Comparação de Algoritmos Baseados nas Meta-heurísticas Firefly Algorithm e Tabu Search na Resolução do Problema da Mochila 0-1

Daniel Freitas Martins
Universidade Federal de Viçosa – *Campus* Florestal
Rodovia LMG 818, km 06, s/n, Florestal - MG, 35690-000
daniel.f.martins@ufv.br

Marcus Henrique Soares Mendes
Universidade Federal de Viçosa – *Campus* Florestal
Rodovia LMG 818, km 06, s/n, Florestal - MG, 35690-000
marcus.mendes@ufv.br

RESUMO

Este trabalho apresenta a comparação de quatro algoritmos na resolução do Problema da Mochila 0-1 e apresenta também duas variações inspiradas na meta-heurística Firefly Algorithm, que são intituladas Simple Binary Firefly Algorithm (SBFA) e Simple Binary Firefly Algorithm With Influencers (SBFAWI). Os algoritmos, denominados de Tabu Search (TS) e Tabu Search With Path Relinking (TSWPR), são implementações da meta-heurística Tabu Search. O desempenho dos algoritmos foi analisado considerando-se um conjunto de treze instâncias do Problema da Mochila 0-1. Os experimentos mostraram que: SBFA, SBFAWI e TSWPR encontraram o ótimo nas treze instâncias, mas somente o SBFAWI alcançou o ótimo em todas as execuções realizadas; o SBFA e o SBFAWI foram os mais adequados para uma instância com alta correlação entre os dados de entrada; e TS e TSWPR mostraram melhor desempenho nas métricas de tempo e número de avaliações da função objetivo para a maioria das instâncias.

PALAVRAS CHAVE. Problema da Mochila 0-1, Firefly Algorithm, Tabu Search.

Tópicos: Otimização Combinatória, Meta-heurísticas

ABSTRACT

This paper presents the comparison of four algorithms in solving the 0-1 Knapsack Problem and also presents two variations inspired by the Firefly Algorithm metaheuristic, which are entitled Simple Binary Firefly Algorithm (SBFA) and Simple Binary Firefly Algorithm With Influencers (SBFAWI). The algorithms, called Tabu Search (TS) and Tabu Search With Path Relinking (TSWPR), are implementations of the Tabu Search metaheuristic. The performance of the algorithms was analyzed considering a set of thirteen instances of the 0-1 Knapsack Problem. The experiments showed that: SBFA, SBFAWI and TSWPR found the optimum in the thirteen instances, but only SBFAWI reached the optimum in all the performed runs; SBFA and SBFAWI were the most suitable for an instance with a high correlation between the input data; and TS and TSWPR showed better performance in the metrics of time and number of evaluations of the objective function for most instances.

KEYWORDS. 0-1 Knapsack Problem, Firefly Algorithm, Tabu Search.

Paper topics: Combinatorial Optimization, Metaheuristics

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings

1. Introdução

O Problema da Mochila 0-1 é um problema NP-Difícil de otimização combinatória [Pisinger e Toth, 1998]. O problema consiste em, dado um conjunto de itens representados por pares <peso, valor>, determinar quais itens devem ser adquiridos de maneira que a soma dos valores individuais de cada item seja a maior possível, respeitando a restrição do peso total que a mochila pode suportar. É importante ressaltar que uma vez escolhido um item, ele não pode ser escolhido novamente. Existem variações do problema em que outras características são acrescentadas, como descritas em [He et al. 2016], [Fréville, 2004], [Pisinger, 2000] e [Pisinger e Toth, 1998]. Estudos acerca deste problema e de suas variações são importantes, uma vez que existem diversas aplicações tanto em termos de aplicações no mundo real, como em problemas industriais e de investimentos, por exemplo, quanto no campo teórico, em que suas propriedades são estudadas a fim de identificar técnicas eficientes para a resolução de outros problemas mais complexos [Pisinger e Toth, 1998].

Vários algoritmos foram propostos para resolver este problema. Em [Basheer e Algamal, 2019], quatro variações de meta-heurísticas inspiradas na natureza são apresentadas: Binary Particle Swarm Optimization (BPSO), Binary Firefly Algorithm (BFA), Binary Flower Polination Algorithm (BFPA) e Binary Monarch Butterfly Optimization (BMBO). Uma outra variação da meta-heurística Monarch Butterfly Optimization (MBO) é apresentada em [Feng et al., 2017]. Uma modificação na meta-heurística Whale Optimization Algorithm (WOA), denominada Improved Whale Optimization Algorithm (IWOA), capaz de resolver as versões unidimensional e multidimensional do Problema da Mochila 0-1, é apresentada por [Abdel-Basset et al., 2019]. Em [Chou et al., 2011] é apresentada uma versão de Tabu Search inspirada em conceitos quânticos.

A meta-heurística Firefly Algorithm (FA) foi proposta inicialmente para resolver problemas contínuos e baseia-se em um comportamento idealizado a respeito das características relacionadas à bioluminescência dos vagalumes [Yang, 2010]. A fim de adaptar FA para problemas binários [Basheer e Algamal, 2019] propõem uma transformação do valor real das variáveis de decisão utilizando-se uma função sigmoide: se o resultado desta função for maior do que um número aleatório gerado no intervalo $[0, 1]$, então a variável recebe o valor 1, e recebe 0, caso contrário. No presente trabalho é proposta uma variação do FA capaz de resolver problemas binários, usando uma estrutura condicional que considera os valores de atratividade e aleatoriedade envolvidos no algoritmo. Uma outra variação proposta refere-se à inclusão de um grupo especial na população de vagalumes. Os indivíduos desse grupo são chamados de *influencers* e possuem critérios de inicialização e de movimento diferentes dos demais. Ainda, é mostrada uma comparação do desempenho destas estratégias com duas implementações da meta-heurística Tabu Search, sendo uma com *path relinking* e a outra sem. Tabu Search possui excelentes resultados em problemas discretos e baseia-se na premissa de que a solução de problemas, para se qualificar como inteligente, deve incorporar memória adaptativa e exploração responsiva [Glover, 1989], [Glover, 1990] e [Glover e Laguna, 1998]. O *path relinking* consiste em explorar outras soluções do espaço de busca perturbando-se uma solução base até que se alcance uma solução guia a fim de encontrar melhores soluções [Glover e Laguna, 1998].

O restante deste documento está assim organizado: a seção 2 descreve matematicamente o problema analisado; a seção 3 apresenta a meta-heurística Tabu Search e os aspectos de implementação que foram considerados, incluindo *path relinking*; a seção 4 apresenta a meta-heurística Firefly Algorithm e as adaptações realizadas para a resolução de problemas com variáveis binárias; a seção 5 apresenta os resultados dos testes e as comparações realizadas; a seção 6 apresenta a conclusão e as considerações finais deste trabalho.

2. O Problema da Mochila 0-1

De acordo com [Feng, 2017], dados um conjunto U de n itens $u_1, u_2, \dots, u_n$, isto é, $U = \{u_1, u_2, \dots, u_n\}$, um peso w_i e um valor (ou lucro) v_i para cada item u_i, $1 \leq i \leq n$, e um inteiro w_{max} representando a capacidade total da mochila, o objetivo é encontrar um subconjunto A de

itens de modo a maximizar o lucro total $\sum_{u_i \in A} v_i$, sujeito à condição de que a soma dos pesos de cada item $\sum_{u_i \in A} w_i$ não exceda a capacidade da mochila.

De modo geral, o Problema da Mochila 0-1 pode ser definido como se segue:

$$\text{Maximizar} \quad f(\vec{x}) = \sum_{i=1}^{n} v_i x_i \tag{1}$$

$$\text{Sujeito a} \quad w_{total} \leq w_{max}$$
$$x_i \in \{0,1\}, \quad i = 1,2,\dots n, \quad w_{total} = \sum_{i=1}^{n} w_i x_i \tag{2}$$

onde $x_i = 1$ se $u_i \in A$ e $x_i = 0$, caso contrário.

3. Tabu Search

A meta-heurística Tabu Search (TS) possui um bom desempenho para problemas de otimização combinatória. Além disso, tem um amplo campo de aplicação, envolvendo áreas como análise financeira, distribuição de energia, conservação ambiental, dentre outras [Glover e Laguna, 1998].

A palavra *tabu* refere-se a algo proibido ou coisa sagrada que não pode ser tocada. TS usa esse conceito para marcar elementos em uma lista de movimentos proibidos (lista *tabu*) a partir de dados históricos da execução. Esses dados históricos guardados mostram uma propriedade importante da TS: o uso de memória. Com base na memória, a TS consegue sair de ótimos locais e explorar novas áreas, aceitando piora da função objetivo. Além da lista *tabu*, utiliza-se o conceito de vizinhança, que consiste de um conjunto de soluções próximas a uma solução considerada. Os vizinhos podem ser verificados se são válidos através da lista *tabu*. Porém, como o recurso de memória computacional é limitado, tanto o tamanho da lista *tabu* quanto o tamanho da vizinhança devem ser limitados. Dependendo dos movimentos configurados e dos tamanhos atribuídos para cada um desses elementos, o algoritmo pode ter uma convergência prematura e não alcançar o ótimo global. A descrição completa do método pode ser encontrada em [Glover, 1989] e [Glover, 1990].

Neste trabalho, considerou-se apenas o movimento de troca de bits, isto é, se $x_i = 0$, então $x_i \leftarrow 1$ ou $x_i \leftarrow 0$, caso contrário. Sendo assim, cada elemento da vizinhança difere de apenas uma variável de decisão com relação a última solução computada pela iteração anterior. Se um vizinho indicar um movimento como *tabu*, isto é, a troca do i-ésimo bit não pode ser feita, então ele só será considerado se satisfazer algum critério de aspiração.

Dois critérios de aspiração foram utilizados, são eles:

1) Aspiração por objetivo com condição aleatória: Se a troca do i-ésimo bit resultar em uma solução melhor do que a melhor solução atual e $r > 0.9$, onde $r \in [0,1]$ é um número aleatório, então a troca é considerada. Empiricamente, o uso dessa condição aleatória mostrou uma melhora na solução global.

2) Aspiração por *default*: Realiza o movimento *tabu* mais antigo se todos forem *tabu*. Em outras palavras, se nenhum vizinho está qualificado para ser a solução escolhida da iteração atual, escolhe-se aquele cujo movimento associado está por último na lista *tabu*.

A fim de melhorar as soluções encontradas, foi implementada uma estratégia de *path relinking* (veja Fig. 1) baseada nas ideias apresentadas em [Glover e Laguna, 1998]: Ao término de todas as iterações da TS, adote como solução base a solução inicial gerada aleatoriamente na primeira iteração. Partindo da solução base, caminhe em direção à solução final obtida pelo processo de otimização (solução guia), alterando-se apenas os bits que diferem de uma solução para a outra. A partir disso, busque soluções próximas e substitua a solução atual por essa nova

solução caso ela seja melhor que a solução guia atual (a melhor solução sempre deve ser a solução guia). Para isso, utilize uma nova instância de TS para aproveitar os métodos de busca a partir da solução atual. Este processo se repete para todos os bits que diferirem com relação à solução guia. Desta forma, é impossível estabelecer de antemão quantas iterações serão realizadas em todo esse processo. Foram utilizados os seguintes parâmetros fixos para as chamadas de TS pós-otimização: $q = 50; |V| = 50; |T| = 12$, que correspondem ao número de iterações para cada instância de TS, a cardinalidade da vizinhança e a cardinalidade da lista *tabu*, respectivamente. Essa implementação com *path relinking* foi intitulada Tabu Search With Path Relinking (TSWPR).

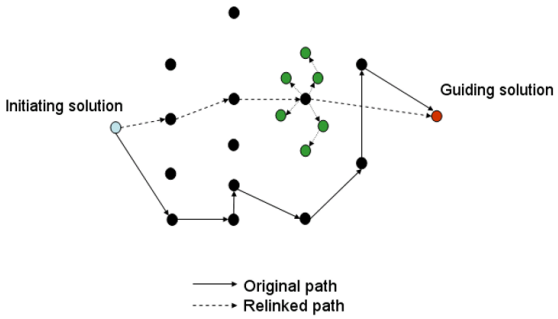


Figura 1: Ilustração de uma exploração no espaço de busca por meio de *path relinking*. Imagem retirada de [Qi, 2006].

4. Firefly Algorithm

A meta-heurística Firefly Algorithm (FA) foi construída tendo como base as características relacionadas à bioluminescência dos vagalumes [Yang, 2010]. A partir das luzes emitidas pelos vagalumes, eles podem atrair parceiros para acasalamento ou presas em potencial. Além disso, a emissão dessas luzes pode servir como um mecanismo de proteção contra possíveis predadores. De modo a abstrair estes detalhes, algumas considerações foram adotadas [Yang, 2010]:

I. Todos os vagalumes possuem o mesmo sexo. Desta forma, todos os vagalumes podem ser atraídos por qualquer outro independente do sexo;
II. Atratividade é proporcional ao seu brilho, isto é, o vagalume menos brilhante deve se mover em direção ao vagalume com brilho mais intenso. Caso o brilho entre dois vagalumes sejam indistinguíveis, o movimento é aleatório. Outro ponto em consequência dessa proporcionalidade é que a atratividade entre dois vagalumes deve se reduzir à medida que a distância entre eles aumenta devido à absorção da luz pelo ambiente; e
III. A intensidade do brilho de um vagalume é afetada ou determinada pelo panorama (*landscape*) da função objetivo considerada para a otimização.

Na formulação do FA, existem duas questões importantes: a variação da intensidade da luz e a formulação da atratividade para cada vagalume. Seja r a distância entre dois vagalumes quaisquer. Na forma mais simples, a intensidade da luz $I(r)$ de um vagalume com relação a outro varia a partir de uma função exponencial e monótona com relação à distância r entre eles [Yang, 2010]. Seja I_0 e γ a intensidade de luz original e o coeficiente de absorção de luz, respectivamente. Então a função $I(r)$ é dada pela Equação 3:

$$I(r) = I_0 e^{-\gamma r} \tag{3}$$

Como a atratividade é proporcional ao seu brilho, $\beta(r)$ é dado pela Equação 4:

$$\beta(r) = \beta_0 e^{-\gamma r} \tag{4}$$

onde β_0 é a atratividade quando $r = 0$. Em [Durkota, 2011] é proposta uma modificação da Equação 4 para uma computação mais rápida, uma vez que operações exponenciais podem ser caras. Essa proposta é mostrada na Equação 5.

$$\beta(r) = \frac{\beta_0}{1 + \gamma r^2} \tag{5}$$

No presente trabalho, duas versões de FA: Simple Binary Firefly Algorithm (SBFA) e Simple Binary Firefly Algorithm With Influencers (SBFAWI), foram propostas. Ambas consistem de uma população de N vagalumes gerados aleatoriamente e propõem duas mudanças nas Equações 4 e 5. A primeira delas faz a inclusão do termo de aleatoriedade α, de modo a permitir que os vagalumes tenham ainda um poder de escolha, isto é, mesmo que eles sejam atraídos por outros vagalumes, ainda há a possibilidade dele não querer seguir este outro vagalume. A outra mudança considera que todo vagalume tem um valor de atração mínimo β_{min}. Desta forma, as Equações 4 e 5 se tornam as Equações 6 e 7, respectivamente. Essas duas versões utilizam a Equação 7 com $r = 1$. Pelos testes empíricos, não houve diferença nos resultados ao calcular r.

$$\beta(r) = \beta_{min} + (\beta_0 - \beta_{min})e^{-\gamma \alpha r} \tag{6}$$

$$\beta(r) = \beta_{min} + \frac{\beta_0 - \beta_{min}}{1 + \gamma \alpha r^2} \tag{7}$$

O valor de α é calculado como sendo:

$$\alpha = \alpha_0 * k^t \tag{8}$$

onde $k \in (0, 1)$ é um parâmetro que controla a redução de α ao longo do tempo t. O valor de t é o número da iteração atual do algoritmo. Em ambas implementações SBFA e SBFAWI, se a solução não apresenta melhora de uma iteração para a outra, o valor de t assumido é sua metade. Ainda, caso a solução não melhore por 5 iterações, o valor de t assumido é sua quarta parte. Tais variações tendem a aumentar a aleatoriedade de modo a diversificar a busca, saindo de ótimos locais.

Algoritmo 1 inicializarSBFAWI(U_{ord})

Entrada: U_{ord} // conjunto de itens ordenado decrescentemente de acordo com a razão lucro/peso
Saída: H // população de vagalumes
Início:
1. H ← População de vagalumes gerada aleatoriamente, ordenada* do melhor para o pior indivíduo;
2. $J \leftarrow \Psi N$ primeiros indivíduos de H;
3. H ← H – J;
4. $I \leftarrow \emptyset$; // população de *influencers* inicialmente vazia
5. i_0 ← Indivíduo gerado pela escolha dos primeiros itens de U_{ord}, enquanto a solução se mantiver factível;
6. $I \leftarrow \Psi N$ indivíduos distribuídos no espaço de busca conforme a distância de *hamming*** tendo i_0 como solução inicial;
7. H ← H + I;
8. **retorna** H;

Fim

***Ordenação feita de acordo com a factibilidade e o valor de função objetivo dos vagalumes.
****O valor inicial para a distância de *hamming* (vdh) foi calculado como sendo a divisão inteira do nº de variáveis de decisão por ΨN. Se o resultado dessa divisão for zero, tem-se $vdh = 1$. Quando todas as soluções possíveis tiverem sido geradas tendo vdh de distância entre si, caso ainda faltem *influencers*, diminui-se vdh em uma unidade e gera-se novas soluções a partir da última solução computada. O processo é repetido até gerar todos os *influencers* requeridos. O objetivo é que todos *influencers* sejam distintos, contudo dependendo dos parâmetros de entrada pode haver necessidade de repetir *influencers*.

A versão SBFAWI consiste na divisão da população de vagalumes em dois grupos: um grupo geral e um grupo especial. Os vagalumes do grupo especial são chamados de *influencers* e possuem pelo menos uma característica especial que os diferenciam do restante do grupo e os tornam uma referência diante do grupo. Na natureza, tal característica pode estar relacionada à espécie, sexo, tamanho ou idade, por exemplo. Neste trabalho, define-se que os vagalumes têm 5%

a mais de chance de seguirem um vagalume *influencer* do que seguir qualquer outro vagalume. Um parâmetro $\Psi \in [0, 1]$ foi adicionado para indicar a proporção de *influencers* na população. Outras características de SBFAWI incluem o modo de inicialização e o movimento dos *influencers*: o modo de inicialização de SBFAWI pode ser visualizado no Algoritmo 1; o movimento dos *influencers* foi definido de forma que eles nunca pioram em termos da função objetivo e da factibilidade. Para isso, uma cópia do *influencer* é criada e realiza-se o movimento. Se a cópia for infactível, ela passa por um processo de correção com base no vetor de itens reordenado decrescentemente de acordo com a razão lucro/peso. Nesse processo de correção, os itens com as menores razões são removidos da cópia até que se alcance a factibilidade. Se a cópia encontrar em uma melhor solução, o *influencer* se torna essa cópia após o movimento.

Como a meta-heurística FA foi proposta inicialmente para problemas contínuos, é necessário que haja uma adaptação quanto à forma como é dado o movimento dos vagalumes. O Algoritmo 2, proposto neste trabalho, mostra como realiza-se o movimento no SBFA e no SBFAWI. O parâmetro $\lambda \in [0, 1]$ indica a probabilidade de uma troca de bits aleatória acontecer. Empiricamente, verificou-se que valores para $\lambda \leq 0.05$ geram melhores resultados.

Algoritmo 2 mover(A, B) // movimento de um vagalume A em direção a um vagalume B

Entrada: A, B // dois vagalumes, A e B.
Saída: A // vagalume A com os bits alterados.
Início:
1. **para cada** variável de decisão a_i em A, **repita**
2. **se** $random(0, 1) + \alpha < \beta$ **então** // $random(0, 1)$ gera um valor aleatório entre 0 e 1.
3. Altere a_i para ser igual à i-ésima variável de decisão de B, isto é, $a_i = b_i$;
4. **senão**
5. **se** random(0, 1) < λ **então**
6. Inverta o bit de a_i // simula um movimento aleatório.
7. **fim se**
8. **fim se**
9. **fim para**
10. **retorna** A;

Fim

Note pelas Equações 7 e 8 que à medida que t aumenta, α diminui e β aumenta. Com isso, o Algoritmo 2 sugere que em alguma iteração t_f os vagalumes estarão sujeitos apenas à atratividade, tendo a possibilidade de realizar um movimento aleatório quase nula. Assim, é razoável interromper a execução do algoritmo a partir da iteração t_f. Uma forma de calcular o valor de t_f é usando as Equações 7 e 8. Note que β na Equação 7 atinge um valor próximo de seu valor máximo quando $\gamma\alpha = \epsilon$, onde ϵ é tão próximo de zero quanto se queira. Fazendo $\epsilon = 10^{-8}$, substituindo α por $\frac{\epsilon}{\gamma}$ e isolando t na Equação 8, t_f é dado pela Equação 9.

$$t_f = \log_k \frac{\epsilon}{\gamma\alpha_0} \Rightarrow t_f = \log_k \frac{10^{-8}}{\gamma\alpha_0} \tag{9}$$

A Equação 9 pode ser usada para calcular o número de iterações para o algoritmo, após configurados os valores de k, γ e α_0. O valor obtido é apenas um valor de referência para quando o algoritmo irá prezar pela intensificação. Estratégias para a redução de t na Equação 8, como discutidas anteriormente, podem ter um impacto positivo ao incentivar uma diversificação caso as intensificações não estejam melhorando as soluções. Neste caso, para as estratégias adotadas, tomar um número de iterações igual a quatro vezes o valor de t_f obtido na Equação 9 ainda é adequado.

5. Resultados e Comparações

O computador utilizado para os testes foi um notebook modelo Acer Aspire Nitro 5 AN515-51-50U2, com CPU Intel Core I5-7300HQ Quad Core, 2.50 GHz, Sistema Operacional

Windows 10 Home – 64 bits e RAM 8GB DDR4 2400MHz. A linguagem de programação utilizada foi a linguagem Java.

O conjunto de instâncias do Problema da Mochila 0-1 considerado para os testes foi retirado de [Xiang et al., 2014] (Kp1 ao Kp12) e [Pisinger, 2015] (Kp13 – Nome original: knapPI_1_100_1000_1). Tais instâncias, apesar de serem pequenas e médias, possuem características interessantes que podem se apresentar como um desafio para alguns algoritmos de otimização. Uma dessas características diz respeito à alta correlação positiva entre seus dados de entrada, que está presente nas instâncias Kp4, Kp6, Kp7, Kp8 e Kp11. De acordo com [Pisinger, 2005], quanto mais correlacionados são os dados de uma instância, mais difícil é a sua resolução.

Para cada instância, 50 execuções independentes para cada implementação foram realizadas. Para cada execução as soluções iniciais foram geradas aleatoriamente, com exceção dos *influencers* de SBFAWI, gerados pelo Algoritmo 1. O tratamento das violações à restrição do problema (Equação 2) se deu por meio da aplicação de penalidades à função objetivo. Assim, soluções infactíveis têm seus valores de função objetivo calculados como:

$$f(\vec{x}) \leftarrow f(\vec{x}) - \omega g^2 \tag{10}$$

onde ω é o parâmetro correspondente ao grau da penalidade e g corresponde à distância da factibilidade, sendo calculado como o valor absoluto da diferença $w_{max} - \sum_{i=1}^{n} x_i w_i$. Neste trabalho, $\omega = 3$ foi utilizado para todas execuções, exceto às referentes às instâncias Kp3, Kp4, Kp11 e Kp12. Para Kp3 e Kp4 adotou-se $\omega = 50$, enquanto $\omega = 2.1$ foi usado em Kp11 e Kp12 a fim de melhorar a taxa de sucesso dos algoritmos. Essas mudanças no valor de ω refletem uma característica deste método de penalização que é comentada em [Xiang, 2014], que diz que uma grande desvantagem da função de penalização é que o parâmetro ω é problema-dependente.

Os parâmetros dos algoritmos são mostrados na Tabela 1. Note que as implementações de TS precisaram ter diferentes conjuntos de parâmetros para cada instância. Isto se deve, pois, como mencionado anteriormente, as características dos problemas podem ter impactos sobre alguns algoritmos de otimização. Em particular, os desempenhos de TS e TSWPR demonstraram ter uma forte relação entre seus parâmetros e a instância a ser resolvida.

Tabela 1: Parâmetros utilizados

SBFA e SBFAWI	TS e TSWPR		
	Instância	|V|	|T|
$\beta_0 = 0.995$	Kp1	8	2
$\beta_{min} = 0.3$	Kp2	12	1
$\gamma = 1000000$	Kp3	4	2
$\alpha_0 = 1$	Kp4	4	2
$k = \begin{cases} 0.95 \text{ para Kp11, Kp12 e Kp13} \\ 0.55 \text{ para as demais instâncias} \end{cases}$	Kp5	12	3
	Kp6	6	4
$\lambda = 0.01$	Kp7	3	3
$\Psi = 0.25$	Kp8	20	4
$N = 125$	Kp9	5	4
	Kp10	12	2
	Kp11	14	8
	Kp12	33	7
	Kp13	50	7

A Tabela 2 mostra a comparação geral entre as quatro implementações deste trabalho. Os valores nesta tabela foram arredondados para até quatro casas decimais. Das 50 execuções independentes, a coluna "Taxa de sucesso" mostra quantas delas determinado algoritmo conseguiu alcançar uma solução ótima. TSWPR, SBFA e SBFAWI alcançaram o ótimo de todas as instâncias. Apenas SBFAWI alcançou 100% de taxa de sucesso em todas elas. A instância Kp11 apresentou ser um desafio para os algoritmos. TS não alcançou seu ótimo e SBFA e SBFAWI apresentaram uma taxa de sucesso bem superior para esta instância em relação à TSWPR. Note, também, que TS

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings

e TSWPR apresentaram um melhor tempo de execução para a maioria das instâncias, principalmente para Kp13.

A Tabela 3 mostra a comparação entre as melhores soluções encontradas por cada uma das implementações. A melhor solução, retirada de uma das execuções independentes, foi definida como sendo aquela que alcança o ótimo com o menor número de avaliações da função objetivo.

Tabela 2: Comparação geral entre as quatro implementações deste trabalho para instâncias pequenas e médias do Problema da Mochila 0-1

Instância (n, w_{max})	Solução Ótima	Algoritmo	Melhor Solução	Pior Solução	Mediana	Média	Desvio padrão	Taxa de sucesso	Tempo médio (s)
Kp1 (10, 269)	295	TS	295	295	295	295	0	100%	0.0002
		TSWPR	295	295	295	295	0	100%	0.0007
		SBFA	295	295	295	295	0	100%	0.0250
		SBFAWI	295	295	295	295	0	100%	0.0291
Kp2 (20, 878)	1024	TS	1024	1024	1024	1024	0	100%	0.0005
		TSWPR	1024	1024	1024	1024	0	100%	0.0010
		SBFA	1024	1024	1024	1024	0	100%	0.0553
		SBFAWI	1024	1024	1024	1024	0	100%	0.0631
Kp3 (4, 20)	35	TS	35	35	35	35	0	100%	0.0000
		TSWPR	35	35	35	35	0	100%	0.0002
		SBFA	35	35	35	35	0	100%	0.0023
		SBFAWI	35	35	35	35	0	100%	0.0027
Kp4 (4, 11)	23	TS	23	23	23	23	0	100%	0.0001
		TSWPR	23	23	23	23	0	100%	0.0001
		SBFA	23	23	23	23	0	100%	0.0024
		SBFAWI	23	23	23	23	0	100%	0.0026
Kp5 (15, 375)	481.0694	TS	481.0694	481.0694	481.0694	481.0694	0	100%	0.0003
		TSWPR	481.0694	481.0694	481.0694	481.0694	0	100%	0.0007
		SBFA	481.0694	481.0694	481.0694	481.0694	0	100%	0.0363
		SBFAWI	481.0694	481.0694	481.0694	481.0694	0	100%	0.0087
Kp6 (10, 60)	52	TS	52	52	52	52	0	100%	0.0003
		TSWPR	52	52	52	52	0	100%	0.0009
		SBFA	52	52	52	52	0	100%	0.0154
		SBFAWI	52	52	52	52	0	100%	0.0062
Kp7 (7, 50)	107	TS	107	105	107	106.92	0.3919	96%	0.0001
		TSWPR	107	107	107	107	0	100%	0.0002
		SBFA	107	107	107	107	0	100%	0.0037
		SBFAWI	107	107	107	107	0	100%	0.0037
Kp8 (23, 10000)	9767	TS	9767	9765	9767	9766.94	0.3105	96%	0.0007
		TSWPR	9767	9767	9767	9767	0	100%	0.0014
		SBFA	9767	9767	9767	9767	0	100%	0.1032
		SBFAWI	9767	9767	9767	9767	0	100%	0.0921
Kp9 (5, 80)	130	TS	130	130	130	130	0	100%	0.0001
		TSWPR	130	130	130	130	0	100%	0.0001
		SBFA	130	130	130	130	0	100%	0.0028
		SBFAWI	130	130	130	130	0	100%	0.0032
Kp10 (20, 879)	1025	TS	1025	1025	1025	1025	0	100%	0.0004
		TSWPR	1025	1025	1025	1025	0	100%	0.0009
		SBFA	1025	1025	1025	1025	0	100%	0.0563
		SBFAWI	1025	1025	1025	1025	0	100%	0.0533
Kp11 (50, 1000)	3119	TS	3114	3082	3099	3099.40	6.6843	0%	4.0773
		TSWPR	3119	3096	3108	3107.06	3.9162	2%	3.7504
		SBFA	3119	3114	3119	3118.46	1.2839	78%	5.0897
		SBFAWI	3119	3119	3119	3119	0	100%	4.5227
Kp12 (100, 6718)	26559	TS	26559	26535	26559	26553.62	7.2990	56%	30.1616
		TSWPR	26559	26535	26559	26554.42	6.8937	62%	31.7966
		SBFA	26559	26553	26559	26557.32	2.6934	72%	33.3345
		SBFAWI	26559	26559	26559	26559	0	100%	31.0196
Kp13 (100, 995)	9147	TS	9147	9147	9147	9147	0	100%	0.1308
		TSWPR	9147	9147	9147	9147	0	100%	0.1359
		SBFA	9147	8940	9147	9142.86	28.9800	98%	10.0510
		SBFAWI	9147	9147	9147	9147	0	100%	9.4114

Para cada instância, o vetor correspondente à solução ótima encontrada é listado.

De acordo com a Tabela 3, duas soluções ótimas distintas foram encontradas para a instância Kp6. As colunas referentes ao momento em que o ótimo foi encontrado mostram o melhor

caso dos algoritmos. Note que para a instância Kp3, a solução ótima foi capaz de ser encontrada com apenas uma iteração dos algoritmos. Outras instâncias como Kp11, Kp12 e Kp13 exigiram um maior esforço computacional dos algoritmos até que o ótimo fosse encontrado.

Tabela 3: Comparação das melhores soluções encontradas pelas quatro implementações deste trabalho para instâncias pequenas e médias do Problema da Mochila 0-1

Instância (n, w_{max})	Soluções ótimas encontradas pelos quatro algoritmos $(\vec{x}, f(\vec{x}), w_{total})$	Algoritmo	Iterações	Iteração em que o ótimo foi encontrado	Avaliações da função objetivo	Avaliações da função objetivo quando o ótimo foi encontrado
Kp1 (10, 269)	(0111000111, 295, 269)	TS	100	16	801	129
		TSWPR	100+	9	2307	73
		SBFA	5	1	37303	194
		SBFAWI	5	1	41431	504
Kp2 (20, 878)	(11111111111110101011, 1024, 871)	TS	100	18	1201	217
		TSWPR	100+	22	2203	265
		SBFA	6	5	40574	31262
		SBFAWI	6	1	45926	4968
Kp3 (4, 20)	(1101, 35, 18)	TS	10	1	41	5
		TSWPR	10+	1	243	5
		SBFA	1	1	8053	335
		SBFAWI	1	1	8174	47
Kp4 (4, 11)	(0101, 23, 11)	TS	10	6	41	25
		TSWPR	10+	1	41	1
		SBFA	1	1	8043	374
		SBFAWI	1	1	7608	3
Kp5 (15, 375)	(001010110111011, 481.069368, 354.960784)	TS	100	7	1201	85
		TSWPR	100+	24	1953	289
		SBFA	5	5	35872	30180
		SBFAWI	1	1	8536	188
Kp6 (10, 60)	(0010111111, 52, 57) e (0011101000, 52, 60)	TS	200	69	1201	415
		TSWPR	200+	2	1703	13
		SBFA	3	1	23409	269
		SBFAWI	1	1	8422	235
Kp7 (7, 50)	(1001000, 107, 50)	TS	200	60	601	181
		TSWPR	200+	120	953	361
		SBFA	2	1	7853	175
		SBFAWI	1	1	7183	38
Kp8 (23, 10000)	(11111111001000011000000, 9767, 9768)	TS	100	46	2001	921
		TSWPR	100+	38	6609	761
		SBFA	10	5	70466	32274
		SBFAWI	8	1	61821	6848
Kp9 (5, 80)	(11110, 130, 60)	TS	100	4	501	21
		TSWPR	100+	4	753	21
		SBFA	1	1	8289	462
		SBFAWI	1	1	8048	13
Kp10 (20, 879)	(11111111101111010111, 1025, 871)	TS	100	13	1201	157
		TSWPR	100+	18	2203	217
		SBFA	6	5	41087	31539
		SBFAWI	5	1	39570	2479
Kp11 (50, 1000)	(11010101111010011011011111 11110000101101100000010, 3119, 1000)	TS	500000	---	7000001	---
		TSWPR	500000+	500004	7017515	7000153
		SBFA	280	43	1317890	323713
		SBFAWI	180	8	1422090	58338
Kp12 (100, 6718)	(11111111111111111111111111 11111111111111111110111111 11010001011011011111100011 10111000000000000001, 26559, 6717)	TS	1000000	4475	33000001	147676
		TSWPR	1000000+	6584	33017515	217273
		SBFA	680	64	5649520	477567
		SBFAWI	600	34	5583566	272573
Kp13 (100, 995)	(00000010001001000000000101 00001010000110000000010000 10000001000000000000000000 00000000000000000000, 9147, 985)	TS	5000	51	250001	2551
		TSWPR	5000+	53	265013	2651
		SBFA	230	47	1578770	349938
		SBFAWI	180	34	1553847	279322

+: Significa que mais iterações além daquele número podem ter sido feitas. Isso se dá pois a estratégia de *path relinking* é ativada para cada bit diferente com relação à solução inicial e à solução final obtida pelo processo normal de TS.

Apesar da instância Kp13 ter uma dimensão maior do que a de Kp11, os algoritmos conseguiram ter um melhor desempenho ao resolvê-la. Isto se dá pois Kp11 é uma instância com uma alta correlação entre seus dados de entrada, enquanto Kp13 possui uma baixa correlação. A Figura 2 mostra a correlação das instâncias Kp11 e Kp13. A instância Kp11 possui uma alta correlação de Pearson com $R = 0.8515$, enquanto que a instância Kp13 possui uma baixa correlação, com $R = -0.0418$.

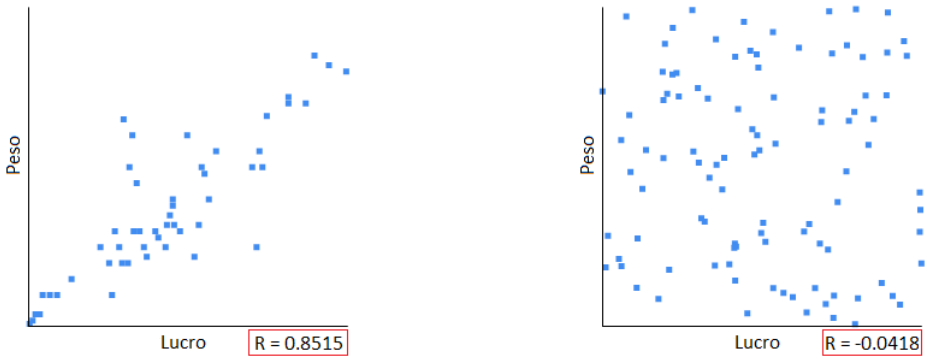


Figura 2: Gráficos que representam as correlações das instâncias (a) Kp11 e (b) Kp13.

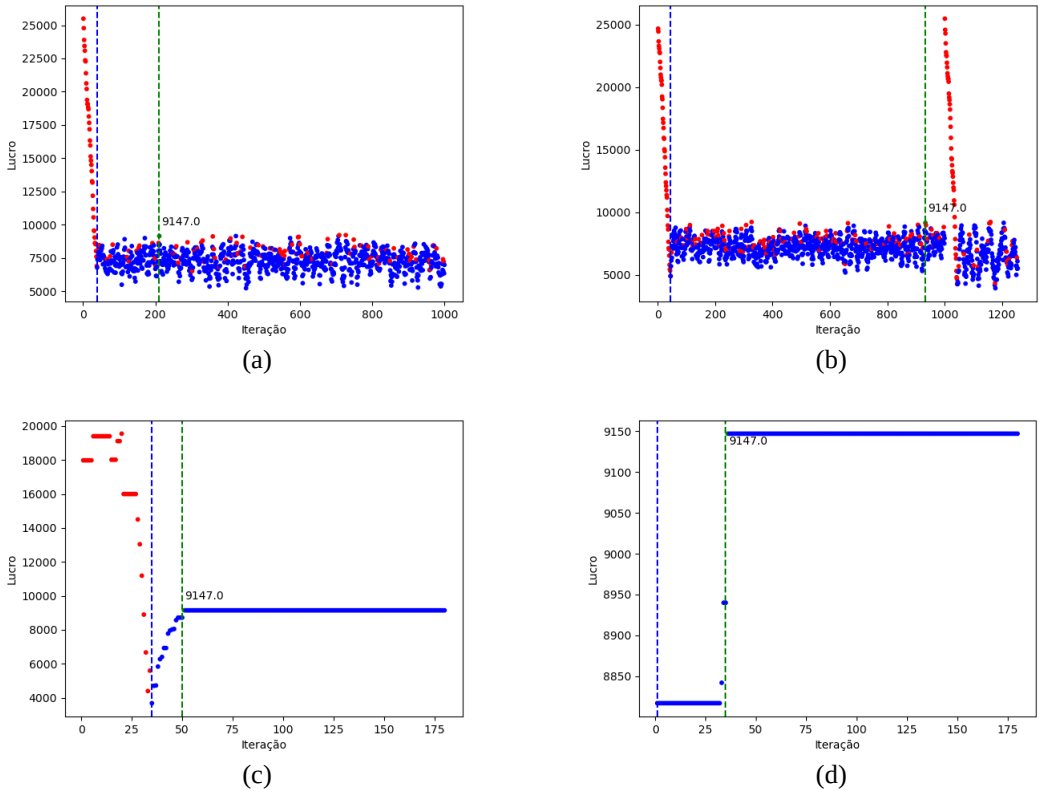


Figura 3: Gráficos que representam a convergência dos algoritmos implementados para uma execução independente da instância Kp13: (a) TS, (b) TSWPR, (c) SBFA e (d) SBFAWI.

Os comportamentos dos algoritmos implementados são ilustrados na Figura 3 para uma execução independente da instância Kp13. As Figuras 3a e 3b mostram as últimas soluções computadas a cada iteração para TS e TSWPR, respectivamente. As Figuras 3c e 3d mostram a melhor solução presente na população de vagalumes ao fim de cada iteração para SBFA e SBFAWI, respectivamente. As bolas vermelhas, azuis e verdes presentes nos gráficos indicam soluções infactíveis, soluções factíveis e a primeira solução ótima encontrada, respectivamente. A linha vertical azul em cada gráfico indica o momento em que a primeira solução factível foi encontrada e a linha vertical verde em cada gráfico indica o momento em que a solução ótima foi encontrada. A solução ótima, em todos os casos é guardada, permitindo que os algoritmos aceitem pioras para explorar outros locais do espaço de busca. Nas Figuras 3c e 3d, é possível notar que a solução ótima (ou uma muito próxima da ótima) ficou presente na população de vagalumes após ela ser encontrada. A Figura 3b mostra que a solução ótima foi encontrada antes da estratégia de *path relinking* ser ativada.

6. Conclusão

Neste trabalho foram feitas quatro implementações capazes de resolver problemas binários como o Problema da Mochila 0-1. Elas possuem como base as meta-heurísticas Firefly Algorithm (FA) e Tabu Search (TS), são elas: Tabu Search (TS), Tabu Search With Path Relinking (TSWPR), SBFA (Simple Binary Firefly Algorithm) e SBFAWI (Simple Binary Firefly Algorithm With Influencers).

Um conjunto de 13 instâncias do Problema da Mochila 0-1, pequenas e médias, incluindo uma com uma alta correlação entre os dados de entrada (Kp11), foi utilizado para avaliar o desempenho dessas quatro implementações. Os resultados dos testes mostraram que SBFA, SBFAWI e TSWPR conseguiram alcançar as soluções ótimas de todas as instâncias deste conjunto. TS conseguiu alcançar as soluções ótimas de 12 das 13 instâncias, não obtendo o ótimo da instância Kp11.

SBFA e SBFAWI apresentaram uma taxa de sucesso superior a TS e TSWPR para as instâncias Kp11 e Kp12. A taxa de sucesso diz respeito à quantidade de execuções que alcançaram soluções ótimas. Por sua vez, TS e TSWPR mostraram um melhor desempenho nas métricas de tempo e número de avaliações da função objetivo para a maioria das instâncias.

A estratégia de *path relinking* em TSWPR se mostrou capaz de melhorar as soluções obtidas por TS. As estratégias de separação de classes, inicialização e movimento dos *influencers* aplicadas em SBFAWI foram capazes de gerar bons resultados, alcançando o ótimo em todas as execuções para todas as instâncias analisadas.

As implementações baseadas em TS demonstraram ter uma relação entre seus parâmetros e a instância a ser resolvida, sendo necessário adaptá-los para cada uma. É possível que haja outros conjuntos de parâmetros para estes algoritmos capazes de melhorá-los. Softwares de *tuning* podem ser utilizados para isso.

O método de tratamento de restrições utilizado foi o método de penalização. O parâmetro de penalidade ω teve um valor constante para a maioria das instâncias. No entanto, para as instâncias Kp3, Kp4, Kp11 e Kp12 foi necessário modificar ω a fim de melhorar o desempenho dos algoritmos na busca pela solução ótima. Isso mostra que existe uma dependência do método de penalização com as entradas do problema. Assim, dependendo do valor de ω, os algoritmos podem ter dificuldades ou até mesmo serem impedidos de encontrar a solução ótima para algumas instâncias.

Futuramente, pretende-se explorar novas formas de tratamento de restrições e reduzir o custo computacional de SBFAWI. Pretende-se também aplicar estes algoritmos em um problema real, avaliando o desempenho destes na obtenção de soluções satisfatórias.

Referências

Abdel-Basset, M., El-Shahat, D. e Sangaiah, A. K. (2019). A modified nature inspired meta-heuristic whale optimization algorithm for solving 0–1 knapsack problem. *International Journal of Machine Learning and Cybernetics*, 10:495–514.

Basheer, G. T. e Algamal, Z. Y. (2019). Nature-inspired optimization algorithms in knapsack problem: A review. *Iraqi Academic Scientific Journals*, 16:1–18.

Chou, Y. H., Yang, Y. J., e Chiu, C. H. (2011). Classical and quantum-inspired Tabu search for solving 0/1 knapsack problem. In *IEEE International Conference on Systems, Man, and Cybernetics*, p. 1364-1369, Anchorage, AK, IEEE.

Durkota, K. (2011). Implementation of a Discrete Firefly Algorithm for the QAP Problem within the SEAGE Framework. Tese (Bacharelado). Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Cybnertics.

Feng, Y., Wang, G., Deb, S. e Zhao, XJ. (2017). Solving 0–1 knapsack problem by a novel binary monarch butterfly optimization. *Neural Computing and Applications*, 28:1619–1634.

Fréville, A (2004). The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155:1–21.

Glover, F. e Laguna, M. (1998). Tabu Search, em Du DZ., Pardalos P.M. (eds.). In Handbook of Combinatorial Optimization. Springer, Boston, MA.

Glover, F. (1989). Tabu Search – Part I. *ORSA Journal on Computing*, 1:190–206.

Glover, F. (1990). Tabu Search – Part II. *ORSA Journal on Computing*, 2:4–32.

Pisinger, D. e Toth P. (1998). Knapsack Problems, em Du DZ., Pardalos P.M. (eds.). In Handbook of Combinatorial Optimization. Springer, Boston, MA.

Pisinger, D. (2000). A Minimal Algorithm for the Bounded Knapsack Problem. *INFORMS Journal on Computing*, 12:75–82.

Pisinger, D. (2005). Where are the hard knapsack problems? *Computers & Operations Research*, 32:2271–2284.

Pisinger, D. (2015). David Pisinger's optimization codes. `http://hjemmesider.diku.dk/~pisinger/codes.html`. Acessado: 2020-06-08.

Qi, H. (2006). Integrated Watershed Management and Agricultural Land-use Optimization under Uncertainty. Tese (Doutorado). University of Mississippi.

Xiang, W., An, M., Li, Y., He, R. e Zhang, J. (2014). A Novel Discrete Global-Best Harmony Search Algorithm for Solving 0-1 Knapsack Problems. Discrete Dynamics in Nature and Society, 2014:1-12.

Yang, X. (2010). Firefly Algorithm, Stochastic Test Functions and Design Optimisation. *International Journal of Bio-Inspired Computation*, 2:78–84.

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings