this context, resource allocation is recognized as an activity as important as complex, especially when several factors are considered in this activity. This dissertation presents an approach for resource allocaation based on computational optimization techniques, using Search-based Software Engineering (SBSE). In the approach presented here the problem is defined as an optimization task in which you want to optimize the quality of the groups formed for a given software project, aiming to overcome current limitations through an automated approach. We expect this work to find the best configuration of working teams taking into account , for the task being performed, the ability of each person, the group preference for each team member and the number of people in each team. At the end of the work, the proposed method and simulations applications for resources allocation in software projects are presented., and an analysis shows the results.

Keywords*:* Search-based Software Engineering, optimization, software engineering, resource allocation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE SIGLAS

AG Algoritmos Genéticos
CE Computação Evolucionária
RAD *Rapid Application Development*
SBSE *Search-Based Software Engineering*

SUMÁRIO

1 INTRODUÇÃO

É cada vez maior a demanda por softwares de qualidade, e a aplicação de técnicas e práticas da engenharia de software no desenvolvimento de sistemas tem se tornado cada vez mais indispensável. Desde a sua origem em 1970, a engenharia de software tem conseguido muitos avanços na qualidade dos softwares desenvolvidos, e tal avanço se justifica pelo emprego de normas e metodologias no desenvolvimento de software.

Conforme Somerville (2003), a engenharia de software está relacionada com todos os aspectos de produção de software, desde os estágios iniciais até sua manutenção, depois de entrar em operação. A engenharia de software permite adotar uma abordagem sistemática e organizada durante o desenvolvimento de sistemas. Isto gera produtos mais eficazes e de alta qualidade.

Os métodos da engenharia de software foram criados com base nos conhecimentos de cientistas e especialistas, e funcionam como guia durante todo o processo de desenvolvimento, informando como e o que deve ser feito durante cada fase. Essas técnicas tem sido cada vez mais empregadas, uma vez que a qualidade do processo de desenvolvimento determina a qualidade do produto final (FUGGETTA, 2000).

Conforme mencionado, a engenharia de software traz vários benefícios durante o processo de desenvolvimento que são agregados no produto final. Os métodos da engenharia de software podem ser empregados nas mais diversas etapas do ciclo de desenvolvimento, tais como planejamento do projeto, análise de requisitos, documentação, desenvolvimento, testes e manutenção.

No desenvolvimento de software, atividades como testes, manutenção e planejamento são essenciais. Um planejamento falho ou fraco pode causar sérios impactos em todo o projeto de software, significando aumento de custos, tempo e quebra de restrições que geralmente são inaceitáveis. Desta forma, o mau planejamento pode levar um projeto ao fracasso, trazendo enormes prejuízos à organização desenvolvedora. O planejamento de sistemas de larga escala envolve uma série de esforços, desde a realização de estimativas até a alocação de pessoas para determinado grupo de trabalho.

Pode-se perceber que produzir software é uma atividade complexa (SOMERVILLE, 2003), e muitas vezes as metodologias desenvolvidas pela comunidade científica não contemplam todo o processo de desenvolvimento de software. Isto acontece com problemas difíceis, que não podem ser resolvidos por meio de técnicas tradicionais. Esse é o caso, por exemplo, em problemas em que deve-se encontrar a solução considerando um conjunto grande de soluções possíveis. Nestes tipos de problemas são exigidas soluções automatizadas que possam tratar os diversos aspectos relacionados ao problema, varrendo praticamente todo o espaço de soluções possíveis e apresentando uma solução ótima, ou muito próxima a ótima.

Como exemplo de uma atividade complexa no processo de desenvolvimento de software, pode-se citar a fase de testes, onde são definidas diversas fases e critérios. Uma tarefa importante no processo de realização de testes é a seleção de casos de testes. A seleção de casos de testes não pode ser facilmente descrita por meio de regras textuais ou passos descritos em documentos. A modelagem matemática de parâmetros e critérios de satisfação em relação a determinada característica se mostra mais adequada (FREITAS *et al*., 2009).

Outro exemplo de problema não completamente resolvido pelas técnicas convencionais é a alocação de recursos para projetos de software, sobretudo quando diversas características envolvem este processo de seleção. Diversos fatores influenciam na decisão de formação de equipes, desde fatores humanos a aspectos relacionados ao produto e engenharia. Estes aspectos fazem com que sejam necessárias novas abordagens para a alocação de recursos em projetos de software. O processo de alocação de recursos em um projeto, alocação de pessoas a determinado grupo, ou formação de equipe de trabalho, é uma instância do "*bin packing problem"*, um problema da classe NP-difícil (ANTONIOL, PENTA e HARMAN, 2004), ou seja, tal tipo de problema não é resolvido por meio de técnicas tradicionais ou algoritmos determinísticos. Esse problema é a base do trabalho apresentado aqui.

Para superar estas dificuldades que surgem durante aplicação de técnicas tradicionais da engenharia de software, criou-se o conceito de *Search-Based Software Engineering* (SBSE), que utiliza meta-heurísticas para resolver problemas da engenharia de software que podem ser vistos como NP-difícil.

A SBSE, através do uso de meta-heurísticas, soluciona problemas relacionados a todo o ciclo de vida de desenvolvimento de software, tais como versionamento, estimativas, testes, planejamento e estudos de evolução de software.

A SBSE complementa as técnicas tradicionais de engenharia de software, e os avanços conseguidos pelas últimas nas ultimas décadas, em teoria e prática, continuam válidos. Porém, alguns problemas que antes não eram completamente resolvidos ou sequer tratados pelas metodologias e métodos convencionais passaram a ser estudados e solucionados. Além disso, esta nova visão da engenharia de software apresenta resultados mais aplicáveis em diferentes projetos e empresas (FREITAS *et al.,* 2009).

Uma meta-heurística bastante utilizada na resolução de problemas intratáveis por meio de técnicas tradicionais são os Algoritmos Genéticos (AG). Os algoritmos genéticos (GOLDBERG, 1989) são algoritmos de busca, baseados na teoria da evolução das espécies de Charles Darwin, onde os indivíduos de uma determinada população evoluem de acordo com os princípios de seleção natural e sobrevivência dos mais aptos. Estes algoritmos simulam os processos de evolução da natureza, onde cada indivíduo representa uma possível solução para um problema dado.

Algoritmos genéticos são indicados para tratar problemas de otimização complexos, que envolvem um grande número de variáveis e, consequentemente, espaços de soluções de dimensões elevadas. Além disso, ao contrário da maioria dos algoritmos tradicionais, eles identificam e exploram o espaço de busca global e não ficam presos em ótimos locais. Esta característica é uma das mais interessantes dos algoritmos genéticos, e faz com que eles sejam uma técnica extremamente adequada para funções multimodais e de perfis complexos, como a maioria das funções de custo associadas a problemas reais (LINDEN, 2008).

O principal objetivo deste trabalho é estudar o problema da alocação de recursos a grandes projetos de software, levando em consideração não apenas fatores técnicos, mas também fatores relacionados aos aspectos humanos, tais como a afinidade entre os indivíduos de cada equipe. Este problema pertence a classe dos problemas NP – difíceis, e algoritmos comuns são incapazes de resolvê-los em tempo polinomial. Para tal, serão empregadas técnicas de SBSE, utilizando algoritmos genéticos na resolução do problema. Espera-se com o emprego destes obter a formação de equipe ideal, levando em consideração os aspectos requeridos.

Este trabalho está organizado em sete capítulos. Os capítulos 2, 3, 4 e 5 apresentam uma revisão teórica dos principais conceitos de engenharia de software, *Search-Based Software Engineering* e Algoritmos Genéticos. O capítulo 5, por sua vez, descreve o método proposto, ou seja, a abordagem proposta para solucionar o problema

de alocação de recursos a grandes projetos. O capítulo 6 apresenta simulações, resultados obtidos e uma análise destes resultados. Conclusões e trabalhos futuros serão apresentados no capitulo 7.

2 ENGENHARIA DE SOFTWARE E *SEARCH BASED SOFTWARE ENGINEERING*

Segundo o IEEE (1990), engenharia de software é a aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de software. A engenharia de software é uma área de conhecimento da computação voltada para a especificação, desenvolvimento e manutenção de sistemas de software, onde se aplicam tecnologias, normas e práticas de gerência de projetos objetivando organização, produtividade e qualidade. Como resultado do emprego da engenharia de software, a organização desenvolvedora consegue entregar produtos confiáveis e de alta qualidade a um custo aceitável, visto que todo o processo de desenvolvimento é controlado e dirigido pelas normas e práticas da engenharia de software.

A *Search-Based Software Engineering*, como o próprio nome indica, compreende um conjunto de técnicas de busca indicadas para resolver problemas relacionados à engenharia de software que estão na classe NP-difícil. A SBSE utiliza meta-heurísticas para varrer o espaço de busca, afim de encontrar uma solução ótima ou quase-ótima. Nas últimas décadas esta técnica tornou-se mais difundida, uma vez que os algoritmos comuns são incapazes de revolver certos problemas da engenharia de software, tais como: alocação de recursos a projetos de softwares, estimativas de custos de softwares, seleção de casos de testes, etc.

2.1 Engenharia de software: uma visão geral

A Engenharia de software abrange um conjunto de três elementos fundamentais – métodos, ferramentas, procedimentos - que possibilita ao gerente o controle de processo de desenvolvimento do software de alta qualidade produtivamente (PRESSMAN, 1995).

A engenharia de software nasceu em meados de 1970, na tentativa de contornar os problemas gerados pela crise do software e dar um tratamento de engenharia ao processo de desenvolvimento do software, ou seja, aplicar técnicas e normas afim de controlar todo o processo de desenvolvimento de software, principalmente os softwares complexos. Um sistema de software complexo se

caracteriza por um conjunto de componentes abstratos de software (estruturas de dados e algoritmos) encapsulados na forma de procedimentos, funções, módulos, objetos ou agentes interconectados entre si, compondo a arquitetura do software, que deverão ser executados em sistemas computacionais (SILVA, 2011).

Processos de Software

A utilização de processos de software é apontada como fator primordial de sucesso nas empresas desenvolvedoras de softwares. O processo de software indica o conjunto de atividades bem estruturadas que devem ser empregadas para desenvolver um software. Somerville (2009) define processos de software como "um conjunto de atividades e resultados associados que produzem um produto de software".

A comunidade científica tem esforçado para criar processos de softwares específicos para determinada organização ou produto de software. Conforme Schwartz (1975) e Somerville (2003) pode-se observar que, independe do processo de software adotado, todos apresentam as seguintes atividades:

- Especificação de Requisitos: Tradução da necessidade do cliente ou requisito operacional para uma descrição da funcionalidade a ser executa;

- Projeto: Tradução destes requisitos em uma descrição de todos os componentes necessários para codificar o sistema;

- Implementação: Tradução dos requisitos estabelecidos no projeto para uma linguagem de programação, afim de gerar o programa executável;

- Validação: Garantir que o software produzido atende aos requisitos que foram propostos antes da sua construção;

- Manutenção e Evolução: Fase na qual o software entra em um ciclo interativo que abrange todas as fases anteriores;

Modelos de Processos de Software

Um modelo de processo de software é uma representação abstrata das atividades, papéis e artefatos envolvidos em um processo de software. Abaixo descrevemos os modelos que são amplamente discutidos e empregados:

- Clássico, Sequencial ou Cascata:

Modelo onde a principal característica é a sequência de atividades, onde cada fase é executada separadamente, uma após a outra, ou seja, uma etapa "flui" após o encerramento de outra. A Figura 1 mostra o fluxo e apresenta as fases do modelo cascata.

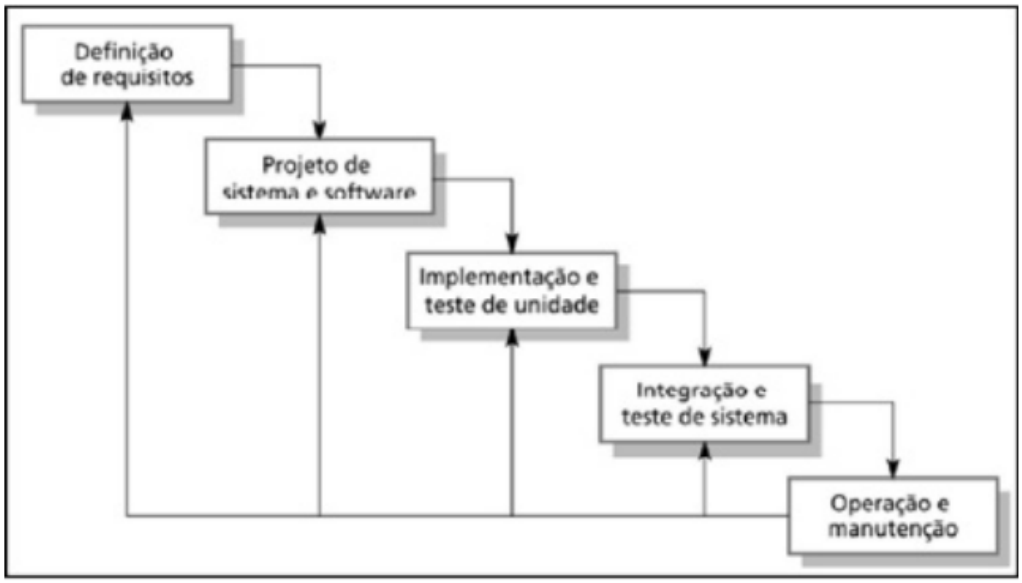

FIG. 1 Fases do modelo cascata. Fonte: SOMERVILLE (2003).

O modelo pressupõe que o cliente participa ativamente das etapas do projetos e que este já possui os requisitos bem definidos. Por isso, este modelo não é indicado para projetos onde os requisitos mudam freqüentemente.

- Modelo de Prototipagem

A ideia deste modelo é construir um protótipo do produto a ser desenvolvido como forma de auxiliar no entendimento dos requisitos. O desenvolvedor e o cliente se reúnem e definem os objetivos globais, assim ocorre a elaboração de um "protótipo rápido" o qual concentra-se em representar aqueles aspectos visíveis aos usuários. A Figura 2 mostra as fases do modelo de prototipagem:

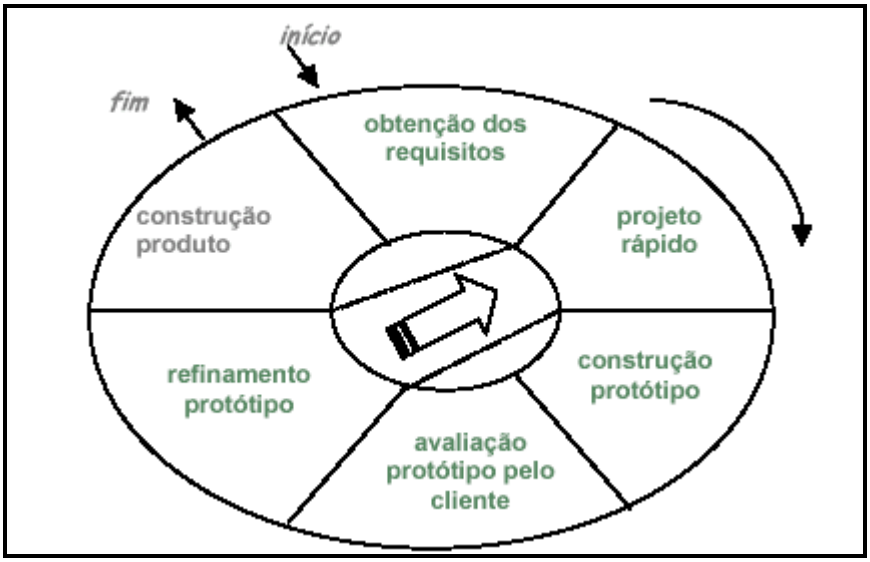

FIG. 2 Fases do modelo de prototipagem. Fonte: Adaptado de PRESSMAN (1995).

A prototipação é um mecanismo eficiente, porém é importante cliente e desenvolvedor estarem cientes que o protótipo foi construído apenas para levar os requisitos e esclarecer dúvidas quanto ao produto que deverá ser desenvolvido. Ele não deve ser utilizado como parte do produto final, uma vez que foi construído visando levantar os requisitos junto aos usuários, e não leva em conta aspectos técnicos, como variáveis do ambiente, sistema operacional e linguagem de programação que será utilizada.

- Modelo Espiral

Este modelo foi criado para abranger as melhores características do modelo cascata e prototipagem, acrescentado um novo elemento: a análise de riscos. Neste modelo o projeto é executado em vários pequenos ciclos, e ao final de cada ciclo uma versão executável é entregue ao cliente.

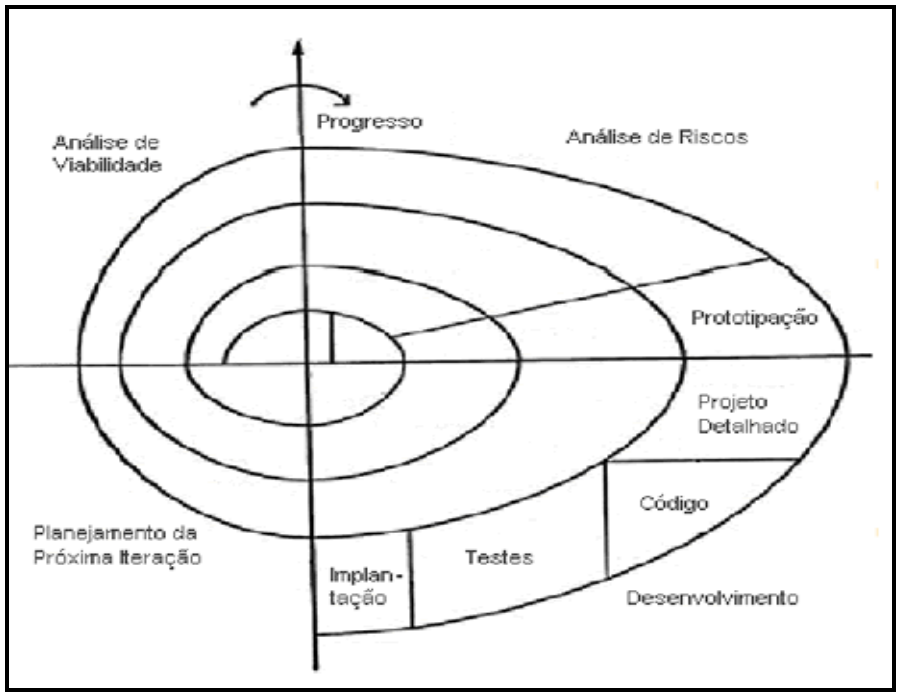

FIG. 3 Fases do modelo espiral. Fonte: BUCHER et. al. (2011).

A Figura 3 representa as fases do modelo espiral. Este modelo é considerado o mais realístico possível, pois assume que usuários, analistas e desenvolvedores adquirem maior conhecimento sobre o projeto como decorrer do tempo (SOMMERVILLE, 2003).

- Modelo de desenvolvimento iterativo e incremental

Este modelo é uma extensão do modelo espiral, porém mais rigoroso e formal. O desenvolvimento de software é complexo, para diminuir esta complexidade pode-se dividir o trabalho em partes menores ou iterações. Estas iterações resultam em incrementos. A Figura 4 apresenta as fases do modelo de desenvolvimento iterativo e incremental

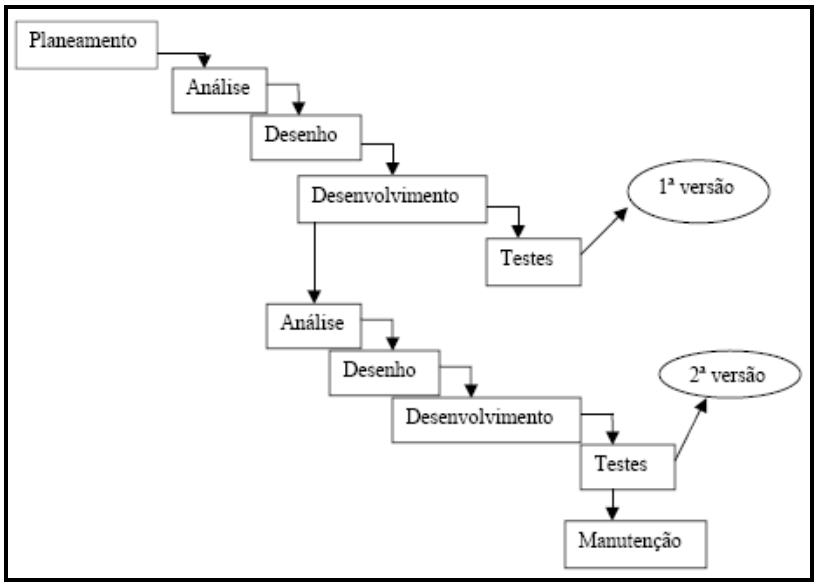

FIG. 4 Representação do modelo iterativo e incremental. Fonte: Adaptado de MACORATTI (2011).

Uma das vantagens deste modelo é a possibilidade de avaliar mais cedo os riscos e pontos críticos do projeto, permitindo assim identificar medidas para eliminá-los ou controlá-los.

- Modelo RAD (*Rapid Application Development*)

Este modelo enfatiza um ciclo de desenvolvimento curto, com o uso de uma abordagem de construção baseada em componentes. Por ser baseada em componentes, possibilita a reutilização de código que permite que a equipe de desenvolvimento possa trabalhar em paralelo e desenvolver um sistema completamente funcional em pouco tempo. A Figura 5 representa o modelo RAD.

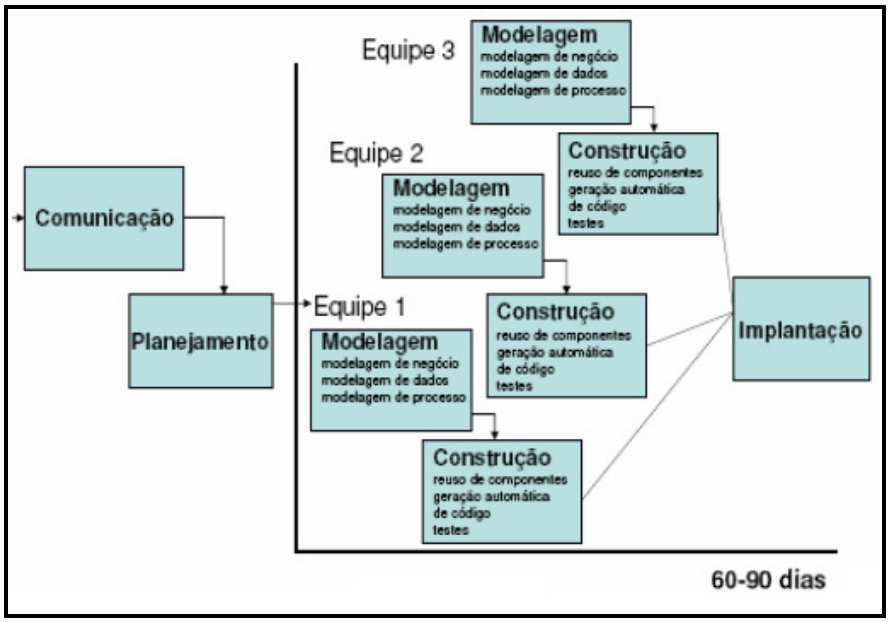

FIG. 5 Representação do modelo RAD. Fonte: MAIA (2009).

Por permitir o desenvolvimento paralelo, os requisitos do produto precisam estar bem definidos e com um certo grau de independência entre eles. Se uma das situações citadas não ocorrerem o modelo pode não ser adequado.

Seja qual o for o modelo de processo adotado, durante o projeto de construção de software, toda organização de desenvolvimento de software necessita alocar recursos para que o produto possa ser produzido. A tarefa de alocação de recursos não é trivial, por isto este trabalho apresenta uma técnica que auxilia a alocação de recursos a projetos de softwares.

2.2 *Search-based software engineering* (SBSE)

Técnicas e normas da engenharia de software são cada vez mais empregadas pelas empresas no intuito de fornecer softwares de alta qualidade. Desde a criação da Engenharia de software em 1970, vários metas foram alcançados devido a pesquisas em modelos, tecnologias, normas e metodologias que pudessem gerar software cada vez melhores e eficazes.

Pesquisas na área de *Search-Based Software Engineering* mostram que cada vez mais problemas da engenharia de software podem ser resolvidos através de meta-heurísticas. São exemplos de problemas clássicos da engenharia de software resolvidos através da abordagem *search-based*: problema do "próximo *release*", estimativa de

custos de software, testes, remodularização, transformação e estudos de evolução do software.

O termo meta-heurísticas representa um conjunto de algoritmos heurísticos que se baseiam em diversas fontes para realizar a busca da solução para problemas de otimização. Alguns exemplos de meta-heurísticas bastante utilizada são algoritmos genéticos e têmpera simulada (*simulated annealing*). Os algoritmos heurísticos, diferentemente dos algoritmos convencionais, conseguem varrer um vasto espaço de busca, podendo ser aplicados a inúmeros problemas complexos. Nas últimas décadas, inúmeros trabalhos tem sido desenvolvidos através da aplicação deste método para resolução de diversos tipos de problemas, incluindo os da engenharia de software.

A engenharia de software, como qualquer outra engenharia, é um campo onde existem problemas que podem ser resolvidos matematicamente. A SBSE tem se mostrado bastante útil na resolução de diversos problemas relacionados ao processo de desenvolvimento de software. Problemas que antes não eram resolvidos, por não serem tratados pelas técnicas convencionais, passam a ser estudados e solucionados.

Alguns problemas relacionados à engenharia de software são complexos e apresentam alto número de possibilidades. Problemas deste tipo não podem ser resolvidos por meio de métodos exaustivos, devendo ser resolvidos por meio de métodos inteligentes e automatizados.

Recentes trabalhos mostram a aplicação da SBSE em diversas áreas da engenharia de software. No entanto as áreas mais comuns e abordadas em grande número de trabalho são: engenharia de requisitos, testes de software e estimativa de software. A seguir, será mostrado como a SBSE pode auxiliar na resolução de problemas destas áreas da engenharia de software.

A engenharia de requisitos é a área que trata da definição dos requisitos. Estes são levantados afim de possibilitar a implementação das necessidades do cliente no software a ser desenvolvido. O problema surge quando é preciso selecionar o conjunto de requisitos que deve estar presente na próxima iteração do desenvolvimento, este problema é conhecido como *"The Next Release Problem"* (problema do próximo Release). Aplicações de meta-heurísticas se mostraram adequadas na resolução deste tipo de problema, como mostrou o trabalho desenvolvido por BAGNALL (2001).

Os Testes de Software são técnicas realizadas para garantir que o produto final atende aos usuários e correspondem aos requisitos levantados junto a eles. A aplicação de meta-heurísticas na área de testes de software tem tentado resolver os

problemas de priorização de casos de teste, seleção de casos de teste e geração de dados de teste para teste unitário. Os testes de softwares se mostram eficazes quando encontram algum defeito no software desenvolvido, e isto não é tarefa fácil, uma vez que, muitas vezes é preciso testar inúmeras combinações para que um defeito seja descoberto. A SBSE tem mostrado ideal para resolução deste tipo de problema, como mostrado no trabalho de YOO, HARMAN (2003).

Estimativas de Software se refere à atividade realizada ainda no planejamento do projeto de software, onde é previstos os recursos necessários, custo, tempo e prazo para execução do projeto. Dado a sua relevância, inúmeros trabalhos foram desenvolvidos na tentativa de encontrar técnicas eficazes que fornecesse estimativas mais realísticas. Exemplos (BURGESS, 2001) mostram que a utilização de meta-heurísticas apresenta melhoria de resultados em relação a outras técnicas.

Não apenas nas áreas citadas acima, mas a SBSE pode auxiliar na resolução de problemas complexos das mais variadas áreas da engenharia de software. A Tabela 1 mostra as diversas áreas da engenharia de software e em qual campo de cada área a SBSE contribui na resolução de problemas. Repare que a SBSE pode ser aplicada a praticamente todos os campos da engenharia de software, desde planejamento do projeto, passando pelo desenvolvimento até as atividades de manutenção do software que acontecem depois da entrega do produto ao cliente.

TABELA 1

Aplicação de SBSE em diversas áreas

Área da engenharia de So	Aplicação da SBSE
Engenharia de Requisitos	Análise de Requisitos Requisitos Seleção de Requisitos
Teste de Software	Seleção de Casos de Teste Priorização de Casos de Teste Geração de Dados de Teste
Estimativa de Software	Estimativa de tamanho Estimativa de custo de Software
Planejamento de Projeto	Alocação de Recursos
Otimização de Código-fonte	Otimização de Código-fonte
Manutenção de Software	Re-engenharia de Software *Automated Maintenance*
engenharia de software orientada a Serviço	Desenvolvimento de Software
Otimização de Alocação Compilador	Alocação em heap Tamanho de código

Fonte: FREITAS (2009).

Conforme pode ser observado a SBSE permite uma nova forma de visualização dos problemas, dando aos mesmos um tratamento racional e automatizado. Problemas que antes não eram solucionados são resolvidos através de meta-heurísticas. Isto permite aos envolvidos com o processo de desenvolvimento se preocupar com as atividades onde a capacidade e inteligência humana são mais convenientes, transfere-se assim a complexidade do problema para os algoritmos meta-heurísticos.

3 ALGORITMOS GENÉTICOS

A Computação Evolucionária (CE) é um modelo computacional para a solução de problemas do tipo "gerar e testar", aplicável onde não há algoritmo eficiente disponível e também nas situações onde não é possível utilizar soluções analíticas. A CE teve suas motivações baseadas na evolução das espécies, no qual um indivíduo se esforça para sobreviver e se reproduzir. A aptidão de cada indivíduo em um determinado ambiente é muito importante para que este consiga alcançar seus objetivos com sucesso, ou seja, consiga sobreviver e reproduzir. Indivíduos mais aptos possuem grandes chances de se manterem na população e auxiliarem na geração dos próximos indivíduos.

Algoritmos Genéticos (AG) são os representantes mais populares da classe CE. O uso de populações de indivíduos permite evoluir várias soluções ao mesmo tempo, à medida que os operadores de recombinação mesclam informações de vários candidatos nos novos indivíduos. Algoritmos genéticos são especialmente atrativos por não exigirem que se saiba como encontrar uma solução ótima para um problema, mas sim como reconhecê-la como ótima (PAPPA, 2002).

3.1 Conceitos Básicos

A teoria da evolução, de acordo com a biologia, vem mostrando que a natureza é composta por vários seres vivos, que passam por dificuldades, onde a política da sobrevivência é determinada pelo mais forte, ou seja, mais apto ao ambiente. Indivíduos mais aptos tem maiores chances de sobreviver e gerar descentes. A cada nova geração, a população gera novos indivíduos, substituindo os indivíduos que não sobreviveram e fazendo com que a diversidade genética se mantenha a cada nova comunidade de indivíduos gerada. Os algoritmos genéticos simulam estes fenômenos,. Para o melhor entendimento deste trabalho, inicialmente introduziremos algumas terminologias:

- **Cromossomo**: Cadeia de caracteres representando alguma informação relativa às variáveis do problema. Cada cromossomo representa uma solução

para o problema sendo resolvido. Um cromossomo é formado por diversos genes;

- **Gene**: É a unidade básica do cromossomo. Cada cromossomo tem um certo número de genes, cada um descrevendo uma certa variável do problema;
- **Lócus**: Posição ocupada por um determinado gene em um cromossomo;
- **Indivíduo**: É apenas um membro da população. É representado por seu cromossomo e um valor de aptidão;
- **População**: Conjunto de indivíduos ou soluções;
- **Geração**: O número de interação que o algoritmo genético executa;
- **Operações Genéticas**: Operação que o algoritmo genético realiza sobre cada um dos cromossomos;
- **Espaço de Busca ou região viável**: É a área, ou conjunto de soluções viáveis de um determinado problema que deve ser otimizado. Esta área de soluções é definida e delimitada pelas restrições do problema proposto;
- **Função Objetivo ou de Avaliação:** É a função que se quer otimizar. Ela contém a informação numérica (valor de aptidão) do desempenho de cada cromossomo na população. Nela estão representadas as características do problema que o algoritmo genético necessita para realizar seus objetivos (CASTRO, 2001).

3.2 Características dos Algoritmos Genéticos

Algoritmos Genéticos são técnicas probabilísticas e não determinísticas. Dessa forma, com uma mesma população inicial, podem acontecer casos em que cada execução do algoritmo genético apresente uma solução diferente para o problema.

Algoritmos genéticos, ao contrário de técnicas determinísticas, trabalham com uma população de resultados levando em consideração o espaço global, e não o espaço local como acontece em técnicas comuns (determinísticas).

Apesar de utilizar técnicas probabilísticas, os AGs diferenciam dos esquemas aleatórios por serem uma busca que utiliza informações pertinentes ao problema e não trabalham com caminhadas aleatórias (*rondom walks)* pelo espaço de

soluções, mas sim direcionando sua busca através do mecanismo da seleção, equivalente ao processo natural (LINDEN, 2008).

Outra característica importante dos AGs é que estes utilizam a representação codificada dos parâmetros a serem utilizados na execução do algoritmo. Toda a informação relativa ao problema deve ser completamente mapeada para uma função (função objetivo) que é utilizada para distinguir a qualidade de cada indivíduo da população.

3.3 Funcionamento

Algoritmos Genéticos, de acordo com Linden (2008), podem ser descrito resumidamente através dos seguintes passos:

1. Inicializar uma população de indivíduos (cromossomos);
2. Avaliar cada cromossomo da população através da função objetivo;
3. Selecionar os pais para gerar novos indivíduos (cromossomos);
4. Aplicar operadores recombinação e mutação a estes pais de forma a gerar os indivíduos da nova geração;
5. Excluir os antigos membros da população;
6. Avalie todos os novos indivíduos e insira na população atual;
7. Se o tempo encerrou, ou o melhor cromossomo satisfaz os requerimentos e desempenho, retorne-o, caso contrário ir para o passo 3.

A Figura 6 mostra um esquema de um algoritmo genético simples. O critério de parada, representando por “Cond”, é alcançado quando o número de gerações foi atingido ou quando a solução para o problema consideravelmente boa, ou quando o algoritmo não consegue mais evoluir.

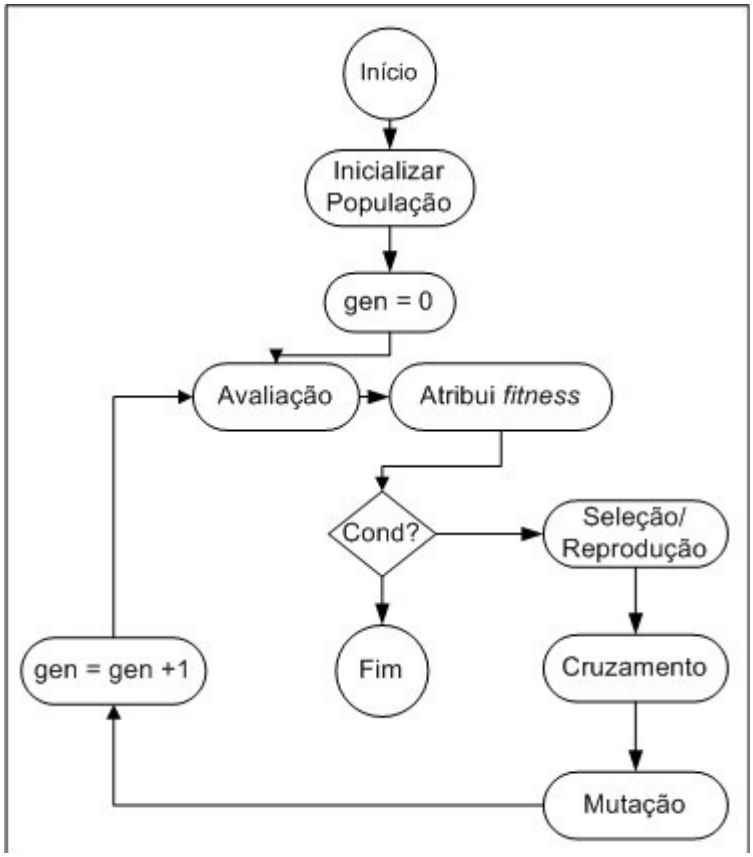

FIG. 6 Diagrama de fluxo de um AG Simples. Fonte: PAPPA (2002).

3.4 Representação Cromossomial

A representação cromossomial é muito importante para a solução do problema em questão. Esta representação consiste em codificar as informações pertinentes ao problema para uma maneira viável de ser tratada pelo computador. No entanto, quanto melhor a representação de um cromossomo representa o problema, melhores resultados serão atingidos.

A representação ou codificação das variáveis de projeto a serem otimizadas proporciona um grande impacto no desempenho de busca e resolução do problema, devendo ser o mais simples e correto possível sem perder, no entanto, as características de representação do problema tratado (CASTRO, 2001). Dentre as representações mais utilizadas estão: binária, inteira ou real. Aqui explicamos a representação binária, que será utilizada nesse trabalho.

A representação binária é a mais simples e mais utilizada pelos programadores de algoritmos genéticos. A representação binária representa um cromossomo como uma cadeia de *bits* (cada gene possui um bit). Dessa forma cada

gene pode assumir apenas dois valores: 0 ou 1. Além de permitir uma simplicidade na programação, a representação binária oferece facilidade e rapidez no emprego de operadores genéticos, tais como o cruzamento e a mutação.

Por outro lado, existem casos em que a representação dos cromossomos através da forma binária pode ser não viável. Considere, por exemplo, um problema em que para efeito de avaliação do indivíduo precise a todo momento transformar a cadeia de bits para números inteiros ou reais. O processamento oriundo desta transformação poderia ser evitado caso a representação do indivíduo fosse feita diretamente em termos de número inteiros ou reais, o que refletiria no desempenho computacional.

3.5 Geração da População Inicial

A inicialização da população, da maioria das vezes, é feitada forma mais simples possível, fazendo-se uma escolha aleatória independente para cada indivíduo da população inicial. (LINDEN, 2008).

Porém, além da inicializar da população aleatória, uma heurística pode ser utilizada no momento na inicialização de cada indivíduo. Isto proporciona a criação de indivíduos mais “interessantes”, pois os indivíduos possuem algum tipo de informação prévia. Pode-se utilizar de várias técnicas para a criação da população inicial, porém o que deve ser garantido é a variabilidade dos cromossomos criados, gerando uma diversidade entre os membros da população.

No entanto, de acordo com o problema, os indivíduos devem obedecer a certas restrições impostas. Uma técnica muito utilizada para favorecer os indivíduos que respeitam as restrições do problema é utilizar alguma forma de bonificar os cromossomos adequados e penalizar os cromossomos inadequados.

Criar uma representação e um conjunto de operadores que sempre mantêm dentro da população apenas indivíduos que sejam admissíveis é uma técnica (que quando possível) muito mais confiável e gera resultados superiores à técnica de penalização de indivíduos (LINDEN, 2008).

3.6 Função de Avaliação

A função de avaliação é um componente importante dos algoritmos genéticos. Ela é responsável por dizer o quão bom um indivíduo é em relação ao

problema. A função de avaliação liga o problema real ao algoritmo genético e, portanto, ela deve possuir uma boa definição. Uma definição que não represente todos os aspectos reais do problema pode não gerar soluções satisfatórias para o mesmo.

A função de avaliação deve ser escolhida com grande cuidado. Ela deve embutir todo o conhecimento que se possui sobre o problema a ser resolvido, tanto suas restrições quanto seus objetivos de qualidade. Quanto mais conhecimento é embutido em um AG, menos ele se comportaria como algoritmos genéricos (LINDEN, 2008).

3.7 Seleção de Indivíduos

O método de seleção deve simular o mecanismo de seleção natural. Ou seja, no processo de seleção, os indivíduos com melhor valor de aptidão possuem muito mais chances de reproduzirem e gerarem filhos. No entanto, devem-se criar mecanismos de modo a privilegiar os indivíduos com valor de aptidão mais alto, mas também não se podem desprezar os indivíduos que possuem um valor de aptidão baixo. Estes indivíduos, uma vez selecionados no processo de cruzamento, podem ser de grande utilidade, pois mantém a diversidade da população e podem gerar indivíduo que pode vir a ser a melhor solução do problema.

É importante salientar que, permitindo apenas os melhores indivíduos se reproduzirem, a população tenderá a ser composta de indivíduos cada vez mais semelhantes e faltará diversidade a esta população para que a evolução possa prosseguir de forma satisfatória, com outras palavras, pode-se dizer que haverá uma convergência genética (LINDEN, 2008).

Vários métodos podem ser utilizados para realizar a seleção de indivíduos, entre eles, a seleção proporcional, a seleção por *ranking* e a seleção por torneio (PAPPA, 2002).

Na seleção proporcional, os indivíduos podem ser selecionados de acordo com o valor de sua função de avaliação. Logo, em um problema de minimização de funções, indivíduos que possuem um valor de aptidão baixo tem muito mais chances de serem selecionados, do que indivíduos que possuem um valor de aptidão mais alto .

Um dos métodos bastante utilizados na implementação de seleção proporcional é o método da roleta viciada. Neste método uma roleta é dividida em N partes, sendo que N representa o número de indivíduos da população. Cada parte é

proporcional ao valor de aptidão de cada indivíduo. A roleta então é girada, e então a posição apontada pelo ponteiro indica qual é o indivíduo a ser escolhido para cruzamento. Para melhor entendimento, considere a Tabela 2, onde são mostrados os indivíduos, a sua avaliação e o valor representado na roleta.

TABELA 2

Representação de indivíduos, avaliação e o respectivo valor na roleta

Indivíduo	**Avaliação**	**Pedaço da roleta (º)**
0001	1	5.8
0011	9	52.2
0100	16	92.9
0110	36	209.1
TOTAL	**62**	**360.0**

Fonte: Adaptado de Linden (2008).

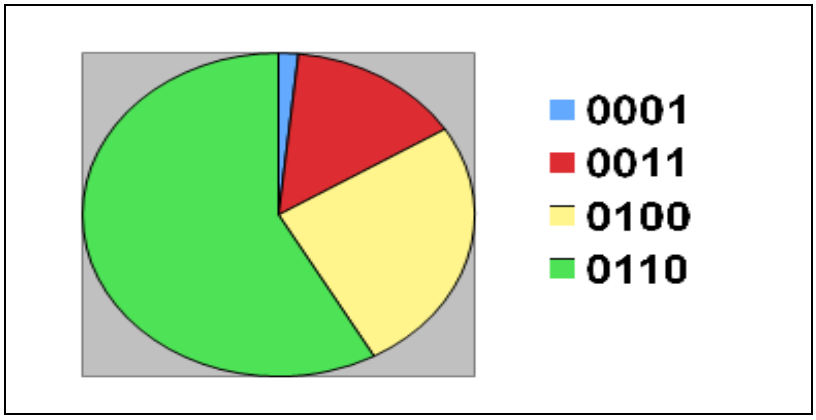

FIG. 7 Roleta para os indivíduos da Tabela 2. Fonte: Próprio autor.

No método da roleta, como dito anteriormente, cada fatia da roleta se refere a um indivíduo, conforme a Figura 7. Nela, pode-se observar que o indivíduo “0001”, mesmo possuindo um valor de aptidão baixo, possui chances de ser selecionado (embora estas chances sejam mínimas), o que faz com que a população possua diversidade genética.

O método de seleção por *ranking* pode ser dividida em duas etapas. Na primeira etapa os indivíduos são ordenados em função dos seus valores de aptidão. Estando a lista ordenada, a cada indivíduo é atribuído um novo valor da função de avaliação, sendo que este valor é equivalente à sua posição no ranking. Na segunda etapa um método semelhante à seleção proporcional é aplicado, o que faz com que quanto mais privilegiada a posição de um indivíduo no *ranking*, maior a chance dele ser selecionado.

Já na seleção por torneio, não há chances proporcionais à aptidão de cada indivíduo. Através deste método, inicialmente todos os indivíduos possuem chances iguais de serem selecionados para o processo de cruzamento. Através deste método, P indivíduos, sendo P > 2, onde P é denominado tamanho do torneio, são escolhidos aleatoriamente. O indivíduo com maior valor de aptidão é então selecionado para reprodução.

Dentre as vantagens do método torneio pode-se destacar (CASTRO, 2001):

- Não acarreta convergência prematura;
- Combate à estagnação;
- Nenhum esforço computacional extra é necessário, tal como ordenamentos;
- Aptidão explícita é desnecessária;
- Inspiração biológica do processo.

3.8 Elitismo

Por que perder a melhor solução encontrada? Pensando nessa questão foi criado o mecanismo denominado elitismo. O elitismo consiste em transferir para a nova geração k melhores indivíduos da população.

Elitismo é uma pequena modificação no módulo de população que quase não altera o tempo de processamento, mas que garante que o desempenho do algoritmo genético sempre cresce no decorrer das gerações (LINDEN, 2008).

Na maioria das implementações existentes, pelo menos o elitismo do melhor indivíduo é utilizado. Sua desvantagem é a possibilidade de forçar a busca, pela presença de mais uma cópia do melhor indivíduo, na direção de algum ponto ótimo

local que tenha sido descoberto antes do global, embora um algoritmo genético possa escapar destas armadilhas (CASTRO, 2001).

O elitismo, apesar de sua simplicidade, normalmente colabora de forma dramática para a melhoria do desempenho de uma execução de um algoritmo genético (LINDEN, 2008).

3.9 Operadores Genéticos

Os operadores genéticos são os responsáveis por transformar os indivíduos através de sucessivas gerações. Os operadores genéticos são importantes, uma vez que são os responsáveis por manter a diversidade da população e manter as características de adaptabilidade conforme o processo natural de evolução. Os algoritmos genéticos tradicionais consistem basicamente de dois operadores, são eles: o operador de cruzamento e o operador de mutação.

3.9.1 Operador de Cruzamento

O cruzamento, também chamado de recombinação, é o responsável pela troca de material genético entre dois genitores. Ele garante a propagação do material genético pelas *g* gerações. A troca de material genético entre dois cromossomos ocorre a partir de um ou mais pontos da cadeia de genes, escolhidos aleatoriamente. O operador de cruzamento, também chamado de *crossover,* pode ser empregado de diversas maneiras, as mais comumente usadas são: *crossover* de um ponto, dois pontos ou uniforme.

3.9.1.1 *Crossover* de um ponto

No *crossover* de um ponto, após ter selecionado dois pais através de um método de seleção, um ponto de corte é definido aleatoriamente. Um ponto de corte constitui uma posição entre dois genes de um cromossomo (LINDEN, 2008).

A troca de material genético no *crossover* de um ponto acontece à direita do ponto escolhido. Um exemplo de cruzamento de um ponto é mostrado nas Figuras 8 e 9. O ponto de corte é indicado pela seta.

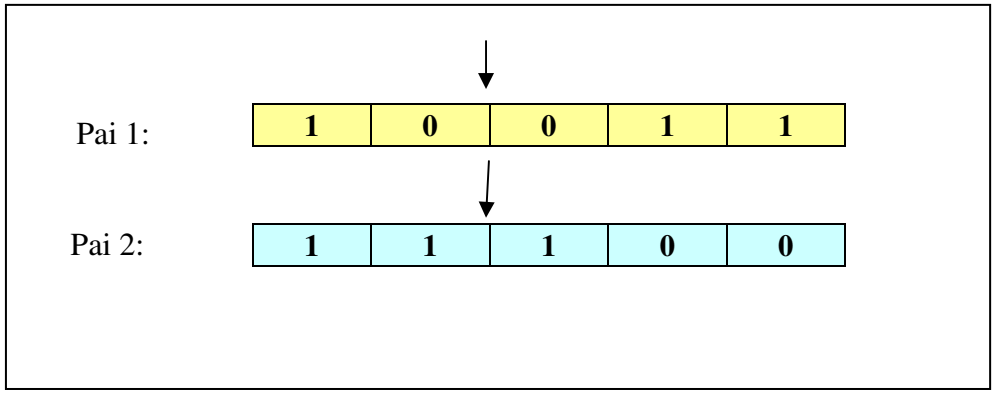

FIG. 8 Cromossomos pais antes do crossover de um ponto. Fonte: Próprio autor.

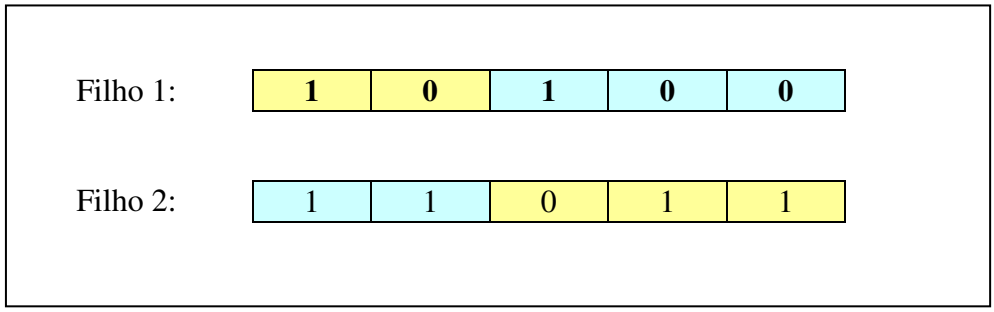

FIG. 9 Cromossomos filhos gerados pelo crossover de um ponto. Fonte: Próprio autor.

3.9.1.2 *Crossover* de dois Pontos

O *crossover* de dois pontos é bem similar ao *crossover* de um ponto. A diferença é que aquele define dois pontos de corte aleatoriamente, ao invés de um. O primeiro filho é então formado pela parte do primeiro pai fora dos pontos de corte e pela parte do segundo pai dentro do ponto de corte, o segundo filho será formado pelas partes restantes. Nas Figuras 10 e 11 os pontos de corte são representados pela seta indicativa.

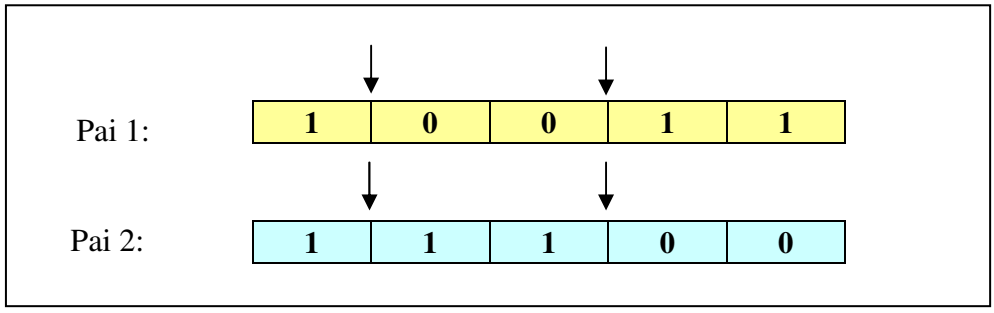

FIG. 10 Cromossomos pais antes do *crossover* de dois pontos. Fonte: Próprio autor.

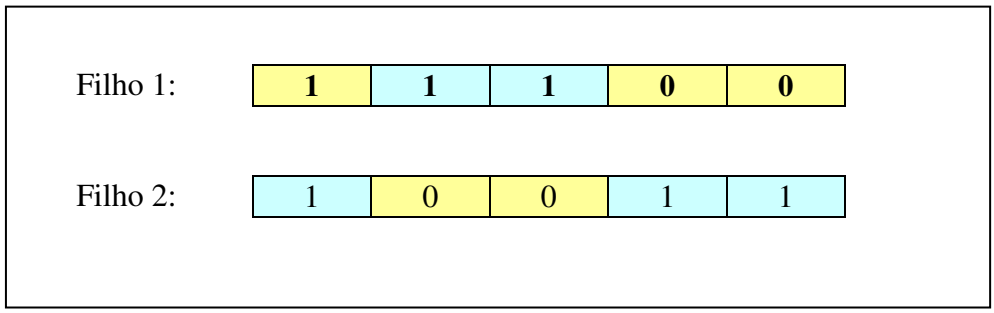

FIG. 11 Cromossomos filhos gerados pelo crossover de dois pontos. Fonte: Próprio autor.

3.9.1.3 *Crossover* Uniforme

O *crossover* uniforme não utiliza pontos de corte, o que ele faz é sortear um número, zero ou um, para cada gene que compõe o cromossomo. Ou seja, ele utiliza uma espécie de máscara para auxiliar no processo de *crossover*. Se o valor sorteado para determinado gene for igual a um, o primeiro filho recebe o gene da posição corrente do primeiro pai, e o segundo filho recebe o gene da posição corrente do segundo pai. Se o valor sorteado for zero, as atribuições ocorrem em ordem contrária. Um exemplo é mostrado nas Figuras 12 e 13:

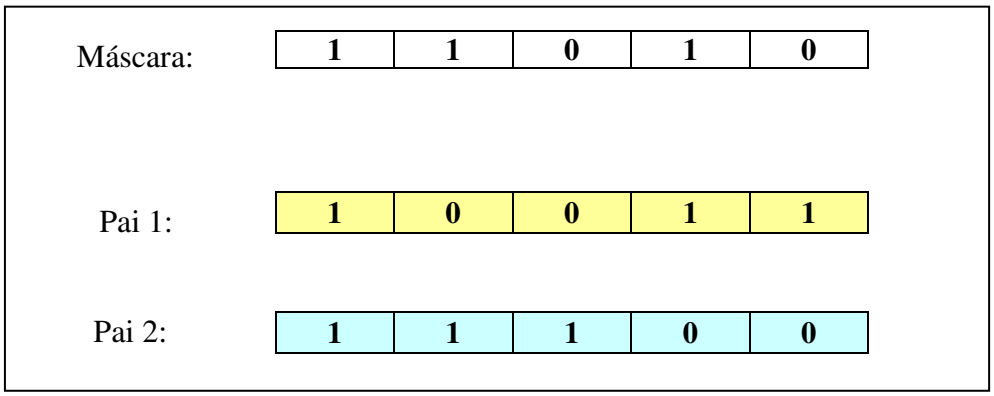

FIG. 12 Máscara e Cromossomos pais antes do *crossover* uniforme. Fonte: Próprio autor.

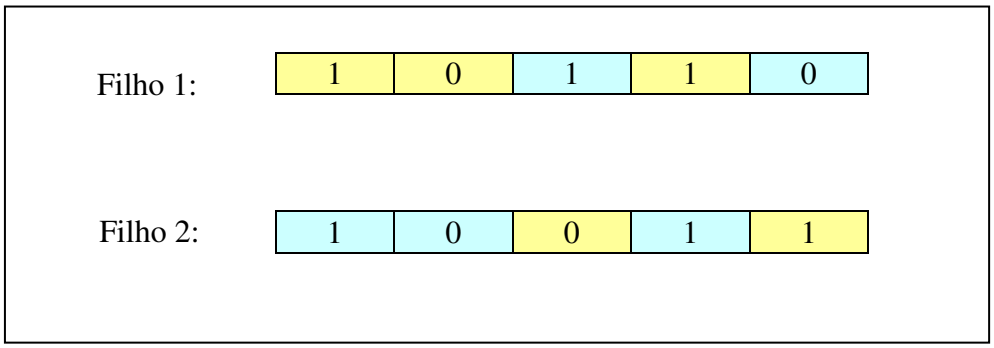

FIG. 13 Cromossomos filhos gerados pelo *crossover* uniforme. Fonte: Próprio autor.

3.9.2 Operador Mutação

O operador de mutação é fundamental para um algoritmo genético. Ele é o responsável pela diversidade genética da população, alterando arbitrariamente o valor de uma estrutura escolhida. Dessa forma, o operador mutação oferece meios para inserir novos membros na população, contornando o problema de ótimos locais.

O operador mutação tem associado uma taxa de probabilidade extremamente baixa (da ordem de 0,5 %)., o contrário do operador de cruzamento, que é aplicado com taxas de probabilidades altas (cerca de 90 a 99%). Para cada gene sorteia-se um número entre limites pré-estabelecidos, se o número sorteado for menor que a probabilidade pré-determinada então o operador mutação atua sobre o gene em questão alterando seu valor. Este processo continua até que se atinjam todos os genes do cromossomo.

A definição da taxa de mutação é muito importante, se o valor desta taxa for baixa demais, o AG agirá de forma extremamente parcimoniosa e a população não terá diversidade depois de um certo número de gerações, estagnando bem rápido devido à convergência genética. Por outro lado, se o operador de mutação receber uma taxa alta demais, o AG se comportará como um algoritmo aleatório (*random walk)* e perderá suas características interessantes (LINDEN, 2008).

A Figura 14 mostra o efeito do operador mutação sobre um cromossomo, o círculo mostra o gene que sofre a mutação.

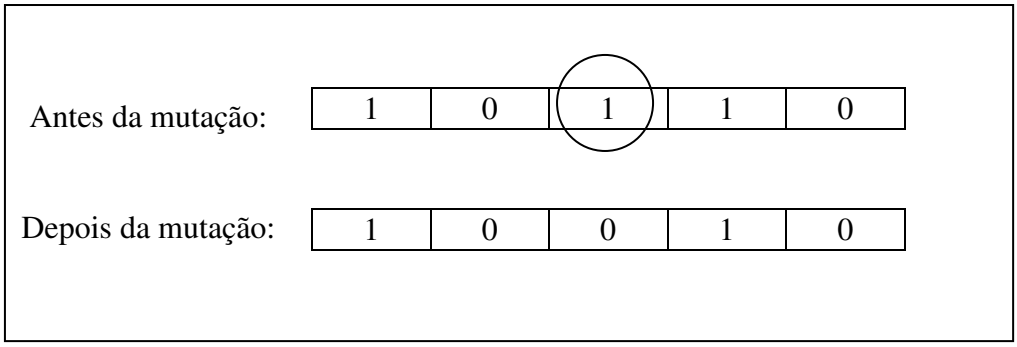

FIG. 14 Exemplo de aplicação do operador mutação. Fonte: Próprio autor.

3.10 Parâmetros de influência nos Algoritmos Genéticos

A correta definição dos parâmetros de influência é uma das mais importantes características dentro da estratégia dos algoritmos genéticos. A eficiência de execução e resultado do algoritmo genético é altamente dependente de seus parâmetros de controle. Os parâmetros básicos são: tamanho da população, probabilidade de Cruzamento e probabilidade de Mutação.

O tamanho da população deve ser definido com bastante cuidado, uma vez que o algoritmo genético é bastante sensível ao tamanho da população. Uma população pequena demais, não terá espaço para termos uma variabilidade genética suficientemente grande dentro da nossa população, o que fará que nosso algoritmo seja incapaz de achar boas soluções. Caso este número seja grande demais, o algoritmo demorará demais, e pode se aproximar de uma busca exaustiva (LINDEN, 2008).

Uma idéia bastante interessante e defendida por diversos autores é relacionar o tamanho da população ao tamanho do cromossomo, ou seja, quanto maior o cromossomo maior também deve ser o número de indivíduos que compõe a população.

A probabilidade de cruzamento indica qual a chance de dois indivíduos cruzarem e se reproduzirem. Quanto maior esta taxa, mais rapidamente novos indivíduos serão introduzidos na população. Por outro lado, indivíduos com boas aptidões serão mais rapidamente eliminados da população. Se esta taxa for muito baixa, o algoritmo pode se tornar muito lento.

A probabilidade de mutação indica com que probabilidade os indivíduos sofrerão mutações. Uma baixa taxa de mutação evita que a população entre em estagnação. Com uma taxa muito alta a busca tende a se tornar estreitamente aleatória, além de aumentar a chance de que uma boa solução do problema seja destruída.

Castro (2001 *apud* HESSER; MANNER, 1998) sugere que uma taxa ótima de mutação pode ser achada pela expressão:

$$P_m = (N.L^{1/2})^{-1}$$

sendo N é o tamanho da população e L o comprimento dos cromossomos.

No entanto, a taxa de mutação varia de acordo com o problema a ser desenvolvido, e na maioria dos casos varia de 0.001 a 0.1.

3.11 Conclusão

Por tratarem de problemas com grande espaço de busca, intratáveis por algoritmos determinísticos, AGs são ideais para o problema de formação de grupos de trabalhos. Um fator importante do AG é que este retorna um conjunto de soluções o que permite, neste caso, deixar a cargo do gerente de projeto escolher a solução adotada dentre as melhores apresentadas pelo AG.

Por outro lado, o AG pode exigir um tempo acentuado para retornar a solução do problema. O tempo requerido se justifica pelo fato do AG precisar efetuar constantes avaliações dos indivíduos para tomar as respectivas decisões no problema.

No próximo capitulo será apresentada a técnica utilizada para resolver o problema da formação de equipes de trabalho.

4 *SEARCH BASED SOFTWARE ENGINEERING* (SBSE) APLICADAS À ALOCAÇÃO DE RECURSOS PARA PROJETOS DE SOFTWARE

No processo de desenvolvimento de software, assim como as fases de testes e manutenção, a fase de planejamento é essencial para determinar o sucesso de um projeto. O bom gerenciamento de recursos é uma característica importante , e que deve ser levada em conta no planejamento do projeto. Existem técnicas e ferramentas tradicionais que visam auxiliar a gerência de projetos, no entanto estas técnicas não conseguem identificar a solução ótima pra o problema de alocação de recursos (ANTONIOL, PENTA e HARMAN, 2004).

Grandes projetos de software geralmente envolvem grande número de programadores e grupos de trabalhos. A alocação destes indivíduos a cada grupo não é tarefa, e fica ainda mais complexa quando são levados em conta vários aspectos e não apenas o fator técnico.

Uma abordagem da área de alocação de recursos para projetos de software é a formação de grupos de trabalhos, que é base do problema que será tratado aqui. O problema de alocação de recursos pode ser visto como uma instância do "*bin packing problem*", o qual a solução é do tipo NP-Dificil. Para resolução deste tipo de problemas, a SBSE tem se mostrado efetiva, retornado soluções ótimas ou próximas a ótimas, atendendo satisfatoriamente ao problema.

4.1 Formação de grupos de trabalhos para projetos de software

A formação de equipes de trabalhos não é uma tarefa fácil. Os softwares tem se tornado cada vez maiores e mais complexos, o que tem exigido cada vez um número maior de profissionais envolvidos.

Sallas (2003) define que uma equipe é um conjunto de duas ou mais pessoas distintas que interagem de forma dinâmica, independente e adaptativa em direção a um objetivo comum, valorizando meta, objetivo e missão, onde cada um destes possui a sua função específica, ou função a desempenhar, e que possui tempo de duração limitada.

O PMBoK (PMI, 2008), conhecido como guia de melhores práticas no que tange ao gerenciamento de projetos, possui uma área de conhecimento denominada

gerenciamento de recursos humanos. O PMBoK sugere técnicas e ferramentas para apoiar a atividade de planejamento de projeto e formação de equipes, tais como MS Project (Microsoft *Corporation*, 2010) e OpenProj (Serena Software, 2010). Porém estas ferramentas tradicionais apresentam limitação quanto ao escopo, principalmente no que se refere a recursos humanos. Diante disto, visando superar estas limitações, a abordagem utilizada neste trabalho visa possibilitar um maior grau de automatização, varrendo todo o espaço de busca o que gera resultados ótimos ou próximos a ótimos, satisfazendo às necessidades requeridas para o sucesso do desenvolvimento do projeto.

Diversos fatores influenciam na formação de equipes, tais como, tipo de produto a ser desenvolvido, as necessidades do cliente, e as característica do projeto, dentre outros. Em um projeto de software isto não é diferente, e tais fatores fazem com que surja a necessidade de novas abordagens para tratar do problema de formação de equipes.

Burdett *et al*. (1995) enfatiza a importância de utilizar não somente os aspectos técnicos, mas também as habilidades pessoais de cada participante na formação de uma equipe de projeto de software. A formação inadequada de uma equipe pode levar o projeto ao fracasso.

4.2 Trabalhos Relacionados

Vários trabalhos na literatura já foram realizados utilizando SBSE para tratamento de problemas da alocação de recursos a projetos de software. Porém, cada trabalho apresenta sua particularidade e pode ser aplicada a uma situação específica.

Antoniol, Penta e Harman (2004) realizou um trabalho aplicando algoritmos genéticos ao problema de alocação de equipes em projetos de manutenção de software. Ele considerou fatores imprevisíveis a que projetos estão sujeitos, tais como erro nas estimativas, abandono e retrabalho. Porém, o trabalho não considera outros fatores relevantes, tais como habilidades pessoais e experiência dos indivíduos.

Ferreira (1998) aplica algoritmos genéticos na formação de equipes de trabalho, levando em consideração as habilidades requeridas por cada tarefa, as habilidades disponíveis ao gestor de projeto e a preferência de cada pessoa por determinada tarefa. Através da utilização desta meta-heurística pode-se obter resultados

ótimos, onde os custos foram minimizados e as habilidades necessárias para cada tarefa foram conseguidas.

Lima (2006) aplica uma meta-heurística multiobjetivo para resolver problemas de formação de equipe para aprendizagem cooperativa apoiada por computador. O trabalho leva em consideração fatores pedagógicos na formação de grupos de trabalhos. Através da inteligência computacional e automatização são gerados grupos específicos para cada atividade.

Conforme mencionado, existem vários trabalhos envolvendo SBSE aplicados a formação de grupos de trabalhos, porém cada qual com sua particularidade. Este trabalho apresenta uma nova abordagem, levando em consideração os aspectos realísticos de um projeto de software. Detalhes da abordagem utilizada serão descritos no Capítulo 5.

5 FORMAÇÃO OTIMIZADA DE EQUIPES UTILIZANDO ALGORITMOS GENÉTICOS

Na abordagem aqui proposta para alocação de recursos a projetos de software, toma-se o problema de formação de equipes para determinada tarefa durante processo de desenvolvimento de software. Nesta abordagem pretende-se obter uma distribuição (quase-)ótima de pessoas a determinada atividade do processo de criação do software. A tarefa de otimização leva em consideração tanto aspectos técnicos quando aspectos humanos das pessoas envolvidas, aproximando-se assim dos problemas reais.

O problema de alocação de recursos pode ser definido como no trabalho de Burdett (1995):

Se um time de *r* pessoas é selecionado de um total de *n* pessoas disponíveis, o total de combinações possíveis será dado por.

$$\frac{n!}{r!\,(n-r)!}$$

Por exemplo, assumindo que há um total de 100 pessoas disponíveis:

· Um time de 5 membros terá 75,287,520 possíveis combinações

· Um time de 10 membros terá 17,310,309,456,440 possíveis combinações. (Ferreira, 1998).

O problema pertence a classe de problemas referenciados como NP - difíceis, onde o "tempo de execução para um algoritmo conhecido qualquer para garantir uma solução ótima é uma função exponencial do tamanho do problema" (EGLESE, 1990)

Na abordagem aqui apresentada será utilizada a meta-heurística algoritmos genéticos. Sua utilização justifica-se pelo fato de que estes procuram pela solução em um vasto espaço de busca, não ficando preso a ótimos locais. Dessa forma, os algoritmos genéticos retornam a solução ótima ou próxima a ótima, que é a característica desejável para o problema proposto.

O objetivo do problema tratado neste trabalho é retornar a melhor configuração de equipes, de modo a encontrar os melhores indivíduos para compor determinada equipe de trabalho. Para tal, durante o processo de formação de equipe, será levada em consideração a habilidade de cada integrante, a habilidade requerida por cada atividade em determinada tarefa e a preferência de agrupamento de cada individuo, ou seja, a afinidade entre eles. Esta última característica visa aumentar o grau de satisfação do indivíduo, melhorando o trabalho em equipe e evitando conflitos entre os componentes.

Além das características citadas acima, também será levando em conta o tamanho da equipe de trabalho, ou seja, a quantidade de membros na equipe. A intenção é formar equipes com o mínimo de pessoal possível (mínimo definido pelo gestor do projeto), porém este pessoal deve ser tecnicamente capaz de trabalhar no respectivo projeto. Ao formar equipes com o mínimo de pessoal possível, o gasto com o projeto tende a reduzir, uma vez que a profissionais geram custo às empresas. Mais adiante será mostrada a função de avaliação que foi modelada para representar as restrições e objetivo do problema.

5.1 Estratégias de Implementação

Para uma execução eficiente é necessário definir estratégias de implementação, tais como os parâmetros, modelagem e metodologias. As seções seguintes apresentam como a abordagem aqui apresentada foi tratada, como o problema foi modelado e como a implementação foi realizada.

5.1.1. Codificação do individuo

Uma das primeiras decisões que se deve tomar ao implementar um programa de computação evolutiva é sua representação em termos computacionais. Neste trabalho, cada cromossomo representará uma equipe para determinada atividade no projeto de desenvolvimento de software. O tamanho mínimo do cromossomo é definido pelo gestor do projeto de software. Cada "*gene*" do cromossomo representa uma pessoa da equipe.

P1	P2	P3	P4	P5

FIG. 15 Exemplo da representação de um indivíduo. Fonte: Próprio autor.

Como cada gene representa uma pessoa da equipe, o AG não permite que dois genes tenham o mesmo valor, ou seja, cada componente pode aparecer uma única vez em cada cromossomo.

Uma boa configuração do cromossomo é a que consegue, com o menor número de pessoas possível, maximizar as habilidades da equipe e ainda assim cumprir da melhor forma as preferências por composição da equipe de cada participante, dado o grau de relevância dos parâmetros de habilidade e afinidade.

Pelo fato de querer minimizar o número de pessoas envolvidas na equipe, o cromossomo tem tamanho variável. No entanto este tamanho deve respeitar as restrições de número mínimos/máximos de pessoas requeridas para cada equipe.

Note que pode acontecer dos objetivos sendo otimizados serem conflitantes, ou seja, uma configuração de um indivíduo pode ser dita ótima levando em consideração os aspectos técnicos da equipe (isto é, estes atendem plenamente os objetivos requeridos por determinada atividade) mas, no entanto, pode ser que dois ou mais dos indivíduos selecionados não possuam a afinidade requerida. Desta forma, a solução do problema torna ainda mais complexa.

5.1.2. Inicialização da População

A inicialização da população pode ser basicamente dividida em duas etapas: definir o tamanho da população e os procedimentos realizados para inicializar a população.

Não existem estudos conclusivos sobre o tamanho ótimo da população (MITCHELL, 1996). No entanto há autores que defendem que o tamanho da população deve ser proporcional ao tamanho do cromossomo. Desta forma o tamanho da população foi definido baseado em testes.

Para realizar a inicialização da população a escolha aleatória de pessoas é adotada neste trabalho. Isto garante uma maior variabilidade genética e representa com maior fidelidade os aspectos da evolução natural. Desta forma é possível, através da recombinação genética, convergir para uma solução ótima ou próxima a ótima.

5.1.3. Função de Avaliação

A função de avaliação modelará o problema matematicamente, a fim de que o mesmo possa ser tratado no ambiente computacional. O problema levará em consideração três fatores: as preferências dos membros candidatos ao time, suas habilidades técnicas e o tamanho da equipe.

A fim de modelar o problema de preferência entre os componentes da equipe, as seguintes regras foram criadas (adaptadas de LIMA, 2006):

R 1: Não formar grupos onde não há nenhuma relação entre seus membros;

R2 : Sempre formar grupos satisfazendo pelo menos uma escolha (preferência por agrupamento) de todos os componentes;

R 3: Quando possível, formar grupos que preservem as preferências mútuas;

Usando as regras *R*, pode-se definir a seguinte função *S* [g]:

$$s[g]_R = \begin{cases} 0\text{ , se } \neg R_1(g) \vee \neg R_2(g) \\ 0{,}5,\ \text{se } R_1(g) \wedge R_2(g) \neg R_3(g) \\ 1\text{, caso contrário;} \end{cases}$$

Já o atributo “habilidade” de cada candidato é modelado pela função *J* (*T*) :

$$j[T] : \sum_{t \in T} h$$

onde:

***T*:** Conjunto de funcionários pertencentes à equipe selecionada;

***h*:** Quantidade de habilidades, que são necessárias determinada atividade, que o funcionário *t* possui;

Nesta função, caso o indivíduo possua a habilidade *h* seja requisitada por determinada atividade, esta será contabilizada.

Afim de garantir que o somatório de habilidade esteja no intervalo [0..1], a função j [*T*] é normalizada, gerando a seguinte função $G(\sigma)$:

$$G(\sigma) = (\sigma - min) / (max - min),$$

onde:

σ: é o valor do somatório gerado pela função j [*T*],

***min*:** representa o valor mínimo de habilidade que o indivíduo pode possui para a atividade *n*;

***max* :** representa o valor necessário para que o indivíduo atenda todas as necessidades de habilidades requeridas na atividade *n*;

Neste problema, o tamanho da equipe também deve ser minimizado, de acordo com o mínimo de integrantes definido pelo gestor do projeto de software. Para isto, foi adicionado à função de avaliação o parâmetro "ω". Este parâmetro assume um valor no intervalo [0..1] , e é utilizado para penalizar o indivíduo quando necessário. Neste caso, sempre que um indivíduo representar uma equipe cujo número de integrantes seja maior que o número mínimo desejado, uma penalidade é aplicada ao indivíduo.

A penalidade foi modelada de forma que, quanto mais distante do número mínimo aceitável o indivíduo esteja, maior será a penalidade aplicada e, quanto mais próximo, menor será a penalidade. Caso o indivíduo possua a quantidade mínima aceitável, nenhuma penalidade é aplicada. Isto força o AG a convergir para uma configuração cuja equipe represente o mínimo de integrantes necessários para o projeto.

O parâmetro"ω " é definido da seguinte forma:

$$\omega = 1 - (tamInd - tamMin) / (tamMax\text{-}TamMin) ,$$

onde:

tamInd: Tamanho do indivíduo avaliado;

tamMin: Tamanho mínimo aceitável do indivíduo;

tamMax: Tamanho máximo aceitável do indivíduo;

Estando definidos todos os componentes da função de avaliação do indivíduo, a função de avaliação utilizada neste AG é caracterizada pela a soma da $s[g]$ e $G[\sigma]$, incluindo uma penalização. Desta forma, a função de avaliação F é dada por:

$$F(s[g], G[\sigma]) = x \cdot s[g] + y\, G[\sigma] + \omega$$

onde,

ω : operador de penalização ao indivíduo. Penaliza o indivíduo quando este representa uma equipe maior que o mínimo aceitável;

x e y: são pesos que são atribuídos pelo gestor de projeto para favorecer uma das duas abordagens (habilidades técnicas ou afinidade entre os integrantes);

5.1.4 Método de Seleção

O método de seleção empregado neste algoritmo proposto foi o método de seleção por torneio binário. Este método foi escolhido por apresentar melhores resultados, se comparado a outros métodos, tais como a roleta viciada.

> [..] o método torneio com dois participantes costuma apresentar resultados melhores do que o método da roleta viciada, não sendo nem um pouco sensível a questão de escala da função de avaliação e superindivíduo, entre outros. (LINDEN, 2008, p.173).

O torneio binário escolhe aleatoriamente dois indivíduos da população, identifica o valor de aptidão para cada um destes. O vencedor do torneio é determinado pelo indivíduo que apresenta o melhor valor de aptidão. Escolhido o indivíduo, este realiza o cruzamento. Este processo é executado até que se atinja o tamanho de uma determinada população.

5.1.5 Operador de Cruzamento

O mecanismo de cruzamento utilizado no algoritmo genético proposto é o cruzamento uniforme. Por se tratar de cruzamento de cromossomos de tamanho variável, será realizada uma escolha aleatória do tamanho do novo cromossomo filho, sendo que este tamanho não poderá ser menor do que o do menor cromossomo e não poderá superar o tamanho do maior cromossomo.

Definido o tamanho do novo cromossomo, será criada uma máscara binária (zero ou um), para cada gene que compõe o cromossomo do indivíduo filho.

Pode acontecer, no processo de cruzamento, o surgimento de um cromossomo com valores de gene iguais (representando o mesmo indivíduo da equipe), o que não é permitido. Quando estão situação ocorrer será realizada uma reparação do indivíduo através da invocação à função "reparação" que é detalhada a frente. Feita a reparação, caso o tamanho do indivíduo seja menor que o número mínimo aceitável, o filho é descartado e novo processo de cruzamento é iniciado.

5.1.6 Operador de Mutação

Nos algoritmos genéticos, o operador de mutação possibilita restaurar a diversidade genética eventualmente perdida durante o processo evolutivo (GOLDBERG, 1989).

O operador mutação é empregado neste trabalho. Desta forma, um gene é escolhido aleatoriamente e é substituído por um outro membro disponível. Caso o membro escolhido já exista na equipe (cromossomo) o gene da mutação é excluído e é verificado se o tamanho do cromossomo atende à restrição do problema (mínimo de quantidade de membros por equipe). Caso atenda, a mutação é efetivada, caso contrário a mutação é refeita e um novo membro é escolhido para integrar a equipe (cromossomo).

5.1.7 Função de reparação

O processo de *crossover* ou mutação pode gerar um indivíduo (equipe) com repetição de gene (integrante), o que não é aceitável. Quando esta situação ocorrer a

função de reparação será invocada, e será responsável por efetuar a exclusão do gene duplicado. Após o indivíduo ter sido reparado, é verificado se o tamanho do cromossomo atende à restrição do problema (mínimo de quantidade de membros na equipe). Caso atenda, o cruzamento ou mutação são efetivados, caso contrário o processo (cruzamento) deve iniciar novamente.

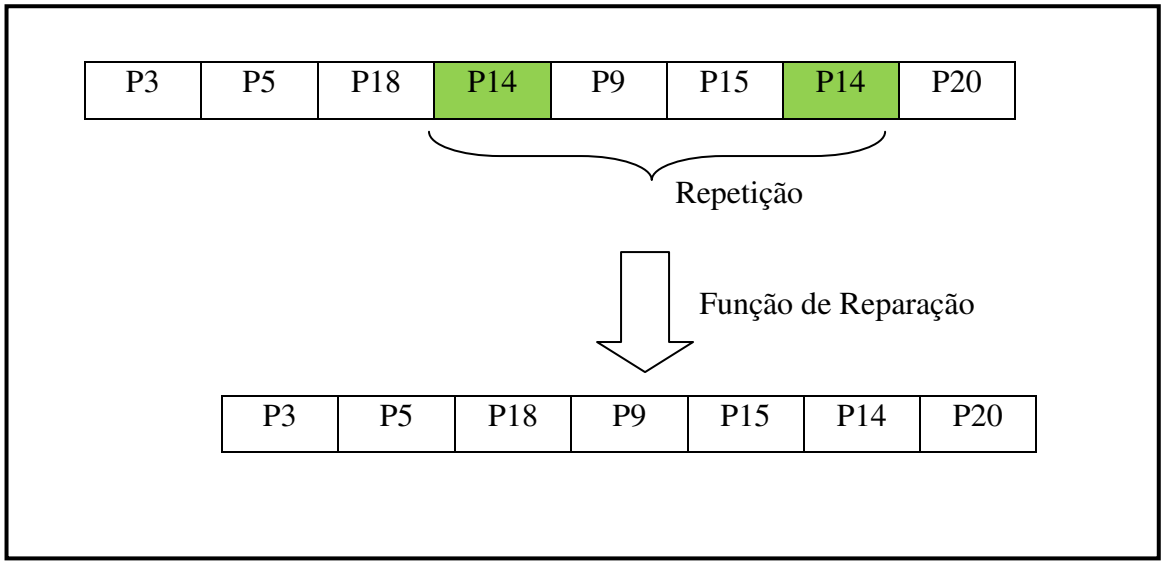

FIG. 16 Exemplo de um cromossomo sendo reparado. Fonte: Próprio autor.

5.1.8 Modelagem e Implementação

Quatro aspectos devem ser considerados na resolução do problema:

A.1 - As habilidades que cada individuo possui;

A.2- As habilidades requeridas por cada tarefa;

A.3- A existência de afinidade entre os indivíduos;

A.4 - A quantidade de membros na equipe.

A fim de calcular o função de avaliação de cada indivíduo serão utilizadas bases de dados geradas aleatoriamente para representar as características dos problemas reais. Esses dados são apresentados em tabelas auxiliares descritas no Apêndice A. Para tratar o 1º aspecto (A.1), considere a Tabela 3, nela é apresentada a relação entre pessoas e suas respectivas habilidades.

TABELA 3

Relação entre pessoas e suas respectivas habilidades

Pessoas/Habil	H1	H2	H3	H4	H5
P1	1	0	1	1	0
P2	0	1	0	0	1
P3	0	0	0	1	1
P4	1	1	1	0	0

Fonte: Próprio autor.

Nesta tabela, os dados informam se uma habilidade está associada a determinado indivíduo, ou seja, "1" que dizer que o individuo possui a habilidade, "0" indica que o indivíduo não possui a referida habilidade. Vale ressaltar que, caso haja habilidades comuns a um ou mais integrantes, estas habilidades comuns não tornarão o cálculo do *fitness* tendencioso, ou seja, a função *fitness* foi projetada de forma a tratar este tipo de situação.

Para tratar o aspecto 2 (A.2), uma tabela como a Tabela 4 será utilizada:

TABELA 4

Relação atividade x habilidade requerida

Atividade/Habil	H1	H2	H3	H4	H5
A	1	0	1	1	0

Fonte: Próprio autor.

Na Tabela 4 são mapeadas as habilidades necessárias para execução de determinada atividade. Pode-se fazer uma analogia a projetos reais, ou seja, o projeto "*x*" necessita de indivíduos que dominem a linguagem de programação *JAVA,* a linguagem de modelagem UML, a linguagem de banco de dados SQL e conheça de computação remota.

Uma tabela, como a Tabela 5, será utilizada para tratar o aspecto A.3.

TABELA 5

Relação entre afinidades dos Integrantes

Pessoas/Pessoas	P1	P2	P3	P4
P1	1	0	1	1
P2	0	1	0	0
P3	0	1	1	1
P4	1	0	1	1

Fonte: Próprio autor.

Na Tabela 5, a preferência por agrupamento (afinidade) de cada indivíduo é mapeada. Para cada pessoa (*P*), a relação *Pm x Pn* é construída. Assim caso a pessoa "P1" possua afinidade com a pessoa "P3", a informação desta preferência será representada pelo bit "1"e, caso contrário, pelo bit "0".

Por fim, para tratar o aspecto "A.4", quantidade de membros na equipe, será realizada a contagem de integrante na equipe, caso esteja maior que o mínimo aceitável, uma penalidade será aplicada.

6 SIMULAÇÕES E RESULTADOS

Esta seção mostrará simulações e resultados do algoritmo proposto. Serão apresentados alguns gráficos para melhor visualização.

6.1 Base de dados

Como mencionado anteriormente, a base de dados utilizados para simulações do algoritmo proposto é mostrada no Apêndice A. No Apêndice A, será detalhada a tarefa avaliada, as habilidades requeridas por determinada tarefa, as habilidade técnica do pessoal disponível e preferência por agrupamento de cada indivíduo.

As simulações executadas consideram um problema de alocação de equipe onde obedecendo a um número mínimo/máximo de pessoas. Essas pessoas serão escolhidas dentre um grupo de tamanho 30 pessoas. Para cada pessoa, são listadas 06 habilidades técnicas. Uma lista de afinidades de cada uma das 30 pessoas está disponível no Apêndice A.

6.2 Simulação e Resultados

Nesta seção serão mostradas as simulações de execução do AG proposto. Serão mostrados gráficos que permitirá a melhor visualização dos resultados obtidos. Em cada gráfico pode-se observar a função de avaliação do melhor indivíduo (linha azul) e o valor médio da função de avaliação dos indivíduos da população em determinada geração (linha vermelha). Em cada simulação será mostrado ainda o número de integrantes da melhor equipe encontrada.

Para as execuções neste AG o tamanho da população (30 indivíduos) e a taxa de mutação (0,5%) foram mantidas fixas, os demais parâmetros foram variados e, serão informados em cada simulação. O elitismo também foi implementado, ou seja, a cada geração o melhor indivíduo é preservado e transferido para a nova população.

6.2.1 Simulação 1

Nesta simulação foi definido o tamanho de geração como 5, o tamanho mínimo da equipe foi definido como 5 e o tamanho máximo como 10. Ou seja, o AG deve, através destes parâmetros, encontrar a melhor configuração de equipe, levando em conta a preferência por agrupamento, tamanho mínimo/máximo desta equipe, e habilidades técnicas requeridas por determinada tarefa. O Gráfico 1 mostra a evolução dos indivíduos no decorrer das 5 gerações. Como pode ser observado no Gráfico 1, a função de avaliação da população como um todo aumenta a cada geração. Há também uma evolução do melhor indivíduo.

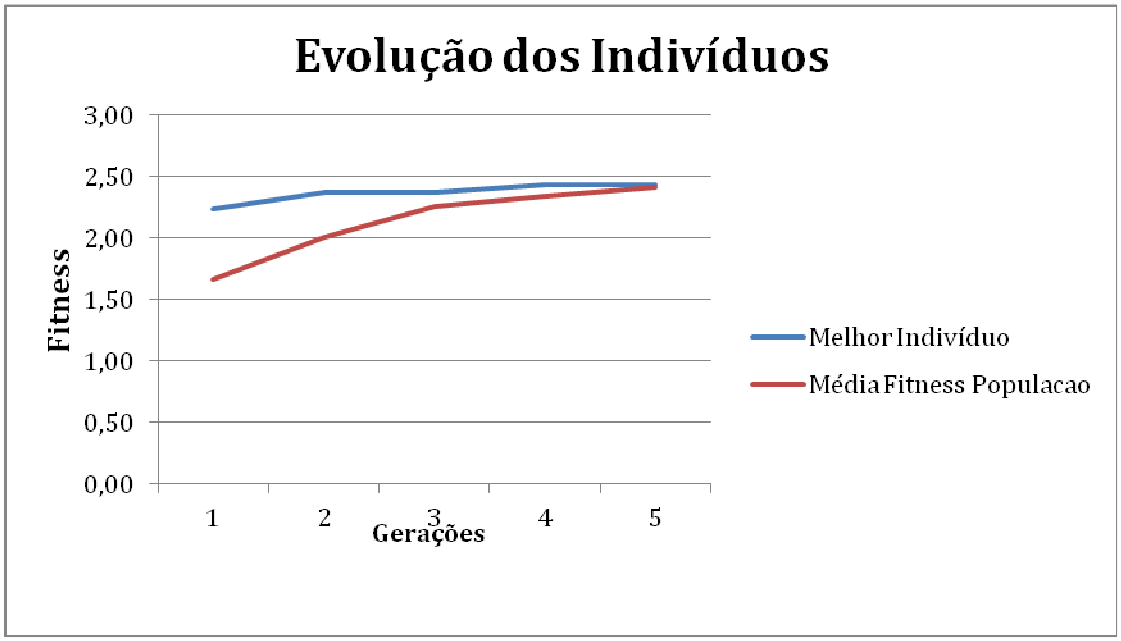

GRÁFICO 1: Relação habilidade x indivíduo –Simulação 1. Fonte: Próprio autor.

Ao final desta simulação, e de acordo com os parâmetros de seleção, o algoritmo encontrou a melhor equipe, que é composta dos seguintes integrantes: `16, 9, 8, 29 e 24.`

Verificando a base de dados do Apêndice A, pode-se observar que a equipe formada é realmente ótima, visto que o indivíduo 16 apresenta afinidade com todos os outros indivíduos da equipe, o indivíduo 9 apresenta afinidade com 2 indivíduos. Já o indivíduo 8 apresenta afinidade com três destes indivíduos, o indivíduo 29 apresenta afinidade com 1 indivíduo e o individuo 24 apresenta afinidade com 3 indivíduos da

equipe. Outra característica desta equipe é que ela foi formada com o mínimo de integrantes requeridos, o que se presume redução de custos, caso tratasse de projeto real. Percebe-se ainda que 4 dos indivíduos possuem todos as habilidades técnicas requeridas pela atividade a ser desenvolvida e que 1 indivíduo possui 2 das três habilidades requeridas. Outra característica importante desta equipe e que a preferência por agrupamento mútua foi atendida, ou seja, 3 dos integrantes possuem afinidades mútuas.

6.2.2 Simulação 2

Nesta simulação foi definido o tamanho de geração como 5, o tamanho mínimo da equipe foi definido como 3 e o tamanho máximo como 8. O Gráfico 2 mostra a evolução dos indivíduos no decorrer das 5 gerações. Novamente, como pode ser observado no Gráfico 2, o *fitness* da população como um todo aumenta a cada geração (ver linha vermelha). Ao final da última geração a população é composta por indivíduos cuja média do *fitness* é bem próxima ao *fitness* do melhor indivíduo. Percebe-se também uma evolução do melhor indivíduo.

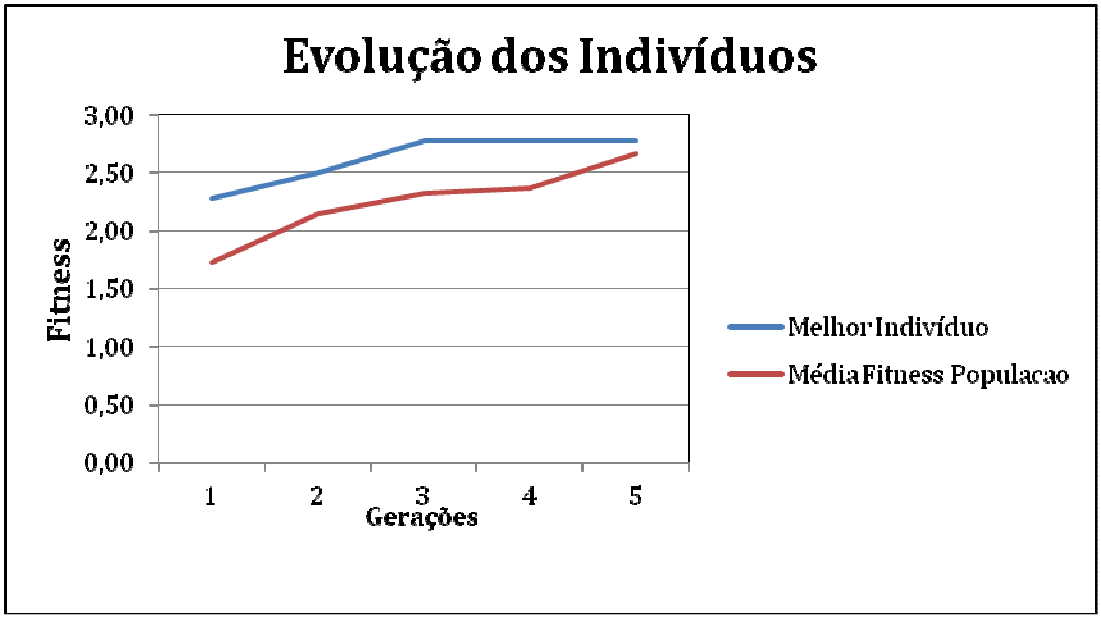

GRÁFICO 2: Relação habilidade x indivíduo –Simulação 2. Fonte: Próprio autor.

Nesta simulação, a melhor equipe é composta pelos indivíduos 18, 9, 11. Em análise ao Apêndice A, pode-se observar que esta equipe é composta por indivíduos que atendem as habilidades técnicas exigidas. Dois dos integrantes tem as preferências por agrupamento mútuo atendidas, e o tamanho da equipe é o menor requerido.

6.2.3 Simulação 3

Nesta simulação foi definido o tamanho de geração como 7, o tamanho mínimo da equipe foi definido como 8 e o tamanho máximo como 12. O Gráfico 3 mostra a evolução dos indivíduos no decorrer das 7 gerações. Nesta simulação, devido ao número de integrantes, repare que o melhor indivíduo é encontrado logo nas primeiras gerações, sendo que este evolui no decorrer da segunda geração, ficando constante e voltando a evoluir na 5ª geração. Também nesta simulação, ao final da última geração a população é composta por indivíduos cuja média do *fitness* é bem próxima ao *fitness* do melhor indivíduo.

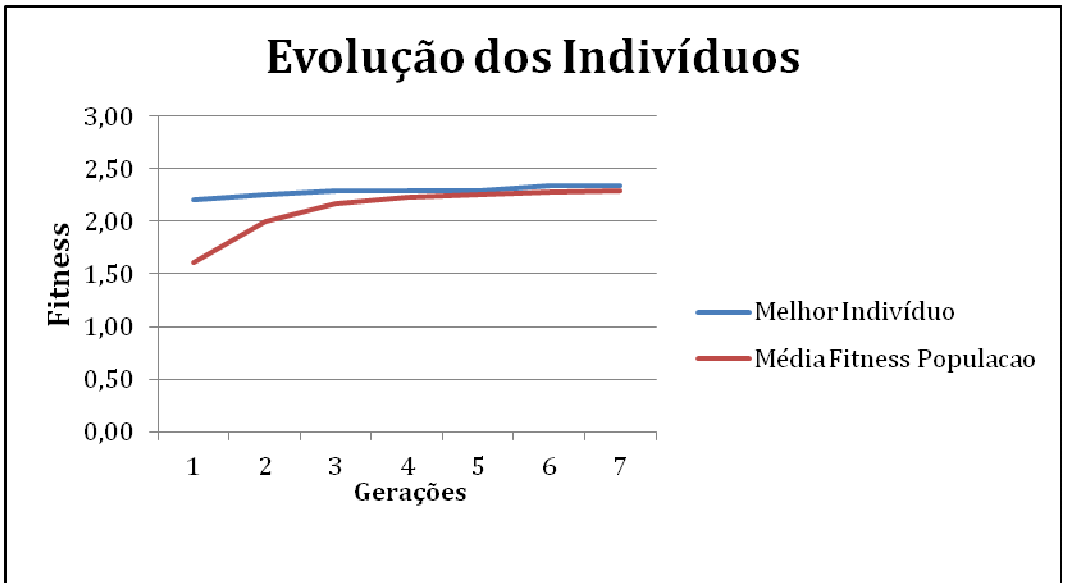

GRÁFICO 3: Relação habilidade x indivíduo–Simulação 3. Fonte: Próprio autor.

Nesta simulação, a melhor equipe é composta pelos indivíduos 29, 11, 2, 12, 24, 14, 8 e 10. Em análise ao Apêndice A, pode-se observar que todos os indivíduos atendem todas ou quase todas habilidades técnicas. Além disto, a equipe é composta por indivíduos que possuem afinidades entre si e, também, alguns possuem

afinidade mútua. Mais uma vez o AG retornou uma equipe com menor número de integrantes requeridos, com alta capacidade técnica e alto grau de afinidade.

6.2.4 Simulação 4

Nesta simulação foi definido o tamanho de geração como 5, o tamanho mínimo da equipe foi definido como 3 e o tamanho máximo como 4. O Gráfico 4 mostra a evolução dos indivíduos no decorrer das 5 gerações. Nesta simulação, o AG atingiu o *fitness* máximo, ou seja, encontrou a solução ótima para o problema. Repare que o AG converge rapidamente e, mesmo antes da última geração já encontrão melhor indivíduo. Ao final da última geração a população é composta por indivíduos cuja média do *fitness* é bem próxima ao *fitness* do melhor indivíduo.

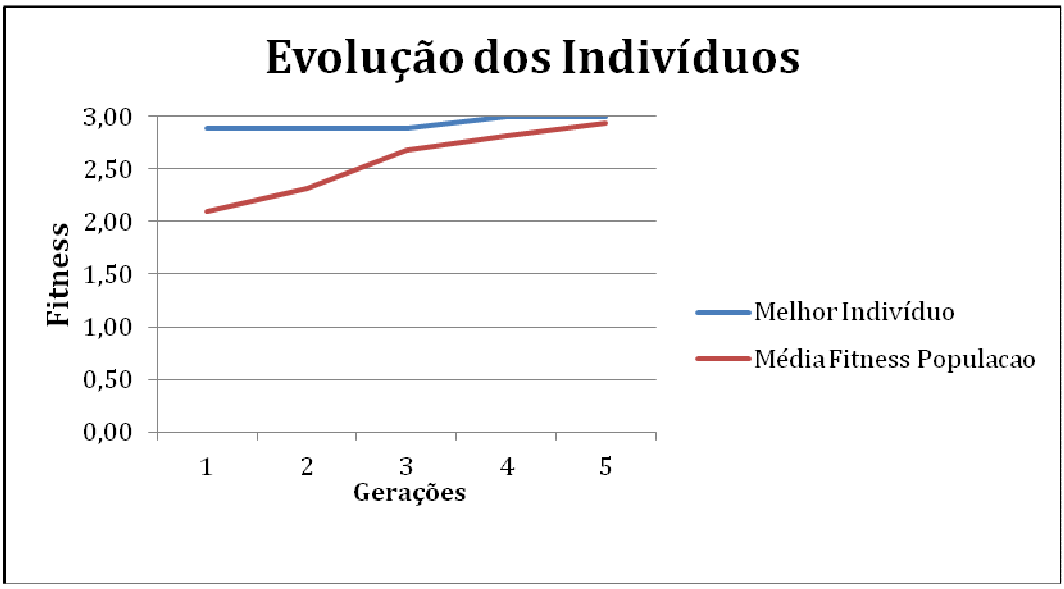

GRÁFICO 4: Relação habilidade x indivíduo –Simulação 4. Fonte: Próprio autor.

Nesta simulação, a melhor equipe é composta pelos indivíduos `24, 8 e 16`. Repare no Apêndice A, que tais indivíduos são os melhores e possuem todas as habilidades requeridas para a tarefa. Além disto, a equipe é composta somente por indivíduos que possuem afinidade mútua. Além destes aspectos a equipe retornada possui o menor número de integrantes requerido, ou seja, o possui o mínimo de pessoal definido pelo gestor do respectivo projeto.

7 CONCLUSÃO E TRABALHOS FUTUROS

A alocação de recursos a projetos de softwares não é uma tarefa trivial, sobretudo quando esta tarefa envolve vários aspectos, tais como o que foram abordados neste trabalho: habilidade técnicas, preferência por agrupamento de cada integrante e número de integrantes de cada equipe.

Esta dissertação envolveu o estudo sobre os algoritmos genéticos para alocação de recursos a projetos de softwares Foi apresentado primeiramente uma revisão teórica dos principais conceitos utilizados neste trabalho. Foi realizado, ainda, um estudo apresentando alguns trabalhos que propõem métodos para aplicação da SBSE e ao final deste foi apresentado o método proposto neste trabalho como sendo uma tarefa de otimização.

O problema de alocação de recursos foi modelado como um problema de otimização, e portanto foi escolhida uma técnica a de otimização para a sua resolução. A opção pelos AGs foi feita após um trabalho de revisão sobre os métodos de otimização na área de engenharia de software, e por serem mais indicados para tratarem problemas do tipo do abordado neste trabalho, onde os AGs apresentam resultados mais eficientes. Devido à natureza do problema verificaram-se várias vantagens no uso dos AGs.

O presente trabalho contribuiu para a tarefa, retornando resultados satisfatórios e eliminando o esforço humano que, muitas vezes, está sujeito a tendências pessoais.

De acordo com os resultados obtidos, uma boa abordagem para o problema é aquela que consegue equilibrar os valores para habilidades e preferências por agrupamento. O AG retorna um conjunto de soluções possíveis, e cabe ao gerente de projetos decidir qual equipe será utilizada. O gestor de projetos pode ainda alterar a função objetivo adicionando pesos ao aspecto que for mais interessante para seu projeto, assim o AG retornará um conjunto de indivíduos que possuem, primeiramente, a característica enfatizada pelo gestor de projeto.

Sugere-se para trabalhos futuros a formulação de mais fatores essenciais a projetos de softwares, tais como custo, tempo e preferência do indivíduo por terminada atividade. Com isto, espera-se abranger um maior número de critérios pré-formulados a serem adotados na formação otimizada de grupos. Pode-se também ser adotado como trabalhos futuros a utilização de novas meta-heurísticas, onde testes comparativos poderão ser realizados para comprovar a eficiência de cada técnica. Outro ponto a ser considerado como trabalhos futuros é o emprego de técnicas capazes de tratar

problemas multiobjetivos, sendo os objetivos conflitantes. Por último, tendo disponível uma base de dados real, testes podem comprovar a validade do método proposto e testado em dados sintéticos.

Tem-se a convicção que os objetivos propostos foram alcançados. Espera-se que este trabalho seja de grande valia para os gestores e engenheiros de softwares, assim como para os outros pesquisadores que trabalhem em áreas afins.

REFERÊNCIAS

ANTONIOL, G., PENTA, M. di, HARMAN, M., *Search-based Techniques for Optimizing Software Project Resource Allocation. Proceedings* of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04), Vol. 3103/2004, pp. 1425-1426, Seattle Washington USA, 26-30, 2004.

BAGNALL, A. , RAYWARD-SMITH V., WHITTLEY L., "*The next release problem*", Information and Software Technology, pp. 883–890..2001

BUCHER , A. et al. Ciclo de vida de desenvolvimento de sistemas/softwares. Disponivel em <http://www.hlera.com.br/clientes/ciclo_de_vida/index.php?s=introducao>. Acesso em 20 dez. 2011.

BURDETT, G. LI, R. K-Y "*A quantitative approach to the formation of workgroups*". Proceedings of the 1995 ACM SIGCPR conference on Supporting teams, groups, and learning inside and outside the IS function reinventing IS. p.202- 212, April 06-08 Nashville, Tennessee, United States. 1995.

BURGESS, C.J., M, LEFLEY ., "*Can genetic programming improve software effort estimation? A comparative evaluation*", Information and Software Technology, Dec. 2001, pp. 863–873.

CASTRO, R. E. Otimização de estruturas com multi-objetivos via algoritmos genéticos. 2001. 2006p. Tese (Doutorado em Ciências em Engenharia Civil) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2006.

EGLESE, R. "*Simulated Annealing - A Tool of Operational Research*", European Journal of Operational Research, Vol. 46, pp. 271- 281. 1990.

FERREIRA, J. S. Concepção de um ambiente multi-agentes de ensino inteligente integrando o paradigma de aprendizagem cooperativa. 1998. Dissertação (Mestrado em Engenharia da Eletricidade) – Programa de Pós-Graduação em Engenharia da Eletricidade - Universidade Federal do Maranhão, São Luís.

FREITAS, F.G. et al. Aplicação de Metaheurísticas em Problemas da Engenharia de Software: Revisão de Literatura. II Congresso Tecnologico Infobrasil. 2009. Ceará, Brasil.

FUGGETTA, A., "*Software process: a roadmap*", The Future of

GAREY M., JOHNSON, D. Computers and Intractability: a Guide to the Theory of NP-Completeness. W.H. Freeman, 1979.

Guides). Project Management Institute, 4a edição.

KIELIONG E., ROSA, R "Planager - um jogo apra apoio ao ensino de conceitos de ferência de projetos de software". Disponível em <http://www.inf.pucrs.br/~rafael/Planager/Jogo_GP.pdf. .2006>. Acesso em 17 jun. 2011.

LIMA, M. R. C.. Algoritmos Genéticos na Formação de Grupos para Aprendizagem Cooperativa Apoiada por Computador. 2006. Dissertação (Mestrado em Engenharia de Eletricidade) - Universidade Federal do Maranhão.

LINDEN, R. Algoritmos Genéticos: Uma importante ferramenta da Inteligência Computacional. 2.ed. Rio de Janeiro: Brasport,2008. 400p.

MACORATTI, J.C. O processo de software. Disponível em < http://www.macoratti.net/proc_sw1.htm> Acesso em 20 dez. 2011.

MAIA, H. Modelos prescritivos de desenvolvimento de software. Disponível em <http://hugohabbema.blogspot.com/2009/08/modelos-prescritivos-de-desenvolvimento.html>. Acesso em 05 jan. 2012.

MITCHELL, M.. "*An Introduction to Genetic Algorithms*" MIT Press Massachusetts, 1996.

PAPPA, G.L. Seleção de atributos utilizando algoritmos genéticos multiobjetivos. 2002. 75p. Dissertação (Mestrado em Informática Aplicada) – Pontifícia Universidade Católica do Paraná, Curitiba, 2002.

PMI. *A Guide To The Project Management Body Of Knowledge (PMBOK project resource allocation*. January 19, 2004. Disponível em: <http://www.springerlink.com/content/jwnf8qkp6afbpr8r/>. Acesso em 04 jan. 2012.

SALAS, E. et al. "*Toward an understanding of team performance and training, in Team: Their Training and Performance*".1992.

SCHWARTZ, J. I. *Construction of software*. In: Practical Strategies for Developing Large Systems. Menlo Park: Addison-Wesley, 1. ed. 1975.

SILVA ,E. Engenharia de Software. Disponível em http://imasters.com.br/artigo/3691/des_de_software/engenharia_de_software/. Acesso em 04 jan. 2012. Software Engineering, A. Finkelstein (ed), 2000.

YOO S., HARMAN M., "Pareto *Efficient Multi-Objective Test Case Selection*", Proceedings of the International Symposium on Software Testing and Analysis, 2007, pp. 140- 150.

APÊNDICE A

Base de dados

A base de dados utilizados para simulações deste algoritmo proposto é mostrada nas tabelas abaixo.

1) Para representar a atividade a ser executada, uma matriz foi criada conforme a Figura 1a.

0	0	1	1	0	1

FIG. 17a Exemplo de representação de uma atividade. Fonte: Próprio autor.

A matriz indica que para realizar a atividade em questão são necessárias as habilidades de índice 2, 3 e 5. Cada índice indica uma determinada habilidade. O gestor do projeto informa qual habilidade é necessária para que determinada atividade possa ser executada.

2) Para representar as habilidades que cada indivíduo possui, utilizou-se a matriz representada pela Tabela 1a.

TABELA 1a

Relação habilidade x indivíduo

Pessoas/Habilidade	H1	H2	H3	H4	H5
P0	0	0	0	1	0
P1	1	0	0	1	0
P2	1	1	0	1	0
P3	1	0	1	0	1
P4	0	1	0	1	0
P5	0	0	0	1	0
P6	0	0	0	1	0
P7	1	0	0	1	1
P8	0	0	1	1	0
P9	0	0	0	1	1
P10	0	1	1	1	0
P11	1	1	1	1	0
P12	0	1	0	1	0
P13	0	0	1	1	0
P14	0	1	0	1	0
P15	1	0	1	0	0
P16	0	0	1	1	0
P17	1	1	0	1	0
P18	0	1	1	0	0
P19	0	1	0	1	1
P20	0	0	0	1	0
P21	1	0	0	1	0
P22	1	0	0	0	1
P23	1	0	0	0	0
P24	0	0	1	1	0
P25	0	1	0	1	0
P26	1	0	1	0	0
P27	0	1	0	1	0
P28	0	0	0	1	0
P29	0	0	1	1	0

A matriz informa qual habilidade cada indivíduo possui. Por exemplo, o indivíduo P26, possui as habilidades H1 e H2.

3) Para representar as preferências de agrupamento de cada indivíduo, foi construída uma matriz representada pela Tabela 2a.

TABELA 2a

Relação das afinidades de cada indivíduo

Pessoas/Pessoas	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24	P25	P26	P27	P28	P29
P0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	0	1	0	1
P1	0	1	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	1	0	1
P2	0	1	0	1	1	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0	1	1	0	0	1	0	1	1	1	0
P3	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	1	1	0	1	0	0	0	1	0	0	1	0	1
P4	1	1	0	0	0	1	0	0	0	1	0	1	0	0	1	1	0	1	1	0	1	0	0	0	1	0	0	1	0	0
P5	1	0	0	1	0	1	0	0	1	1	1	0	0	1	1	0	0	0	1	0	1	0	0	0	1	0	0	1	0	1
P6	0	1	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0	1	1	1	0	0	0	1	0	0	1	0	1
P7	0	0	1	1	0	0	0	1	0	1	1	0	0	0	1	1	0	0	1	0	1	0	0	0	1	0	0	1	0	1
P8	1	0	0	1	0	1	1	0	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0	1
P9	0	0	0	1	0	1	0	0	1	1	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	0	1	0	0
P10	0	0	0	1	0	1	0	1	0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	1	0	1
P11	0	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0	1
P12	0	1	0	1	0	1	0	0	0	1	0	0	1	0	1	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1
P13	1	0	0	1	0	0	0	1	0	0	0	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	0	1	0	1
P14	0	1	0	1	0	1	1	0	0	1	0	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
P15	1	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	0	1	0	1
P16	0	0	1	0	1	1	0	1	1	1	0	0	1	0	1	0	0	0	1	0	1	0	0	1	1	1	0	1	0	1
P17	1	1	0	1	1	1	0	0	1	1	0	0	0	1	1	1	0	0	1	0	0	0	0	1	1	1	0	1	0	0
P18	0	0	0	1	1	1	0	0	1	1	0	1	0	0	1	0	0	0	1	0	0	0	0	1	1	0	0	1	0	0
P19	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1
P20	0	0	0	1	0	1	0	0	1	1	0	1	1	1	0	1	1	0	1	0	1	0	1	0	0	0	0	1	0	1
P21	1	0	0	1	0	0	1	1	1	0	1	0	0	0	0	0	1	0	1	1	1	1	0	1	0	0	1	0	1	1
P22	0	1	1	0	0	0	0	0	1	0	0	1	0	0	1	1	1	1	0	1	1	1	0	1	0	1	0	1	0	0
P23	1	0	0	1	1	0	1	1	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	0	0	0	0	1	0	1
P24	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	1	1	0	1	0	1	1	0	1	1	0	0	1	0	1
P25	0	0	1	1	0	0	0	1	0	1	0	1	0	1	0	0	1	1	0	1	1	0	1	1	1	0	0	0	0	1
P26	0	1	1	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0
P27	0	0	1	0	0	0	1	1	1	0	1	0	1	1	1	1	1	0	1	1	0	1	0	1	0	0	1	0	0	1
P28	1	0	0	0	0	1	1	1	1	1	0	1	1	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	1	1
P29	0	1	0	1	1	1	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	1

A Tabela 2a informa sobre a preferência de agrupamento de cada pessoa, ou seja, informa se o indivíduo *n*, possui afinidade com um indivíduo *m*. Por exemplo, a pessoa P24, tem preferência por agrupamento com a pessoa P0, P3, P4, P6, P7, P12,P14, P20, P21, P22, P27 e P29.