Incorporation of Restriction Treatment Techniques Based on the particle swarm optimization Metaheuristic in the Software Project Scheduling Problem in Software Projects

Incorporação de Técnicas de Tratamento de Restrição Baseadas em Penalidade na Meta-heurística de Nuvem de Partículas no contexto do Problema do Escalonamento e Atribuição de Tarefas em Projetos de Software

Werthen Santos[1], **Leila Silva**[1], **Andre Britto**[1]

[1]Departamento de Computação
Universidade Federal de Sergipe (UFS) – São Cristóvão, Sergipe – Brasil

{wethen.santos,leila,andre}@dcomp.ufs.br

Abstract. *The Software Project Scheduling Problem in Software Projects consists of allocating employees to tasks in a way that minimizes the duration and cost of the project. To solve the problem metaheuristic algorithms have been applied, among them the multi-objective version of the Particle Swarm Optimization algorithm, SMPSO. However, the algorithm generates many solutions that violate some constraints, called invalid solutions. This work investigates the impact of the incorporation to the SMPSO of restriction treatment techniques based on penalty, in order to increase the number of valid solutions generated. The results suggest that the incorporation of the restriction treatment improves the quality of the generated solutions.*

Resumo. *O Problema do Escalonamento e Atribuição de Tarefas em Projetos de Software consiste na alocação de empregados a tarefas de forma a minimizar a duração e o custo do projeto. Para solucionar o problema algoritmos metaheurísticos têm sido aplicados, dentre eles a versão multiobjetivo do algoritmo Otimização por Nuvem de Partículas, SMPSO. No entanto, o algoritmo gera muitas soluções que violam alguma restrição, chamadas de soluções inválidas. Este trabalho investiga o impacto da incorporação ao SMPSO de técnicas de tratamento de restrição baseadas em penalidade, com objetivo de aumentar o número de soluções válidas geradas. Os resultados sugerem que a incorporação do tratamento de restrição melhora a qualidade das soluções geradas.*

1. Introdução

Os Problemas de Otimização Multiobjetivo, (em inglês, *Multi-Objective Optimization Problems* (MOPs)), são problemas que, para serem solucionados, necessitam otimizar mais de um objetivo simultaneamente, podendo ser problemas de minimização, maximização ou uma combinação destes. Em geral, estes objetivos são conflitantes entre si, de forma que a melhora de um objetivo pode resultar em piora de algum outro objetivo.

Assim não existe uma única solução para o problema, mas um conjunto de soluções formando a fronteira Pareto [Coello et al. 2007].

Os Algoritmos Evolucionários Multiobjetivo (em inglês, *Multi-Objective Evolutionary Algorithms* (MOEAs)) foram aplicados com sucesso para resolver MOPs em vários campos da ciência, como na economia [Tapia e Coello 2007]. Na área de Engenharia de Software, meta-heurísticas foram utilizadas em diversas subáreas, como por exemplo: requisitos [Zhang et al. 2008], projeto de software [Sievi-Korte 2010], teste de software [Afzal et al. 2009] e gerenciamento de projetos [Xiao et al. 2013]. O trabalho de [Harman et al. 2012] apresenta uma revisão da área de otimização em Engenharia de Software.

A área de gerenciamento de projetos de software visa desenvolver projetos de maneira eficiente e eficaz, ou seja, espera-se que o software projetado realize tudo o que foi originalmente solicitado dentro do prazo e do orçamento. No entanto, estudos relatam [Standish Group 2014] que 52,7% dos projetos de software custam mais do que o valor inicial projetado e 31,1% dos projetos de software são cancelados antes de serem concluídos. Assim, um bom planejamento de um projeto, alocando funcionários a tarefas de forma a minimizar custo e duração, é essencial para seu sucesso.

Neste contexto, uma das formas de realizar um bom escalonamento é mapeando-o como o Problema do Escalonamento e Atribuição de Tarefas em Projetos de Software (em inglês, *Software Project Scheduling Problem* (SPSP)), o qual consiste na alocação de empregados a tarefas de forma a minimizar a duração e o custo do projeto. Cada empregado possui um conjunto de habilidades e um tempo máximo de dedicação ao projeto e cada tarefa requer um conjunto de habilidades específicas para serem desenvolvidas, podendo ter dependências entre tarefas. O SPSP é um MOP e vários trabalhos aplicaram meta-heurísticas para encontrar soluções para o problema, [Alba e Chicano 2007], [Biju et al. 2015] e [Xiao et al. 2013]. Em [Rezende et al. 2019] é apresentada uma revisão sistemática sobre este problema.

Em particular, o MOEA SMPSO (do inglês, *Speed constrained Multi-objective* PSO) foi utilizado para solucionar o SPSP [Andrade et al. 2019]. Entretanto, um problema que surgiu ao utilizar SMPSO para solucionar o SPSP foi a convergência para soluções com valores de tempo e/ou custo zerados. Isso indica que essas soluções são impossíveis de serem utilizadas na prática, pois não existe projeto com o custo e/ou tempo necessário para sua conclusão igual a zero. Para lidar com soluções inválidas, técnicas de tratamento de restrições foram incorporadas aos MOEAs [Woldesenbet et al. 2009]. Em [Özgür 2005] é apresentada uma discussão das técnicas de restrição baseadas em função de penalidade.

Este trabalho visa investigar o impacto da incorporação ao SMPSO de técnicas de tratamento de restrição baseadas em penalidade, com objetivo de fazer com que o SMPSO consiga gerar um número maior de soluções válidas. As técnicas estudadas nesse trabalho foram a *Static Penalty* e *Death Penalty* e este trabalho tem como hipótese que as técnicas de tratamento de restrições irão melhorar o resultado do SMPSO no contexto do SPSP. Para isto foram realizados experimentos utilizando vinte e quatro das instâncias propostas por [Alba e Chicano 2007] e os resultados obtidos foram comparados com o próprio SMPSO.

Este artigo está organizado como descrito a seguir: a seção 2 apresenta a fundamentação teórica; a seção 3 apresenta o SMPSO e as modificações propostas neste trabalho; a seção 4 descreve os experimentos e resultados e a seção 5 apresenta as considerações finais.

2. Fundamentação Teórica

2.1. O Problema do Escalonamento e Atribuição de Tarefas em Projetos de Software

Este trabalho baseia-se na formulação do SPSP proposto em [Alba e Chicano 2007], o qual considera dois conjuntos, um de tarefas e outro de funcionários, associando-os de maneira a permitir que todas as tarefas do projeto sejam realizadas pelos funcionários.

O conjunto de funcionários é representado por $E, |E| = n$. Cada funcionário e_i possui um salário, um conjunto de habilidades S_i e um tempo máximo de dedicação a uma tarefa, podendo executar várias tarefas. A figura 1 mostra um conjunto de habilidades necessárias para realização de um projeto e sua distribuição entre os funcionários, além do salário de cada funcionário e o percentual de sua dedicação ao projeto.

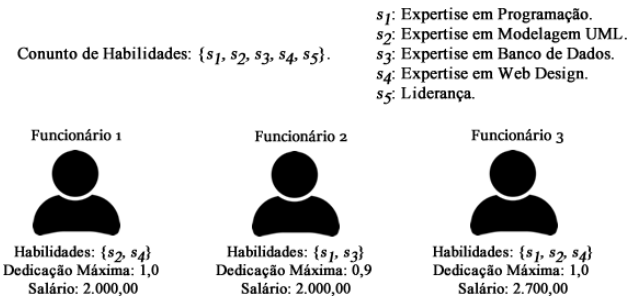


Figura 1. Representação dos funcionários e suas habilidades.

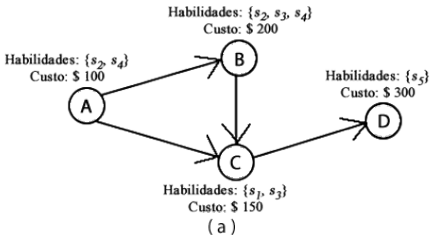
Cada tarefa do conjunto de tarefas *T*, de tamanho *m*, possui um custo associado que pode variar de acordo com as características da tarefa, e um conjunto de habilidades necessárias para concluí-la. A dependência entre tarefas é permitida e, portanto, algumas tarefas são executadas em uma ordem específica e isto é expressa por um Grafo Direcionado Acíclico (GDA).

O GDA é formado por um conjunto de vértices, *T*, representandos pelas tarefas, e um conjunto de arestas, *A*, que indicam a relação de precedência entre as tarefas. Seja uma aresta $(t_i, t_j) \in A$, então a tarefa t_j só pode ser iniciada após t_i ser finalizada.

Uma solução para o SPSP é representada por uma de Matriz de Dedicação (*MD*). Na matriz, cada célula md_{ij} indica o quanto um funcionário irá se dedicar para concluir uma tarefa.

A figura 2 (a) representa um GDA de um projeto. Neste projeto a tarefa C, por exemplo, só pode ser iniciada após a conclusão das tarefas *A* e *B*. A figura 2 (b) apresenta a matriz de dedicação, expressando uma solução do SPSP. Por exemplo, o funcionário 3 está alocado nas tarefas *C* e *D*, dedicando 100% do seu tempo ao projeto.

O modelo do SPSP proposto por Alba e Chicano (2007) prevê a satisfação de três restrições, definidas a seguir:



Tarefas	Funcionário 1	Funcionário 2	Funcionário 3
A	1,0	0,2	0,0
B	0,0	0,6	0,0
C	0,0	0,1	0,3
D	0,0	0,1	0,7

(b)

Figura 2. Exemplos de: (a) Grafo de Precedência de Tarefas e (b) Matriz de Dedicação.

Restrição 1 - Mínimo de funcionário em um tarefa: Cada tarefa deve ser realizada por pelo menos um funcionário. Assim, o somatório de uma linha da matriz de dedicação tem que ser maior que zero.

$$\sum_{i=1}^{n} md_{ij} > 0 \, \forall_j \in \{1, 2, ..., m\} \tag{1}$$

Restrição 2 - Habilidades do funcionário: O conjunto de habilidades exigido por uma tarefa $t_j^{habilidade}$ deve ser subconjunto da união dos conjuntos de habilidades dos funcionários que realizam a tarefa.

$$t_j^{habilidade} \subseteq \bigcup_{\{i \mid md_{ij} > 0\}} e_i^{habilidade} \, \forall_j \in \{1, 2, ..., m\} \tag{2}$$

Restrição 3 - Carga de trabalho: Nenhum funcionário deve exceder a sua máxima dedicação e_i^{max} ao projeto, ou seja, todo planejamento deve ser feito sem considerar horas extras. Para verificar essa restrição, calcula-se a função de trabalho de cada funcionário, considerando o início t_j^{inicio}, e o fim de cada tarefa, t_j^{fim}. Se $e_i^{trabalho} > e_i^{max}$, então a carga de trabalho do funcionário é excedida.

$$e_i^{trabalho}(t) = \sum_{\{j \mid t_j^{inicio} < t < t_j^{fim}\}} m_{ij} \tag{3}$$

2.2. Otimização Restrita Multiobjetivo

Um problema de otimização restrita é definido como um problema de otimização não linear:

$$Minf(\vec{x}) \qquad \vec{x} = (x_1, ..., x_s)^{\mathbf{T}} \in F \subseteq S \subseteq R^s \tag{4}$$

sujeito a

$$g_i(\vec{x}) \leq 0 \qquad i, ..., q \tag{5}$$

$$h_j(\vec{x}) = 0 \qquad j = q + 1, ..., b \tag{6}$$

onde $f(\vec{x})$ é a função objetivo e $\vec{x} = (x_1, ..., x_s)^{\mathbf{T}}$ é o vetor de soluções, S é todo o espaço de busca e F é a região viável, local onde as soluções possuem valores dentro do intervalo definido, existem q restrições de igualdade e $b - q$ de desigualdade. As funções $g_i(\vec{x})$ e $h_j(\vec{x})$ representam as restrições de desigualdade e igualdade impostas ao problema. O vetor $\vec{x}$ que satisfizer todas as restrições é uma solução viável para o problema.

Existem diversos métodos de tratamento de restrição, como métodos baseados em técnicas de penalidade, baseados na preservação da viabilidade das soluções, baseados na busca de soluções viáveis e métodos híbridos [Michalewicz e Schoenauer 1996]. Neste artigo investigamos métodos baseados em técnicas de penalidade.

3. Algoritmo Desenvolvido

O algoritmo SMPSO [Nebro et al. 2009] é a versão multiobjetivo do algoritmo Otimização por Nuvem de Partículas (PSO).O algoritmo funciona de forma a simular o comportamento social de um bando de pássaros em busca de alimento, com o objetivo de determinar qual é a melhor direção a ser tomada no espaço de busca de um problema.

No SMPSO todos os indivíduos da população são soluções potenciais para um problema específico, sendo uma solução chamada de partícula e o conjunto de partículas chamado de enxame. As partículas se movimentam pelo espaço de busca tentando encontrar uma posição que representa uma solução ótima.

O SMPSO utiliza uma estratégia para limitar a velocidade máxima das partículas, com o objetivo de impedir que as partículas percorram regiões distantes do espaço de busca desejado, assim convergindo mais rápido. Cada partícula p_i possui um limite inferior $limite_{inf}$ e superior $limite_{sup}$ de velocidade, a velocidade de p_i para cada valor da velocidade j é definida pelas equações 7 e 8.

$$v_{ij} = \begin{cases} delta_j & \text{se } v_{ij}(t) > delta_j \\ -delta_j & \text{se } v_{ij}(t) \leq -delta_j \\ v_{ij} & \text{Caso contrário} \end{cases} \tag{7}$$

$$delta_j = \frac{limite_{sup} - limite_{inf}}{2} \tag{8}$$

Para evitar variações bruscas de velocidade o SMPSO utiliza um fator K de restrição descrito pela equação 8. Na equação 9 o ϕ é calculado com base em C_1 e C_2 que são constantes que controlam, respectivamente, o efeito do valor da melhor posição encontrada por uma partícula (componente local) e a influência da melhor partícula próxima a ela (componente social).

$$K = \frac{2}{2 - \phi - \sqrt{\phi^2 - 4.\phi}} \tag{9}$$

$$\phi = \begin{cases} C_1 + C_2 & \text{se } C_1 + C_2 > 4 \\ 1 & \text{se } C_1 + C_2 \leq 4 \end{cases} \tag{10}$$

A representação do movimento de uma partícula x_i em um tempo/iteração $\mathcal{T}$ e utilizando uma velocidade $v(\mathcal{T})$ e dada pela equação 11:

$$x_{ij}(\mathcal{T}+1) = x_{ij}(\mathcal{T}) + [K.v_{ij}(\mathcal{T})] \tag{11}$$

Conforme passam as gerações, utilizando a *crowding distance* as soluções são selecionadas para um arquivador A, com objetivo de utilizá-las para influenciar as outras soluções. O SMPSO utiliza também um operador de turbulência (mutação polinomial) para gerar uma alteração aleatória aumentado a variabilidade das soluções.

Este trabalho acrescenta ao SMPSO técnicas de penalidade *Static Penalty* e *Death Penalty* descritas nas seções 3.1.1 e 3.1.2.

O algoritmo começa criando e inicializando uma população de partículas, em seguida as soluções são avaliadas calculando seu *fitness*, é verificado se as soluções desobedecem alguma restrição e são escolhidas as partículas com o papel de líderes. A partir daí inicia-se um processo repetitivo, onde cada uma das iterações funciona como uma nova geração de partículas. Cada partícula tem sua velocidade atualizada, e se necessário é aplicada a restrição de velocidade, é atualizada a posição de cada partícula, é verificado seu *fitness* e se desobedece alguma restrição. Depois disto, ocorre a definição dos novos líderes para recomeçar o processo.

3.1. Funções de Penalidade

Os algoritmos bio-inspirados originalmente são tratados sem restrição, mas alguns problemas necessitam de restrições para que suas soluções sejam viáveis de serem aplicadas. Para contornar esse problema foi acrescentada uma função penalidade à função que determina a evolução do algoritmo, de maneira a fazer com que soluções que desobedeçam alguma restrição tenham o valor de sua função objetivo penalizado.

$$Evol(\vec{x}) = \begin{cases} f(\vec{x}) & \text{se } f(\vec{x}) \in F \\ f(\vec{x}) + penalidade(\vec{x}) & \text{caso contrário} \end{cases} \tag{12}$$

em que, penalidade ($\vec{x}$) é zero se não ocorrer violação das restrições, caso contrário é um valor positivo e proporcional à quantidade de violações ocorridas.

3.1.1. *Static Penalty*

Esse método consiste em aplicar uma punição fixa na função objetivo, que seu valor não depende de qual iteração o algoritmo está, toda vez que uma restrição for desobedecida [Homaifar et al. 1994]. Esse método possui quatro etapas, descritas a seguir:

1. Para cada restrição, crie *l* níveis de violação;

2. Para cada nível i de violação e para cada restrição j, crie um coeficiente de penalidade $R_{ij}(i = 1, 2, ..., l, j = 1, 2, ..., m)$; níveis mais altos de violação possuem valores maiores deste coeficiente;

3. Comece com uma população aleatória de indivíduos (viável ou inviável);

4. Evolua a população; avalie indivíduos usando a seguinte fórmula:

$$Evol(\vec{x}) = f(\vec{x}) + \sum_{j=1}^{m} R_{ij} f^2(\vec{x}) \tag{13}$$

Neste trabalho devido às características do SPSP existe apenas um nível de penalidade para cada algoritmo, que ocorre quando o valor de tempo e/ou custo for zero e o coeficiente de penalidade R é um fator que multiplica o valor da função objetivo. As restrições citadas na 2.1 são tratadas pela própria implementação do SPSP.

3.1.2. *Death Penalty*

Este é o método de penalidade mais simples, que não aplica uma penalização na função objetivo como o método anterior, mas simplesmente rejeita a solução inviável, removendo-a do conjunto de soluções e gerando uma nova solução para manter o conjunto de soluções do mesmo tamanho. Isso faz com que todo o conjunto de soluções seja viável, já que quando uma nova solução é criada, é verificado se ela desobedece a alguma restrição e caso ocorra uma nova violação, a nova solução não é adicionada ao conjunto de soluções. O ponto negativo desta técnica é que a remoção de soluções inviáveis proporciona uma menor variabilidade do conjunto, implicando numa menor cobertura do espaço de busca, o que pode levar a um processo de otimização inviável caso as restrições sejam difíceis de sejam cumpridas aleatoriamente.

4. Experimentos e Resultados

O objetivo dos experimentos foi aferir o impacto da adoção de tratamento de restrições na aplicação do SMPSO ao SPSP. Para os experimentos foram criadas seis variações do SMPSO, uma com o método de penalidade *Death Penalty* e outras cinco com o *Static Penalty*. As cinco variações com o *Static Penalty* diferem apenas no valor da penalidade que é aplicado; as punições variaram de 100% a 500% do valor da função objetivo. A tabela 1 sumariza as variantes utilizadas e a técnica de penalidade adotada.

Tabela 1. Técnicas de penalidade para o SMPSO.

Técnica	Penalidade
Static Penalty - (SP1)	Penalidade de 100% na função objetivo.
Static Penalty - (SP2)	Penalidade de 200% na função objetivo.
Static Penalty - (SP3)	Penalidade de 300% na função objetivo.
Static Penalty - (SP4)	Penalidade de 400% na função objetivo.
Static Penalty - (SP5)	Penalidade de 500% na função objetivo.
Death Penalty - (P)	Exclusão da partícula

As vinte quatro instâncias utilizadas nos experimentos são divididas em grupos, pelo número de tarefas que possuem, num total de quatro grupos de seis instâncias cada um deles. Os números de tarefas em cada grupo são 16, 32, 64 e 128. Cada uma das seis instâncias de um grupo varia em número de funcionários utilizados. Estas instâncias foram propostas por [Alba e Chicano, 2007].

Para a realização dos experimentos foi utilizado o *framework* de otimização multi-objetivo jMetal[1]. O modelo do SPSP foi implementado no JMetal, bem como as variantes do SMPSO. O SMPSO foi executado por 20 vezes com sua configuração padrão, com 100 soluções, o arquivo de tamanho 1000, com 250 iterações e usando a *Polynomial Mutation*.

[1]Disponível em: http://jmetal.sourceforge.net/.

A análise dos resultado foi feita em duas etapas, na primeira foi investigado se os métodos de tratamento de restrição evitavam ou reduziam a geração de soluções inválidas nas instâncias testadas. Para fazer essa verificação, foi realizada a contagem de execuções que produziram um conjunto de soluções inválidas em cada um dos algoritmos.

A figura 3 mostra o gráfico com a quantidade e a distribuição dos conjuntos de soluções inválidas geradas pelas 20 execuções em cada uma das 24 instâncias testadas pelos algoritmos. Os algoritmos SMPSO, SP1, SP2, SP3, SP4, SP5 e P, foram testados para cada uma das 24 instâncias já mencionadas.

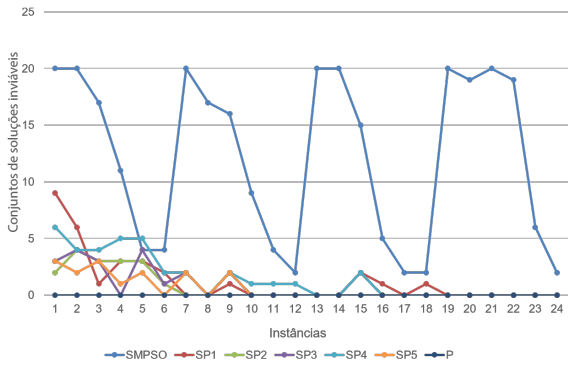


Figura 3. Distribuição de soluções inválidas.

Na figura 3 é evidenciado que o SMPSO "puro", sem utilizar nenhuma técnica de tratamento de restrição, obteve o pior resultado gerando um conjunto de soluções inválidas em sete instâncias distintas (1, 3, 7, 13, 14, 19) em todas as 20 execuções. O número de soluções inválidas do SMPSO é muito maior do que todos os outros algoritmos testados e ocorrem com maior intensidade em instâncias com menos funcionários. Como esperado, a variante P não gerou nenhuma solução inválida.

As variantes do SMPSO que utilizaram o *Static Penalty* geraram poucas soluções inválidas, possuindo melhores resultados com instâncias com maior número de funcionários e tarefas. Por exemplo, o SP1 gera na primeira instância 9 conjuntos de soluções inválidas, na instância 2 o número de cai para 6, na instância 3 cai para 1, aumenta para 3 na instância 4, mantém o mesmo desempenho na instância 5, passando para 2 na instância 6, nas instâncias 7 e 8 os conjuntos de resultados são formados por soluções válidas, o que se repete nas instâncias 10, 11, 12, 13, 14, 17, 19, 20, 21, 22, 23 e 24.

A segunda parte dos experimento visou analisar a qualidade das soluções utilizando o indicador de qualidade hipervolume [Fonseca et al. 2006]. O hipervolume é o calculo da área entre a fronteira de Pareto usando um ponto de referência. Como os dados deste trabalho foram normalizados, de maneira a fazer todos os valores ficarem em um intervalo de 0 a 1, os pontos utilizados como referência neste trabalho foram 1.1 e 1.1.

Para verificar quais algoritmos geravam as melhores soluções, foi realizada a comparação das médias e desvio padrão dos hipervolumes obtidos das 20 execuções, para cada algoritmo. Nesse experimento, o teste de Kruskall-Walis foi adotado verificar se há diferença estatística entre os algoritmos, utilizando um nível. Além disso, o pós-teste de Tukey para fazer uma análise de comparação múltipla entre os conjuntos de dados dos algoritmos. A tabela 2 apresenta os resultados obtidos.

Tabela 2. Resultados dos hipervolumes por algoritmo e por instância.

		SMPSO	SP1	SP2	SP3	SP4	SP5	P
Instância 1	Média	0,0000	0,4917	1,0928	1,0913	1,1084	1,1218	0,8543
	Desvio	0,0000	0,2250	0,0599	0,0599	0,0302	0,0502	0,0844
	p-valor	6,4e-08	0,0002	0,9999	0,9956	0,8705	-	0,1609
Instância 2	Média	0,0000	0,5227	1,0720	1,1052	1,0859	1,0736	1,1188
	Desvio	0,0000	0,1404	0,0776	0,0535	0,0690	0,0736	0,0587
	p-valor	3.4e-10	2.1e-06	0.3600	0.5526	0.3357	0.5526	-
Instância 3	Média	0,8400	1,0318	0,9713	1,0529	1,0489	1,0422	0,9955
	Desvio	0,5036	0,0843	0,0783	0,0765	0,0722	0,0763	0,0975
	p-valor	0.0003	1.0000	0.41462	-	0.9993	1.0000	0.9999
Instância 4	Média	0,9688	1,0548	1,0438	1,0394	0,9893	1,1091	1,0807
	Desvio	0,1307	0,0686	0,0772	0,0801	0,0773	0,0578	0,0693
	p-valor	3.3e-06	0.2732	0.1711	0.4399	0.0003	-	0.9986
Instância 5	Média	0,9906	0,5324	0,9984	1,0285	1,0276	1,0217	1,0373
	Desvio	0,1802	0,1673	0,1046	0,0944	0,1060	0,0982	0,0855
	p-valor	1.0000	0.0386	1.0000	1.0000	0.9876	0.9720	-
Instância 6	Média	0,9250	1,0137	0,9580	0,9405	0,9800	0,9940	0,9833
	Desvio	0,1926	0,0889	0,1061	0,1143	0,1083	0,1221	0,1220
	p-valor	0.5400	-	0.9900	0.9900	1.0000	0.9700	1.0000
Instância 7	Média	0,0000	1,0561	0,2926	1,0370	1,0662	0,9696	1,0629
	Desvio	0,0000	0,0729	0,1713	0,0611	0,0702	0,0641	0,0710
	p-valor	7.8e-09	0.9998	0.0003	0.9990	-	0.1711	0.0253
Instância 8	Média	0,8561	1,0126	1,0456	0,9599	0,9127	1,0073	0,9841
	Desvio	0,4997	0,0897	0,0909	0,0971	0,0765	0,0896	0,0799
	p-valor	2.3e-08	0.9484	-	0.1667	0.7939	0.0875	0.9945
Instância 9	Média	0,6913	0,4441	0,9960	1,0348	0,9847	1,0485	0,7507
	Desvio	0,2344	0,0295	0,0812	0,0793	0,0899	0,0638	0,0926
	p-valor	1.1e-08	7.1e-07	0.9996	0.9998	0.9979	-	0.0066
Instância 10	Média	0,9408	1,0105	1,0510	1,0295	1,0753	1,0124	0,9964
	Desvio	0,1884	0,0616	0,0651	0,0898	0,0711	0,0848	0,0924
	p-valor	0.0003	0.3368	0.9999	0.9325	-	0.5353	0.9999
Instância 11	Média	0,9515	0,9959	1,0055	1,0559	0,9764	1,0091	1,0108
	Desvio	0,1633	0,0696	0,0818	0,0846	0,0983	0,08054	0,0885
	p-valor	0.0130	0.5020	0.7540	-	0.0960	0.8550	0.8550
Instância 12	Média	0,8698	0,9959	0,9840	0,9966	1,0097	1,0201	0,8902
	Desvio	0,2347	0,0889	0,1053	0,0799	0,0946	0,0754	0,1025
	p-valor	0.0474	0.9969	0.8850	0.9886	0.9992	-	0.0046
Instância 13	Média	0,0000	1,0391	1,0334	0,9946	1,0523	1,0090	0,8272
	Desvio	0,0000	0,0588	0,0426	0,0616	0,0492	0,0862	0,0811
	p-valor	3.0e-11	0.9990	0.9781	0.3778	-	0.8671	0.0029
Instância 14	Média	0,0000	1,0213	0,9590	1,0196	1,0024	0,9907	0,6966
	Desvio	0,0000	0,0588	0,0426	0,0616	0,0492	0,0862	0,0811
	p-valor	1.8e-10	-	0.4272	0.3448	0.9974	0.9982	0.0014
Instância 15	Média	0,7785	1,0103	1,0263	1,0287	0,9443	0,9906	0,9229
	Desvio	0,2321	0,0616	0,0788	0,0734	0,0763	0,0883	0,0734
	p-valor	5.5e-07	1.0000	0.9990	-	0.1900	0.5900	0.4200
Instância 16	Média	0,9900	1,0709	1,0225	1,0290	1,0438	1,0204	0,9483
	Desvio	0,1190	0,0719	0,0659	0,0698	0,0688	0,0696	0,0810
	p-valor	0.0482	0.9977	0.9851	0.9993	-	0.9791	0.1223
Instância 17	Média	0,9432	1,0487	1,0384	1,0333	1,0401	1,0104	0,9225
	Desvio	0,1426	0,0579	0,0714	0,0633	0,0832	0,0758	0,0669
	p-valor	0.0350	-	1.0000	1.0000	1.0000	0.9780	0.8290
Instância 18	Média	0,8536	0,9796	0,9614	1,0685	1,0239	0,9676	1,0123
	Desvio	0,1920	0,0974	0,0787	0,0721	0,0992	0,0944	0,0776
	p-valor	1.6e-05	0.0392	0.0084	-	0.8045	0.6885	0.8637
Instância 19	Média	0,0000	0,9888	0,9814	0,9356	0,9768	1,0116	0,6684
	Desvio	0,0000	0,0554	0,0642	0,0714	0,0666	0,0635	0,0979
	p-valor	6.4e-12	0.9928	0.9493	0.1994	0.8705	-	3.5e-07
Instância 20	Média	0,58337	1,0092	0,9769	1,0505	1,0404	1,0245	0,8592
	Desvio	0,0000	0,0508	0,0582	0,0582	0,0490	0,0536	0,0562
	p-valor	3.5e-12	0.6809	0.0592	-	0.9998	0.9898	0.0002
Instância 21	Média	0,0000	1,0844	1,0321	1,0824	1,0615	1,0559	0,8556
	Desvio	0,0000	0,0511	0,0474	0,0583	0,0453	0,0514	0,0521
	p-valor	1.9e-11	-	0.2960	0.4400	0.9940	0.9999	0.0003
Instância 22	Média	0,7492	1,0028	0,9729	1,0715	1,0189	1,0771	0,9062
	Desvio	0,0000	0,0561	0,0834	0,0580	0,0534	0,51890	0,0663
	p-valor	0.2582	0.0010	0.0300	3.4e-08	0.0001	-	0.9256
Instância 23	Média	0,8469	0,9372	0,9967	1,0635	1,0420	1,0341	0,8593
	Desvio	0,1766	0,0753	0,0729	0,0529	0,0590	0,0667	0,0736
	p-valor	1.1e-06	0.0013	0.2612	-	0.9844	0.9986	0.0001
Instância 24	Média	0,8326	0,9327	1,0094	0,9338	1,0597	1,0341	0,8593
	Desvio	0,228	0,0611	0,0889	0,0874	0,0596	0,0962	0,0689
	p-valor	2.7e-05	0.0015	0.6200	0.0014	-	0.0010	0.0118

Para apresentar os resultados de forma mais concisa, a tabela 2 apresenta somente o p-valor entre o algoritmo que obteve a melhor média em relação aos demais algoritmos. Assim, por exemplo, o algoritmo SP5 obteve a maior média de hipervolume 1,1218 e avaliando o seu p-valor contra os outro algoritmos, percebemos que ele é estatisticamente igual aos algoritmos SP2, SP3, SP4 e P, já que o p-valor deles são maiores que 0,05. Já ao comparar o SP5 com SMPSO e SP1 verificamos que ele são estaticamente diferente, como SP5 possui uma média de hipervolume maior, logo ele obteve um desempenho melhor do que o SMPSO e SP1.

Na tabela 3 é mostrado quais algoritmos obtiveram o melhor resultado médio de seus hipervolumes, nas 24 instâncias analisadas.

Tabela 3. Melhor algoritmo por instância

Algoritmo	Instâncias
SMPSO	
SP1	6, 14, 17 e 21.
SP2	8.
SP3	3, 11, 15, 18, 20 e 23.
SP4	7, 10, 13, 16 e 24.
SP5	1, 4, 9, 12, 19 e 22.
P	2 e 5.

A tabela 3 mostra que mesmo quando o SMPSO gerou soluções viáveis, estas não obtiveram um resultado melhor que suas variantes em nenhuma das instâncias testadas. As variantes SP3 e SP5 foram os algoritmos que obtiveram o melhor desempenho nos testes, alcançando melhores resultados em 6 das 24 instâncias analisadas. Já os algoritmos SP1, SP2, SP4 e P obtiveram o melhor resultados respectivamente em 4, 1, 5 e 2 instâncias.

5. Considerações Finais

O objetivo desse trabalho foi avaliar o impacto na qualidade das soluções geradas para o SPSP, obtidas pela adição de métodos de tratamento de restrições, no algoritmo SMPSO. Foram analisados dois métodos de tratamento de restrição, o *Static Penalty* e o *Death Penalty*.

Foram realizadas baterias de experimentos, considerando-se 24 instâncias sintéticas extraídas da literatura [Alba e Chicano 2007]. Os resultados sugerem que o tratamento de restrições melhoram a qualidade das soluções obtidas em comparação a aplicação do SMPSO sem este tratamento.

Dos métodos investigados o *Static Penalty* apresentou o melhor resultado médio de hipervolume, superando o *Death Penalty* em quatro das cinco variações de penalidade testadas. Como trabalhos futuros pretende-se investigar outros métodos de tratamento de restrições, assim como realizar experimentos com outras meta-heurísticas para consolidar os resultados.

Referências

Afzal, W., Torkar, R., e Feldt, R. (2009). A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957 – 976.

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

Andrade, J., Silva, L., Britto, A., e Amaral, R. (2019). *Solving the Software Project Scheduling Problem with Hyper-heuristics*, páginas 399–411.

Biju, A., Victoire, T., e Mohanasundaram, K. (2015). An improved differential evolution solution for software project scheduling problem. *The Scientific World Journal*, 2015.

Coello, C. A. C., Lamont, G. B., e Van Veldhuizen, D. A. (2007). *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer.

Fonseca, C. M., Paquete, L., e López-Ibáñez, M. (2006). An improved dimension-sweep algorithm for the hypervolume indicator. In *Proceedings of the Evolutionary Computation on 2006. CEC '06. Proceedings of the 2006 Congress*, CEC '06, páginas 1157–1163.

Harman, M., McMinn, P., de Souza, J. T., e Yoo, S. (2012). *Search Based Software Engineering: Techniques, Taxonomy, Tutorial*, páginas 1–59. Springer Berlin Heidelberg, Berlin, Heidelberg.

Homaifar, A., X. Qi, C., e Lai, H.-Y. (1994). Constrained optimization via genetic algorithms. *Transactions of The Society for Modeling and Simulation International - SIMULATION*, 62:242–253.

Michalewicz, Z. e Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32.

Nebro, A., Durillo, J., García-Nieto, J., Coello Coello, C., Luna, F., e Alba, E. (2009). Smpso: A new pso-based metaheuristic for multi-objective optimization. páginas 66 – 73.

Rezende, A. V., Silva, L., Britto, A., e Amaral, R. (2019). Software project scheduling problem in the context of search-based software engineering: A systematic review. *Journal of Systems and Software*, 155:43 – 56.

Sievi-Korte, O. (2010). A survey on search-based software design. *Computer Science Review*, 4:203–249.

Standish Group (2014). Chaos report.

Tapia, M. G. C. e Coello, C. A. C. (2007). Applications of multi-objective evolutionary algorithms in economics and finance: A survey. In *2007 IEEE Congress on Evolutionary Computation*, páginas 532–539.

Woldesenbet, Y. G., Yen, G. G., e Tessema, B. G. (2009). Constraint handling in multiobjective evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 13(3):514–525.

Xiao, J., Ao, X.-T., e Tang, Y. (2013). Solving software project scheduling problems with ant colony optimization. *Computers Operations Research*, 40(1):33 – 46.

Zhang, Y., Finkelstein, A., e Harman, M. (2008). Search based requirements optimisation: Existing work and challenges. In Paech, B. e Rolland, C., editors, *Requirements Engineering: Foundation for Software Quality*, páginas 88–94, Berlin, Heidelberg. Springer Berlin Heidelberg.

Özgür, Y. (2005). Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and Computational Applications*, 10:45–56.