Metaheurística MOVNS para o Problema de Desenvolvimento de Cronogramas de Projetos de Software

Sophia Nóbrega[1], **Sérgio R. de Souza**[1], **Marcone J. F. Souza**[2]

[1]Centro Federal de Educação Tecnológica de Minas Gerais (CEFET/MG)
Av. Amazonas, 7675 – 30.510-000 – Belo Horizonte – MG – Brasil

[2]Departamento de Computação
Universidade Federal de Ouro Preto (UFOP)
Campus Universitário – 35.400-000 – Ouro Preto – MG – Brasil

sophia@dcc.ufmg.br, sergio@dppg.cefetmg.br, marcone@iceb.ufop.br

Abstract. *Development schedule in large-scale software projects is a very challenging problem. The Problem of Developing Schedules (PDS) is a NP-hard problem, since it comprises the Resource Allocation Problem (RAP) and Resource-Constrained Project Scheduling Problem (RCPSP). Many researchers have driven efforts to apply Search-Based Software Engineering (SBSE) techniques to solve this problem. This work presents an approach based on the MOVNS (MultiObjective Variable Neighborhood Search) metaheuristic for solving PDS. The results are compared against a literature algorithm NSGA-II. Computational experiments show that proposed MOVNS is statistically better than NSGA-II in relation to the cardinality and hypervolume metrics.*

Resumo. *O Desenvolvimento de Cronogramas em grandes projetos de software é um problema extremamente desafiador. O Problema de Desenvolvimento de Cronograma (PDC) é um problema NP-Completo, já que trata a junção do Problema de Alocação de Recursos com o Problema de Escalonamento de Tarefas com Restrição de Recursos. Muitos pesquisadores têm dedicado seus esforços aplicando técnicas de otimização em Engenharia de Software (SBSE* - Search-Based Software Engineering*) para resolver esse problema. Para resolver o PDC, esse artigo apresenta uma abordagem baseada na metaheurística MOVNS (*MultiObjective Variable Neighborhood Search*). Os resultados obtidos são comparados com um algoritmo NSGA-II existente na literatura. Os experimentos computacionais executados mostram que o algoritmo MOVNS proposto é estatisticamente melhor que o algoritmo NSGA-II em relação às métricas de cardinalidade e hipervolume.*

1. Introdução

O Desenvolvimento do Cronograma do projeto é uma atividade crucial na prática da engenharia de software, já que o orçamento total e os recursos humanos disponíveis devem ser gerenciados eficientemente para se obter um projeto de sucesso. As empresas, para se manterem competitivas, estão sempre preocupadas em reduzir o custo e a duração de seus projetos, e esses dois objetivos são conflitantes.

Os atuais projetos de software normalmente demandam um gerenciamento complexo, envolvendo cronogramas e planejamentos. Existe a necessidade de controlar pessoas e o processo de desenvolvimento, de forma a alocar eficientemente os recursos disponíveis para executar as tarefas demandadas pelo projeto, sempre satisfazendo uma variedade de restrições.

A alocação de recursos e o escalonamento de tarefas com restrição de recursos são atividades que possuem diversos aspectos que precisam ser considerados, como a disponibilidade dos recursos; os prazos; os custos; as interdependências entre as tarefas; os replanejamentos; as estimações de custos; e as estimações de dimensão das tarefas. No presente estudo, somente serão considerados os recursos humanos envolvidos, já que o desenvolvimento de projeto de software pode ser considerada uma atividade essencialmente intelectual e social, conforme afirma [Colares 2010].

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

Dentro desse contexto, o presente trabalho apresenta importantes contribuições. Segundo o conhecimento dos autores desses artigo, é apresentada aqui a primeira abordagem baseada em otimização multiobjetivo usando a metaheurística *MultiObjective Variable Neighborhood Search* (MOVNS) para resolver o Problema de Desenvolvimento de Cronograma (PDC). A metaheurística MOVNS é baseada em um simples princípio: processo de busca local realizada em uma estrutura de vizinhança com mudanças sistemáticas. A segunda contribuição é a proposição de um conjunto de 9 estruturas de vizinhança com o objetivo de explorar todo o espaço de busca para o problema. Todos os movimentos são descritos na subseção 4.2.

Outra importante contribuição é a realização de uma análise estatística da abordagem proposta, comparando os resultados obtidos com o algoritmo NSGA-II, na implementação desse algoritmo proposta por [Colares 2010]. As comparações são feitas com algoritmo NSGA-II, pois grande parte das pesquisas em SBSE, como em [Alba and Chicano 2007, Antoniol et al. 2005, Penta et al. 2011, Barreto et al. 2008, Colares 2010], são feitas utilizando algoritmos genéticos. Para avaliar o algoritmo MOVNS proposto, foram realizados experimentos computacionais que mostram que o algoritmo MOVNS é estatisticamente melhor que o algoritmo NSGA-II em relação as métricas de cardinalidade e hipervolume.

2. Trabalhos Relacionados

Em [Antoniol et al. 2005] foram utilizados algoritmos genéticos, *Simulated annealing*, Busca Randômica e *Hill Climbing* para resolver o problema de alocação de pessoas. Nesse estudo, os autores também consideram problemas de retrabalho e abandono de projetos, aspectos importantes e comuns em projetos de Engenharia de Software.

Em [Alba and Chicano 2007], o Problema de Desenvolvimento de Cronograma de projetos foi tratado utilizando um algoritmo genético tradicional, que foi validado uti-

lizando dados fictícios obtidos a partir de um gerador automático de projetos. Os autores tratam dois objetivos: minimizar o tempo e o custo do projeto, que são combinados em uma única função objetivo, usando pesos diferentes.

Em [Gueorguiev et al. 2009] os autores apresentam a primeira formulação multi-objetivo para esse tipo de problema, no qual robustez e tempo de conclusão do projeto são tratados como dois objetivos concorrentes para resolver o problema de planejamento de projeto de software. O algoritmo proposto foi testado em quatro projetos reais, e os resultados indicam um bom desempenho da abordagem proposta.

Em [Penta et al. 2011] os autores apresentam uma modelagem mono-objetivo e outra multiobjetivo para o problema de cronograma e alocação de equipes em projetos de software. A abordagem mono-objetivo foi implementada, utilizando, para a solução, algoritmo genético, *Simulated Annealing (SA)* e *Hill Climbing*. Para validar a abordagem, foram realizados estudos empíricos utilizando dados reais de dois projetos. Os resultados mostram que o algoritmo genético e SA conseguem gerar soluções que podem ser valiosas para os gerentes de projeto no processo de tomada de decisão. A abordagem multiobjetivo foi modelada, mas não foi avaliada.

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

Em [Colares 2010] o autor utilizou um algoritmo genético multiobjetivo para resolver o problema de alocação de equipes e desenvolvimento de cronogramas. A função de aptidão proposta busca minimizar o tempo total do projeto, o custo total, o atraso nas tarefas e as horas extras.

Os autores de [Minku et al. 2012] apresentam a primeira análise de tempo de execução para o Problema de Desenvolvimento de Cronograma, definindo quais características podem transformar o Desenvolvimento de Cronograma em um problema fácil ou difícil. Baseado nessa análise é proposto um Algoritmo Evolucionário (AE), que é comparado com o resultado apresentado em [Alba and Chicano 2007].

Uma limitação das publicações que tratam o Problema de Desenvolvimento de Cronograma é a dificuldade de reproduzir os resultados apresentados pelos autores, já que as instâncias utilizadas não são totalmente divulgadas. Em geral, são descritas algumas características que as representam, mas que são insuficientes para reproduzir os resultados e realizar comparações. Dada essa limitação, outra importante contribuição desse trabalho é a publicação das 3 instâncias de projetos reais aqui utilizadas nos testes da abordagem proposta.

3. Formulação do Problema

O algoritmo proposto recebe, como parâmetros de entrada, tarefas e recursos humanos. As tarefas possuem, como atributos, esforço estimado, nível de importância e datas de

início e fim. Os recursos humanos são divididos em contratado e empregado. O primeiro possui, como atributos. valor hora e dedicação diária. O empregado possui os atributos salário, dedicação diária, valor hora extra e tempo máximo de hora extra. Ambos os tipos possuem um atributo que representa seu calendário de disponibilidade.

Cada recurso humano possui uma lista de habilidades individuais, com seu respectivo nível de proeficiência, e uma lista de tarefas, com seu respectivo nível de experiência. Assim, cada tarefa possui uma lista de habilidades necessárias para sua execução. Cada recurso é alocado a uma determinada tarefa em valores percentuais que variam de zero ao máximo permitido para o recurso.

Para definir a interdependência entre tarefas, ou sequenciamento, a abordagem proposta utiliza os quatro conceitos de relacionamento utilizados pela maioria das ferramentas de gerência de projeto: Início-Início (II), Início-Final (IF), Final-Início (FI) e Final-Final(FF). Além disso, o algoritmo reajusta o início de tarefas que não pertencem ao caminho crítico do projeto, evitando a quebra de restrição de recursos.

A produtividade de uma equipe $prod_{r,t}$ pode ser obtida pela expressão:

$$prod_{r,t} = x_{r,t}.r^{dedicacao}.\left(\prod_{s\in\left(S^r \bigcap S^t\right)} r^{proef}(\ s)\right).r^{exp}(\ t) \tag{1}$$

em que $x_{r,t}$ representa a proporção de esforço do recurso r para executar a tarefa t; $r^{dedicacao}$ representa a dedicação diária do recurso r em horas; S^r é o conjunto de habilidades que o recurso r possui; S^t é o conjunto de habilidades requeridos pela tarefa t; $r^{proef}(s)$ representa o fator de ajuste devido à proficiência do recurso r na habilidade s; e $r^{exp}(t)$ representa o fator de ajuste, devido à experiência do recurso r na tarefa t. O tempo de duração de uma tarefa em dias $t^{duracao}$ pode ser obtido por:

$$t^{duracao} = \frac{t^{esforco}}{\sum_{r\in R} prod_{r,t}}, \forall t \in T \tag{2}$$

em que $t^{esforco}$ é o esforço da tarefa em Pontos de Função (PF). Na Figura 1, tem-se a representação de uma matriz de produtividade $prod_{r,t}$ e o cálculo de duração para uma tarefa do projeto.

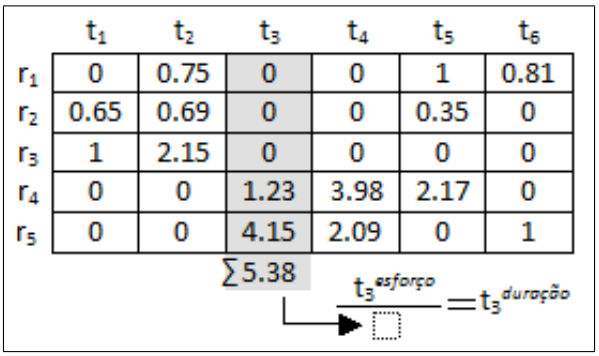

Figura 1. Representação do cálculo de duração das tarefas do projeto.

Estimado o tempo de duração de cada tarefa $(t^{duracao})$, é calculada a duração total

do cronograma do projeto, ou makespan, representada por:

$$S = makespan \quad (3)$$

A minimização do makespan é o primeiro objetivo proposto.

O segundo objetivo proposto é minimizar o custo total do projeto, dado por:

$$C = \sum_{i=1}^{R}\sum_{j=1}^{T}(x_{ij}.r_i^{dedicacao}.r_i^{valorHora}.t_j^{duracao}) \quad (4)$$

O custo total é, portanto, a soma do pagamento dos recursos por sua dedicação no projeto. Esse custo é calculado multiplicando-se o salário pago por hora para o empregado pelo seu tempo dedicado ao projeto. O tempo dedicado ao projeto é calculado pela soma da dedicação do recurso multiplicado pela duração de cada tarefa.

4. Modelo Heurístico

4.1. Representação da Solução

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

Uma dimensão da Matriz de Alocação representa os recursos humanos disponíveis, dados por $\left\{r_1, r_2, ..., r_{|R|}\right\}$, e a outra as tarefas que devem ser executadas, dadas por $\left\{t_1, t_2, ..., t_{|T|}\right\}$. Na matriz X, cada variável $x_{r,t}$ recebe o valor -1 ou um valor inteiro entre 0 e a dedicação diária de cada recurso, acrescida do tempo máximo de hora extra, se for o caso. Esse valor inteiro é interpretado dividindo-o pela dedicação diária, de forma a se obter porcentagens de 0 a 100%, que representam o esforço dedicado pelo recurso r na execução da tarefa t. Quando o percentual for superior a 100%, indica a realização de hora extra. Quando a variável $x_{r,t}$ recebe o valor -1, representa a incompatibilidade entre o recurso r e a tarefa t, ou seja, o recurso não pode executar a tarefa pois não possui a habilidade requerida.

O vetor Cronograma possui dimensão T, sendo T o total de tarefas. Os índices do vetor representam as tarefas, e cada posição do vetor é preenchida por um valor real, que indica o tempo de início de execução da tarefa. Na Figura 2, tem-se um exemplo de uma solução para o PDC.

4.2. Estruturas de Vizinhança

Para explorar o espaço de soluções, foram propostos e utilizados 9 movimentos, descritos a seguir:

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

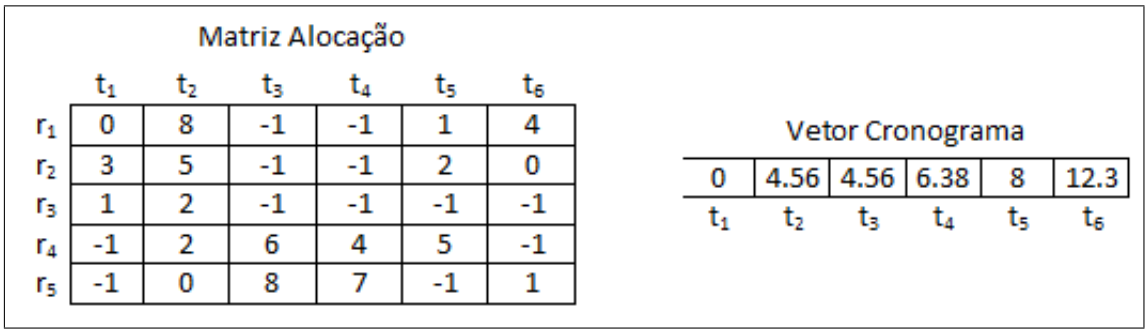

Figura 2. Representação de uma solução do problema.

- **Movimento Realocar Recurso de uma Tarefa -** $M^{RT}(s)$**:** este movimento consiste em selecionar duas células (x_{it} e x_{kt}) da matriz X e repassar a dedicação de x_{it} para x_{kt}. Assim, a dedicação de um recurso i é realocada para um recurso k que esteja trabalhando na tarefa t. Restrições de compatibilidade entre recursos são respeitadas, havendo realocação apenas quando houver compatibilidade.
- **Movimento Desalocar Recurso de uma Tarefa -** $M^{DT}(s)$**:** consiste em selecionar uma célula x_{rt} da matriz X e zerar seu conteúdo, isto é, retirar a alocação de um recurso r que estava trabalhando na tarefa t.
- **Movimento Desalocar Recurso no Projeto -** $M^{DP}(s)$**:** consiste em desalocar toda a dedicação de um recurso r no projeto. O movimento retira todas as alocações do recurso r, que deixa de trabalhar no projeto. O recurso volta a trabalhar no projeto assim que uma nova tarefa for associada a ele.
- **Movimento Dedicação de Recursos -** $M^{DR}(s)$**:** este movimento consiste em aumentar ou diminuir a dedicação de um determinado recurso r na execução de uma tarefa t. Neste movimento, uma célula x_{rt} da matriz X tem seu valor acrescido ou decrescido em uma unidade.
- **Movimento Troca de Recursos entre Tarefa -** $M^{TB}(s)$**:** duas células x_{ri} e x_{rk} da matriz X são selecionadas e seus valores são permutados, isto é, os recursos que trabalham nas tarefas i e k são trocados. Restrições de compatibilidade entre recursos e tarefas são respeitadas, havendo troca de recursos apenas quando houver compatibilidade.
- **Movimento Troca de Recursos de uma Tarefa -** $M^{TO}(s)$**:** duas células x_{it} e x_{kt} da matriz X são selecionadas e seus valores são permutados, isto é, os recursos i e k que trabalham na tarefa t são trocados. Restrições de compatibilidade entre recursos são respeitadas, havendo realocação apenas quando houver compatibilidade.
- **Movimento Insere Tarefa -** $M^{IT}(s)$**:** este movimento consiste em inserir uma tarefa que está em uma posição i em outra posição j do vetor Y. Esse movimento é realizado somente entre tarefas sem relações de precedência. Caso o movimento quebre alguma restrição de recurso, a nova solução gerada é descartada.
- **Movimento Troca Tarefa -** $M^{TT}(s)$**:** consiste em trocar duas células distintas y_i e y_k do vetor Y, ou seja, trocar o tempo de início de execução das tarefas i e k. Os movimentos de trocas serão feitos sempre respeitando a ordem de precedência para executar as tarefas. Caso esse movimento quebre alguma restrição de recurso, a nova solução gerada é descartada.

5. Algoritmo Proposto

Nesse artigo, é proposto um algoritmo multiobjetivo nomeado GRASP-MOVNS, que consiste na combinação dos procedimentos Heurísticos *Greedy Randomized Adaptative Search Procedure* - GRASP e *Multiobjective Variable Neighborhood Search* - MOVNS [Geiger 2008] [Souza et al. 2010]. O algoritmo GRASP-MOVNS foi implementado utilizando a mesma estratégia proposta por [Coelho et al. 2012].

O algoritmo 1 apresenta o pseudocódigo do algoritmo GRASP-MOVNS. Na linha 2 é gerado o conjunto inicial de soluções não dominadas através do procedimento parcialmente guloso $constroiSolucaoInicialGRASP$, mostrado no Algoritmo 2. São geradas duas soluções iniciais s_1 e s_2, que são construídas utilizando, como regra de prioridade, respectivamente, as tarefas de maior duração e as tarefas que pertencem ao caminho crítico.

Na linha 4 do algoritmo 2 é realizada a busca local. A busca local foi implementada utilizando a heurística *Variable Neighborhood Descent* - VND [Hansen and Mladenovic 1997], que envolve a substituição da solução atual pelo resultado da busca local, quando há uma melhora; porém, a estrutura de vizinhança é trocada de forma determinística, cada vez que se encontra um mínimo local. A solução resultante é um mínimo local em relação a todas as nove estruturas de vizinhanças utilizadas: M^{IT}, M^{TT}, M^{DR}, M^{RD}, M^{RT}, M^{DP}, M^{DT}, M^{TB}, M^{TO}.

Nas linhas 6 e 7 do Algoritmo 1 é feita a seleção de um indivíduo do conjunto de soluções não-dominadas D, marcando-o como "visitado". Quando todos os indivíduos estiverem marcados como visitados, a linha 32 retira estes marcadores. As variáveis $level$ e $shaking$ (linhas 3 e 4 do Algoritmo 1) regulam a perturbação utilizada no algoritmo. Esta versão do algoritmo MOVNS, proposta por [Coelho et al. 2012], possui um mecanismo que regula o nível de perturbação do algoritmo, ou seja, a variável *shaking* é incrementada quando o algoritmo passa um determinado tempo sem obter boas soluções. Da linha 9 à linha 12 do Algoritmo 1, ocorre o laço de perturbação do algoritmo.

O algoritmo $adicionarSolucao$ 3, acionado na linha 16, adiciona, ao conjunto solução D, as soluções geradas pelo GRASP-MOVNS. No mecanismo utilizado, quanto maior o valor da variável *shaking*, maior será a intensidade da perturbação na solução. Para cada unidade dessa variável, aplica-se um movimento aleatório dentre as nove vizinhanças: M^{IT}, M^{TT}, M^{DR}, M^{RD}, M^{RT}, M^{DP}, M^{DT}, M^{TB}, M^{TO}. A variável *level* regula quando a variável *shaking* será incrementada. As linhas 24 e 25 retornam os valores das variáveis *level* e *shaking* para uma unidade, quando, pelo menos, uma solução é adicionada ao conjunto eficiente D.

O algoritmo proposto GRAPS-MOVNS é comparado ao algoritmo NSGA-II utilizado por [Colares 2010] para resolver o PDC. Em sua pesquisa, [Colares 2010] implementou a versão clássica do algoritmo NSGA-II, originalmente proposta por [Deb et al. 2002]. O NSGA-II utiliza, como operador de seleção, a versão clássica do Torneio Binário e, para mutação, foi implementada a versão clássica do *Bit Flip*. O operador de cruzamento foi implementado exclusivamente para o PDC, pois o indivíduo é representado por uma matriz em vez de um vetor. A implementação do operador de cruzamento bidimensional foi baseada no cruzamento tradicional *Single Point*. Para isso, é selecionado um ponto aleatório (x, y) e, a partir dele, trocadas as informações genéticas

Algoritmo 1: GRASP-MOVNS

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

início
  $D \leftarrow constroiSolucaoInicialGRASP(graspMax)$
  $level \leftarrow 1$
  $shaking \leftarrow 1$
  enquanto (*Critério de parada não satisfeito*) **faça**
    Seleciona uma solução não visitada $s \in D$
    Marque s como visitada
    $s \leftarrow s'$
    para $i \rightarrow shaking$ **faça**
      Selecione aleatóriamente uma vizinhança $N_k(.)$
      $s' \leftarrow$ *Pertubação(*s', *k)*
    fim
    $k_{ult} \leftarrow k$
    $incrementa \leftarrow verdadeiro$
    para $s'' \in N_{k_{ult}}(s')$ **faça**
      $adicionarSolucao$(D, s", f(s"), added)
      se $adicionado = verdadeiro$ **então**
        $incrementa \leftarrow falso$
      fim
    fim
    se $incrementa = verdadeiro$ **então**
      $level \leftarrow level + 1$
    senão
      $level \leftarrow 1$
      $shaking \leftarrow 1$
    fim
    se $level \geq levelMax$ **então**
      $level \leftarrow 1$
      $shaking \leftarrow shaking + 1$
    fim
    se *todo $s \in D$ estão marcadas como visitadas* **então**
      Marque todos $s \in D$ como não visitado
    fim
  fim
  Retorne D
fim

Algoritmo 2: constroiSolucaoInicialGRASP

Entrada: $graspMax$
Saída: *Aproximação de um conjunto eficiente D*

início
  $f(s*) \leftarrow \infty$
  para $i \rightarrow graspMax$ **faça**
    $s \leftarrow constroiSolucaoInicial()$
    $s \leftarrow BuscaLocal()$
    se $f(s) < f(s*)$ **então**
      $s* \leftarrow s$
      $f(s*) \leftarrow f(s)$
    fim
  fim
  $s \leftarrow s*$
  Retorne s
fim

dos indivíduos pais, formando-se dois novos indivíduos filhos.

5.1. Experimentos e Análise

Os algoritmos foram implementados em Java, e os experimentos realizados em um notebook Dell Inspirion 14 Core i3-3110M, 3 MB Cache, 2.4 GHz, 4GB de RAM, sob sistema

Algoritmo 3: adicionarSolucao

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

```
1  início
2      adicionado ← verdadeiro
3      para s ∈ D faça
4          se z(s) ≤ z(s') então
5              adicionado ← falso
6              break
7          fim
8          se z(s') < z(s) então
9              D = D s
10         fim
11     fim
12     se adicionado = verdadeiro então
13         D = D ⋃ s'
14     fim
15     Retorne D, adicionado
16 fim
```

operacional Windows 7. Para realizar os testes, foi utilizado um conjunto de 3 instâncias, disponível em `www.decom.ufop.br/prof/marcone/projects/SBSE.html`. As instâncias 1 e 2 são as mesmas utilizadas em [Colares 2010] e a instância 3 foi gerada para os testes realizados nesse estudo. Os dados da instância 3 são de um projeto real de uma empresa de desenvolvimento de sistemas de Belo Horizonte, que está no mercado a mais de 8 anos.

Após a realização de testes iniciais, foram definidos os seguintes valores para os parâmetros do algoritmo GRASP-MOVNS: $graspMax = 200$ e $levelMax = 10$. A variável $shaking$ teve seu valor máximo fixado em 10. O algoritmo NSGA-II foi executado com os mesmos parâmetros definidos em [Colares 2010]: população inicial com 200 indivíduos, a probabilidade de mutação de 1/*lengh* e probabilidade de cruzamento de 90%. Cada algoritmo foi executado 30 vezes, cada qual por 180 segundos, assim como em [Colares 2010].

5.2. Métricas de Avaliação de Desempenho

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

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

5.3. Análise dos Resultados

As tabelas a seguir apresentam os resultados obtidos pelos algoritmos GRASP-MOVNS e NSGA-II, considerando a aplicação das métricas de hipervolume e cardinalidade. Para

cada métrica, é apresentado o resultado médio e o melhor resultado. O valor do resultado médio de cada instância é obtido pela média aritmética entre as 30 execuções. Analogamente, o valor para o melhor resultado é obtido considerando apenas o melhor resultado para cada instância, em relação a cada métrica.

Na Tabela 1 estão os resultados obtidos pelos algoritmos implementados neste trabalho (GRASP-MOVNS, e NSGA-II) com relação ao número de soluções não dominadas geradas por cada um deles. As três primeiras colunas detalham o tamanho da instância, sendo T o total de tarefas, R o total de recursos e H o total de habilidades. Na quarta coluna é apresentado o total de soluções distintas que compõem o conjunto de referência REF. O conjunto REF é gerado por 30 execuções de 2 horas de cada algoritmo, mais o conjunto de soluções gerado durante os demais testes. Nas demais colunas, mostra-se, para cada algoritmo e para cada instância, o número total de soluções não-dominadas geradas pelo respectivo algoritmo em 30 execuções. Na última linha, totalizam-se os resultados de cada algoritmo.

Tabela 1. Número de soluções não-dominadas

T	R	H	REF	Algoritmo	
				GRASP-MOVNS	**NSGA-II**
27	18	12	91	**88**	0
72	21	7	574	**475**	0
100	16	4	689	**558**	189
Total			1354	**1121**	189

Considerando que, ao todo, 1121 soluções do conjunto de referência foram geradas pelo algoritmo GRASP-MOVNS, de um total de 1354, pode-se afirmar que esse algoritmo conseguiu encontrar o maior número de soluções pertencentes ao conjunto de referência. O algoritmo NSGA-II teve um desempenho muito inferior, encontrando apenas 189 soluções do conjunto de referência. O algoritmo NSGA-II não conseguiu encontrar nenhuma solução pertencente ao conjunto de referência nas instâncias com menos de 100 tarefas.

A Tabela 2 apresenta a média e o melhor resultado obtido em 30 execuções de cada algoritmo considerando a métrica de cardinalidade.

Tabela 2. Resultados da Métrica Cardinalidade

T	R	H	Algoritmo			
			GRASP-MOVNS		NSGAII	
			Avg.	**Best**	**Avg.**	**Best**
27	18	12	**29.10**	**71**	0	0
72	21	7	**11.10**	**57**	0	0
100	16	4	**21.40**	**69**	3.98	38
Average			**20.5**	**65.7**	1.33	12.7

Como pode ser visto na Tabela 2, o algoritmo GRAP-MOVNS consegue gerar uma maior quantidade de soluções não-dominadas quando comparado com algoritmo NSGA-II. O número de soluções do conjunto referência geradas pelo algoritmo GRASP-MOVNS é, em média, 15 vezes maior que o gerado pelo algoritmo NSGA-II.

Tabela 3. Resultados de Hipervolume

T	R	H	Algoritmo			
			GRASP−MOVNS		NSGAII	
			Avg.	**Best**	**Avg.**	**Best**
27	18	12	**0.64**	**0.83**	0.12	0.32
72	21	7	**0.65**	**0.92**	0.26	0.73
100	16	4	**0.80**	**0.85**	0.54	0.79
Average			**0.70**	**0.87**	0.31	0.61

Na Tabela 3 pode ser observado que os valores de hipervolume do algoritmo GRASP-MOVNS são maiores que os valores do algoritmo NSGA-II. Isso significa que o GRASP-MOVNS produz melhores soluções, pois um alto valor de hipervolume indica que houve um elevado espalhamento entre as soluções e indica também que houve uma melhor convergência.

6. Conclusões

Esse trabalho tratou o Problema de Desenvolvimento de Cronograma com dois critérios de otimização para serem satisfeitos: minimizar o makespan e o custo do projeto.

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

Os dois algoritmos foram comparados em relação às métricas de cardinalidade e hipervolume. Os resultados computacionais obtidos em instâncias de projetos de desenvolvimento de software reais mostram que o Algoritmo GRASP-MOVNS é superior ao Algoritmo NSGA-II em relação às métricas avaliadas. O Algoritmo NSGA-II apresentou desempenho muito ruim, principalmente para instâncias com menos de 100 tarefas, nas quais obteve conjuntos de soluções totalmente dominadas pelas soluções do Algoritmo GRASP-MOVNS. Desta forma, fica evidenciada a contribuição da primeira proposta de uso da metaheurística GRASP-MOVNS com os 9 movimentos de vizinhança propostos para resolver o PDC.

7. Agradecimentos

Os autores agradecem à FAPEMIG, CNPq e CEFET-MG por apoiar o desenvolvimento dessa pesquisa.

8. Referências

Referências

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

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

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

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

Colares, F. (2010). Alocação de equipes e desenvolvimento de cronogramas em projetos de software utilizando otimização. Master thesis, UFMG.

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

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

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

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

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

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

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

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

Souza, M. J. F., Coelho, I. M., Ribas, S., Santos, H. G., and Merschmann, L. H. C. (2010). A hibrid heuristic algorithm for the open-pit-mining operacional planning problem. *European Journal of Operational Research*, 207(2):1041–1051.

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