Atividade Orientada de Ensino: Nova abordagem para o Algoritmo Colônia de Abelhas Artificiais aplicado ao problema da Mochila Multidimensional Binária

Amanda Ayumi Yamashita, Luiz Eduardo Batista Garcia

Resumo

A mochila multidimensional binária é um problema de otimização muito explorado no âmbito da computação. Mesmo sendo um problema conhecido de maneira abrangente, sua solução não é trivial, uma vez que pertence à classe dos problemas $\mathcal{NP}$-difíceis, isto é, não há um algoritmo que forneça uma solução que possa ser verificada em tempo polinomial. Assim, a fim de buscar soluções aproximadas em menor tempo, faz-se uso das meta-heurísticas. Neste trabalho, utilizou-se a meta-heurística do Algoritmo Colônia de Abelhas Artificiais (ABC - do inglês *Artificial Bee Colony*) modificada. Nesse algoritmo iterativo, a colônia é simplificada em três classes de abelhas: as empregadas, as observadoras e as escoteiras. Durante as iterações, elas procuram a melhor solução alcançável, que se refere à fonte de alimento procurada pela colônia. Com base nessa meta-heurística, desenvolveu-se um algoritmo ABC discreto, visto que a mochila multidimensional é de ordem binária e o ABC foi proposto inicialmente para ser aplicado a problemas contínuos, a fim de tentar alcançar bons resultados para a questão proposta. O algoritmo apresentado foi avaliado com base na resolução de problemas de otimização encontrados na literatura. Os resultados obtidos foram armazenados e processados para realizar comparações quantitativas quanto à aplicabilidade do algoritmo no problema proposto. Para comprovar a eficiência, foram calculados as médias e os desvios-padrões dos resultados, a fim de validá-los. Por fim, pode-se dizer que o algoritmo obteve resultados promissores na maioria dos casos testados, aproximando-se de outras meta-heurísticas aplicáveis existentes.

1 Introdução

O problema da mochila, KP (*Knapsack Problem*), é um problema de otimização combinatória frequentemente abordado na área da computação, em que é necessário colocar em uma mochila n objetos de diferentes pesos e valores sem exceder a sua capacidade máxima m. Este problema é $\mathcal{NP}$-difícil, ou seja, não é conhecido um algoritmo polinomial para sua solução exata e possuem uma ampla gama de aplicações em problemas computacionais, em programação discreta e em otimização combinatória.

Dito isso, um possível caminho para encontrar a solução dos problemas $\mathcal{NP}$-difícil é por meio de heurísticas e meta-heurísticas, sendo grande parte delas inspiradas nos costumes de diversos animais na natureza.

O Algoritmo Colônia de Abelhas Artificiais (ABC) foi apresentado por Karaboga [3] em 2005 para a resolução de problemas de otimização multimodal e multidimensional [1]. O ABC é um algoritmo com inspiração natural que atua aplicando conceitos de inteligência coletiva apresentados por abelhas reais durante o processo de forrageamento, isto é, a busca e a exploração de recursos e fontes de alimento.

2 Nova Ideia Implementada

Este trabalho é uma variação do Algoritmo Colônia de Abelhas Artificiais proposto por Karaboga [3,4] juntamente com o Algoritmo Colônia de Abelhas Artificiais discreto aplicado ao problema da Mochila Multidimensional Binária apresentação por [7]. Apesar de apresentar a mesma ideia base e de possuir os mesmos passos de execução apresentada anteriormente em [7], esta abordagem se diferencia na fase empregada do ABC. Nesta fase, durante a busca local, foi utilizado o conceito de vizinhança como sendo uma outra abelha artificial, conforme a equação:

$$u_i = x_i + \Phi(x_i - x_j) \tag{1}$$

Embora a (1) seja mesma utilizada na abordagem anterior, ou seja, no Trabalho de Conclusão de Curso, alterou-se sua interpretação. Nesse caso, u_i é um valor candidato para substituir x_i, $\{i, j\}$ $\in$ {1,2,...,n} são índices escolhidos aleatoriamente, $i \neq j$ e Φ é um valor aleatório entre $[-1, 1]$ [8]. Seguindo a solução proposta originalmente por Karaboga [3], o vizinho foi tomado utilizando da posição j de uma abelha vizinha , isto é, uma nova versão da solução original é criada baseada na alteração da posição x_i do vetor solução (abelha), utilizando a Equação (1), onde x_i é a própria posição a ser alterada e x_j é a posição j de uma abelha vizinha a abelha em questão. E, para além disso, adotou-se um número limite de vezes de alterações na própria solução, este é denominado limite local.

$$u_i = \begin{cases} 1, se\ u_i >= 1; \\ 0, se\ u_i <= 0. \end{cases} \tag{2}$$

De maneira análoga, a variável ajustada u_i da Equação (1) é um número real. Logo, aplica-se a Equação (2) para normalizar o resultado decidindo entre 0 e 1.

Proposta a variação descrita acima, o Algoritmo Colônia de Abelhas Artificiais segue sua execução conforme apresentado no algoritmo.1.

Algoritmo .1: ABC AOE

```
Inicialize a população de abelhas;
repita
    Posicione as abelhas empregadas em suas fontes de alimento;
    Testa viabilidade das soluções apresentadas pelas abelhas empregadas;
    Explora vizinhança;
    Calcule a probabilidade de escolha da fonte por abelhas observadoras;
    Posicione as abelhas observadoras nas fontes conforme seus valores de aptidão;
    Interrompa a exploração de fontes esgotadas;
    Envie abelhas escoteiras para buscar novas fontes de alimento;
    Memorize a melhor fonte encontrada;
até Número máximo de ciclos;
```

3 Resultados Experimentais

Para a avaliação do algoritmo proposto neste trabalho, três diferentes conjuntos de testes encontrados na literatura foram usados: o conjunto SAC-94, a biblioteca ORLIB e o conjunto GK. Os programas foram executados em uma máquina com a seguinte configuração:

- Processador Intel Core 2 Duo E8400 de 2 núcleos e 3GHz de frequência
- 3.8 GB de memória;
- 250 GB de cpacidade de disco;
- Gráficos Ilvmpipe (LLVM 12.0.0, 128 bits)

3.1 Resultados obtidos

O biblioteca SAC-94 é composta por 6 conjuntos e 54 problemas ao todo. Esses conjuntos são baseados em problemas reais apresentados em artigos ciêntificos, contento instâncias entre 10 e 105 itens e 2 e 30 recursos, ou seja, n e m respectivamente. Os resultados obtidos com as execuções dos testes para está biblioteca podem ser vistos na Tabela 1.

Tabela 1: Resultados do algoritmo ABC para as instâncias de SAC-94.

Conjunto	Instâncias	Iteração	Ótimos	*Gap* Mínimo	*Gap* Médio	Tempo Médio(s)
hp	2	1000	1	0.0000	1.4551	0.0768
		5000	3	0.0000	1.0936	0.3920
		10000	30	0.0000	0.6277	0.4618
pb	6	1000	28	0.0000	3.6096	0.1509
		5000	49	0.0000	1.8830	0.7192
		10000	91	0.0000	1.4468	1.5168
pet	6	1000	60	0.0000	0.8622	0.0673
		5000	118	0.0000	0.3552	0.2098
		10000	116	0.0000	0.2869	0.4412
sento	2	1000	0	2.8663	9.6192	0.4766
		5000	0	0.3989	4.1762	2.3797
		10000	0	0.1490	3.4268	4.7163
weing	8	1000	0	0.1113	4.4345	0.0950
		5000	60	0.0000	1.9679	0.3960
		10000	99	0.0000	1.4528	0.6836
weish	30	1000	28	0.0000	7.0163	0.1634
		5000	134	0.0000	2.8991	0.7631
		10000	193	0.0000	1.9310	1.4400

Conforme visto na Tabela 1, o ABC encontrou diversas vezes soluções equivalente as soluções de referência para as instâncias de SAC-94. No que se refere as variações do número de iterações globais da população de abelhas, a variação do tempo médio é aproximadamente dobrada ao definir 5000 e 10000 cilos. Assim, temos uma variação de 0.0698, 0.7976, 0.2314, 2.3366, 0.2876 e 0.6769 segundos para hp, pb, pet, sento, weing e weish respectivamente.

Logo, pode-se concluir que quanto maior o número de iterações, mais robusto é o algoritmo, minimizando o erro (*Gap*) e aumentando a número de otimizações encontradas. Além disso, conforme esperado, quanto mais iterações a população faz, aumenta-se também o tempo de execução.

Já biblioteca GK é composta por 11 conjunto de testes, contento instâncias com número de itens (n) de 100, 150, 200, 500, 1500 e 2500 e o número de recursos (m) variando de 15, 25, 50. Os resultados obtidos para os conjuntos GK podem ser vistos na Tabela 2 adiante.

Tabela 2: Resultados do algoritmo ABC para as instâncias de GK.

Conjunto	m	n	Iteração	Ótimos	*Gap* Mínimo	*Gap* Médio	Tempo Médio(s)
gk01	15	100	1000	0	3.0005	3.5590	0.3858
			5000	0	1.6198	2.9306	1.9235
			10000	0	1.9384	2.6730	3.8889
gk02	25	100	1000	0	2.5013	3.5515	0.5768
			5000	0	1.9960	2.8112	2.8859
			10000	0	1.5159	2.5232	6.0840
gk03	25	150	1000	0	2.9349	3.4046	0.8698
			5000	0	2.5283	2.9438	4.3017
			10000	0	2.1040	2.6238	8.9713
gk04	50	150	1000	0	2.5490	3.1368	1.6259
			5000	0	2.0114	2.6629	7.9570
			10000	0	1.9768	2.4727	15.9163
gk05	25	200	1000	0	3.0965	3.4970	1.1487
			5000	0	2.6598	3.0868	5.7355
			10000	0	2.3025	2.8592	11.9671
gk06	50	200	1000	0	2.1246	2.8241	2.1953
			5000	0	1.9943	2.4978	10.7177
			10000	0	2.0334	2.3710	21.4266
gk07	25	500	1000	0	3.4088	3.7157	2.9401
			5000	0	2.9456	3.3229	14.7648
			10000	0	2.6177	3.1092	31.0309
gk08	50	500	1000	0	2.6481	2.8562	5.6604
			5000	0	2.2386	2.6385	27.7581
			10000	0	2.2333	2.4955	55.3934
gk09	25	1500	1000	0	3.3933	3.6008	9.2126
			5000	0	3.1110	3.3741	46.0296
			10000	0	3.0817	3.2951	96.1265
gk10	50	1500	1000	0	2.6128	2.7427	17.7764
			5000	0	2.3563	2.5894	87.0356
			10000	0	2.3179	2.5081	173.7072
gk11	100	2500	1000	0	2.0519	2.1329	57.4450
			5000	0	1.9258	2.0552	290.0420
			10000	0	1.9069	2.0051	572.9691

Assim como verificado para a biblioteca SAC-94, os testes evidenciam uma melhora significativa ao aumentar o número de ciclos globais empregado as abelhas artificiais na busca por soluções ótimas. Assim, conforme aumenta-se a escala de ciclos, os valores de *Gap* mínimo e médio são menores, sendo de interesse *Gap* valendo 0 ou próximo de 0.

Por fim, quanto aos testes realizados na biblioteca ORLIB. Os programas desenvolvidos foram executados utilizando como entrada o conjunto de testes da biblioteca OR [72]. A biblioteca é composta de instâncias de problemas com 5, 10 ou 30 recursos e 100, 250 ou 500 itens.

Tabela 3: Resultados do algoritmo ABC para as instâncias de ORLIB com 5 restrições (OR5).

m	n	α	Iteração	Ótimos	*Gap* Mínimo	*Gap* Médio	Tempo Médio(s)
5	100	0.25	1000	0	8.9386	12.6275	0.2455
			5000	0	3.7073	8.7332	1.2656
			10000	0	3.2421	7.2586	2.4356
		0.50	1000	0	5.8831	10.4917	0.2198
			5000	0	1.5901	6.9874	1.1347
			10000	0	1.5202	5.7117	2.1794
		0.75	1000	0	2.6780	8.3562	0.2101
			5000	0	1.3881	5.5834	1.0872
			10000	0	0.6123	4.5148	2.0862
	250	0.25	1000	0	2.6780	10.3650	0.3364
			5000	0	1.3881	7.4201	1.7030
			10000	0	0.6123	6.1273	3.3414
		0.50	1000	0	2.6780	10.3791	0.3593
			5000	0	1.3881	7.5238	1.8079
			10000	0	0.6123	6.1788	3.5698
		0.75	1000	0	2.6780	9.5762	0.3723
			5000	0	1.3881	6.9449	1.8666
			10000	0	0.6123	5.6978	3.7005
	500	0.25	1000	0	2.6780	10.8813	0.5847
			5000	0	1.3881	8.1449	2.9199
			10000	0	0.6123	6.8000	5.8231
		0.50	1000	0	2.6780	11.0230	0.6219
			5000	0	1.3881	8.3430	3.1065
			10000	0	0.6123	6.9789	6.2071
		0.75	1000	0	2.6780	10.5067	0.6459
			5000	0	1.3881	7.9767	3.2241
			10000	0	0.6123	6.6712	6.4549

Tabela 4: Resultados do algoritmo ABC para as instâncias de ORLIB com 10 restrições (OR10).

m	n	α	Iteração	Ótimos	*Gap* Mínimo	*Gap* Médio	Tempo Médio(s)
10	100	0.25	1000	0	2.6780	10.7343	0.6174
			5000	0	1.3881	8.0806	3.0819
			10000	0	0.6123	6.7768	6.1742
		0.50	1000	0	2.6780	10.5477	0.5889
			5000	0	1.3881	7.8490	2.9389
			10000	0	0.6123	6.5686	5.8917
		0.75	1000	0	1.9172	10.0380	0.5650
			5000	0	1.0974	7.4343	2.8200
			10000	0	0.5690	6.2076	5.6566
	250	0.25	1000	0	1.9172	10.5208	0.5988
			5000	0	1.0974	7.8187	2.9921
			10000	0	0.5690	6.5453	5.9910
		0.50	1000	0	1.9172	10.5417	0.6074
			5000	0	1.0974	7.8347	3.0377
			10000	0	0.5690	6.5458	6.0759
		0.75	1000	0	1.9172	10.2174	0.6142
			5000	0	1.0974	7.5872	3.0740
			10000	0	0.5690	6.3293	6.1433
	500	0.25	1000	0	1.9172	10.6924	0.7306
			5000	0	1.0974	8.0357	3.6583
			10000	0	0.5690	6.7330	7.3013
		0.50	1000	0	1.9172	10.7764	0.7735
			5000	0	1.0974	8.1433	3.8763
			10000	0	0.5690	6.8337	7.7287
		0.75	1000	0	1.9172	10.5355	0.8083
			5000	0	1.0974	7.9744	4.0517
			10000	0	0.5690	6.6933	8.0740

Tabela 5: Resultados do algoritmo ABC para as instâncias de ORLIB com 30 restrições (OR30).

m	n	α	Iteração	Ótimos	*Gap* Mínimo	*Gap* Médio	Tempo Médio(s)
30	100	0.25	1000	0	1.9172	10.6419	0.8078
			5000	0	1.0974	7.9877	4.0492
			10000	0	0.5690	6.7070	8.0723
		0.50	1000	0	1.9172	10.5436	0.8033
			5000	0	1.0974	7.8591	4.0261
			10000	0	0.5690	6.5832	8.0302
		0.75	1000	0	1.9172	10.2663	0.8002
			5000	0	0.9658	7.6234	4.0106
			10000	0	0.4473	6.3738	8.0040
	250	0.25	1000	0	1.9172	10.5438	0.8620
			5000	0	0.9658	7.8245	4.3194
			10000	0	0.4473	6.5475	8.6208
		0.50	1000	0	1.9172	10.5672	0.9033
			5000	0	0.9658	7.8330	4.5276
			10000	0	0.4473	6.5479	9.0369
		0.75	1000	0	1.9172	10.3768	0.9422
			5000	0	0.9658	7.6839	4.7220
			10000	0	0.4473	6.4201	9.4266
	500	0.25	1000	0	1.9172	10.6743	1.1001
			5000	0	0.9658	7.9511	5.5128
			10000	0	0.4473	6.6596	11.0002
		0.50	1000	0	1.9172	10.7445	1.2019
			5000	0	0.9658	8.0298	6.0246
			10000	0	0.4473	6.7286	12.0184
		0.75	1000	0	1.9172	10.5950	1.2933
			5000	0	0.9658	7.9243	6.4816
			10000	0	0.4473	6.6421	12.9307

O comportamento do Algoritmo ABC ocorreu para as instâncias do OR de maneira similar ao constatado nas bibliotecas SAC-94 e GK, ou seja, o desempenho e o tempo de execução cresceram com o número de iterações, já o *gap* mínimo e médio diminuíram. Nenhum dos testes no conjunto OR alcançou o valor de referência ótimo.

Assim, conclui-se que, medida que as dimensões do problema aumentam para cada conjunto de teste (SAC-94, GK e OR), o comportamento do ABC se altera conforme as observações feitas anteriormente,ou seja, observou-se uma diminuição do *gap* à medida que o número de iterações cresce, indicando uma melhoria na qualidade das soluções geradas pelo algoritmo, a custo do tempo de execução, que aumentou à medida que o número de iterações cresceu.

4 Conclusão

Os resultados obtidos durante a realização deste estudo mostram que o ABC é uma abordagem promissora para lidar com problemas de otimização combinatória complexos, como o da mochila multidimensional binária. Durante o desenvolvimento deste estudo, foram realizados experimentos abrangentes, utilizando instâncias de teste já abordadas na literatura, com diversas características, incluindo variações no número de itens, pesos individuais e capacidades.

Além disso, a variação da configuração dos parâmetros do algoritmo em conjunto com a arranjo do algoritmo podem alterar a execução e resultados obtidos. Esta variação do ABC demonstrou um bom desempenho na otimização da função objetivo do MKP, assim como a apresentada em [7].

Referências

[1] DUARTE, G. R., "Um algoritmo inspirado em colônias de abelhas para otimização numérica com restrições". Dissertação Mestrado Acadêmico. Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas, 2005.

[2] DANTAS, B. A., "Metaheurísticas para o Problema da Mochila Multidimensional". Dissertação Doutorado. Universidade Federal de Mato Grosso do Sul, Faculdade de Computação, 2016.

[3] KARABOGA, D., "An idea based on honey bee swarm for numerical optimization". Erciyes University, Engineering Faculty, 2015.

[4] KARABOGA, D., BASTURK, B., "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm". Journal of Intelligent Manufacturing, 23, 4, 1001-1014, 2012.

[5] SUNDAR, S., SINGH, A., ROSSI, R., "An Artificial Bee Colony Algorithm for the 0–1 Multidimensional Knapsack Problem". In: Ranka, S., et al, Contemporary Computing, 94, 141–15, Springer Berlin Heidelberg, 2010.

[6] WEI, Y., "An improved binary artificial bee colony algorithm for solving multidimensional knapsack problem". Internation conference on Electroic Information Engineering and Computer Tecnology (EIECT), 1208719, 1208719, 2021.

[7] YAMASHITA, A., GARCIA, L. E., "Análise da aplicaçao do Algoritmo Colônia de Abelhas Artificiais no problema da Mochila Multidimensional Binária". Trabalho de Conclusão de Curso. Universidade Federal de Mato Grosso do Sul, Faculdade de Computação, 2023.

[8] ZHANG, S., LIU, S., "A Discrete Improved Artificial Bee Colony Algorithm for 0-1 Knapsack Problem". IEEE Access, 7, 104982-104991, 2019.