UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
BACHARELADO EM ENGENHARIA ELÉTRICA

LUCAS HENRIQUE BIUK

ALGORITMOS DE INTELIGÊNCIA DE ENXAME PARA OTIMIZAÇÃO BINÁRIA

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2019

LUCAS HENRIQUE BIUK

ALGORITMOS DE INTELIGÊNCIA DE ENXAME PARA OTIMIZAÇÃO BINÁRIA

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia Elétrica, do Departamento Acadêmico de Eletrônica, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Hugo Valadares Siqueira

PONTA GROSSA

2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Eletrônica
Bacharelado em Engenharia Elétrica

TERMO DE APROVAÇÃO

ALGORITMOS DE INTELIGÊNCIA DE ENXAME PARA OTIMIZAÇÃO BINÁRIA

por

LUCAS HENRIQUE BIUK

Este Trabalho de Conclusão de Curso foi apresentado em 06 de dezembro de 2019 como requisito parcial para a obtenção do título de Bacharel(a) em Engenharia Elétrica. O(A) candidato(a) foi arguido(a) pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof(a). Dr. Hugo Valadares Siqueira
Orientador(a)

Prof(a). Dr. Maurício dos Santos Kaster
Membro Titular

Prof(a). Dr. Eduardo Tadeu Bacalhau
Membro Titular

Prof. Dr. Josmar Ivanqui
Responsável pelos TCC

Prof. Dr. Sergio Okida
Coordenador do Curso

– O Termo de Aprovação assinado encontra-se na Coordenação do Curso –

Dedico este trabalho aos meus pais, que batalharam muito para que eu chegasse até aqui. Amo vocês.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por toda luz e força que me deu todos esses anos.

Aos meus pais, Carlos e Maria Isabel, por me ensinarem os valores que me tornam quem sou.

À minha irmã Jessica, pelo exemplo de perseverança e por todo apoio. Ao meu irmão Thiago, por todas as brigas e risadas.

À minha namorada Viviane, por todas séries, filmes, sonecas e por me acalmar nos momentos mais difíceis.

Ao meu amigo Jônatas, por me ouvir e me aconselhar desde sempre.

Aos meus amigos Eduardo, Felipe, Gabriel, Guilherme, Manoel e Miguel, por todos os momentos de descontração, e em especial, ao Walace, que esteve comigo (quase) todos os dias durante o curso.

Ao meu amigo Fernando Curi, pela ajuda no desenvolver dos algoritmos.

Ao professor Hugo, que além de meu orientador, foi um grande amigo que a vida me deu.

A todos meus amigos do LICON, por todos momentos juntos.

Enfim, a todos que de alguma forma contribuíram para a realização deste trabalho.

“Quando a vida decepciona, qual é a solução? Continue a nadar! Continue a nadar! Continue a nadar, nadar, nadar! Para achar a solução, nadar!” – (WALTERS, GRAHAM; **PROCURANDO NEMO**, 2003.)

RESUMO

BIUK, Lucas Henrique. **Algoritmos de Inteligência de Enxame para otimização binária**. 2019. 60 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica) – Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2019.

Este trabalho reúne uma diversidade de algoritmos bio-inspirados especializados em resolver problemas de otimização binária. O enfoque é dado aos algoritmos de inteligência de enxame, como a Otimização pro Enxame de Partículas (PSO) e a Busca por Cardume de Peixes (FSS), com o objetivo determinar quais as vantagens de cada um, comparando-os em desempenho quando aplicados para solução de problemas de natureza binária. Para tanto, estes são implementados no software MATLAB®, com intuito de facilitar a análise estatística dos dados obtidos através da simulação dos problemas com diversas dimensões, como o *One Max Problem* e o *Knapsack Problem*. Os resultados computacionais, parcialmente comparados com técnicas de computação evolutiva, revelam que o PSO é capaz de chegar aos melhores desempenhos gerais, seguidos da versão melhorada do FSS binário.

Palavras-chave: Inteligência de Enxame. Algoritmos Bio-Inspirados. Otimização Binária. *One Max Problem*. *Knapsack Problem*.

ABSTRACT

BIUK, Lucas Henrique. **Swarm Intelligence algorithms for binary optmization**. 2019. 60 p. Final Coursework (Bachelor's Degree in Eletrical Engineering) – Federal University of Technology – Paraná. Ponta Grossa, 2019.

This paper brings together a variety of bio-inspired algorithms specialized in solving binary optimization problems. The focus is on swarm intelligence algorithms, such as Particle Swarm Optimization (PSO) and Fish School Search (FSS), with the aim of determining the advantages of each one, comparing their performance for binary tasks. To this end, they are implemented in MATLAB ® software, in order to facilitate the statistical analysis of the results obtained by simulating problems with various dimensions, such as *One Max Problem* and *Knapsack Problem*. The computational results, partially compared with evolutionary computation techniques, reveal that the PSO is able to reach the best overall performances, followed by the improved version of the binary FSS.

Keywords: Swarm Intelligence. Bio-Inspired Algorithms. Binary Optmization. One Max Problem. Knapsack Problem.

LISTA DE ALGORITMOS

Algoritmo 1 – Vetor Instintivo 38
Algoritmo 2 – Novo Cálculo do Baricentro 38

LISTA DE ILUSTRAÇÕES

Figura 1 – Fluxograma de funcionamento do algoritmo PSO 21
Figura 2 – Função de Rastrigin no intervalo [-5,5] 21
Figura 3 – Função Sigmóide 24
Figura 4 – Função *V-Shaped* 25
Figura 5 – Movimento Coletivo Instintivo ilustrado antes (círculos cinza escuro) e depois (círculos cinza claro) de sua ocorrência. 30
Figura 6 – Fluxograma de funcionamento do algoritmo BFSS 35
Figura 7 – *Boxplot* do *Fitness* do *One Max Problem* com 100 dimensões 41
Figura 8 – *Boxplot* do *Fitness* do *One Max Problem* com 500 dimensões 43
Figura 9 – *Boxplot* do *Fitness* do *One Max Problem* com 1000 dimensões . . . 44
Figura 10 – *Boxplot* do *Fitness* do *One Max Problem* com 5000 dimensões . . . 45
Figura 11 – *Boxplot* do *Fitness* do *Knapsack Problem* com 30 dimensões 48
Figura 12 – *Boxplot* do *Fitness* do *Knapsack Problem* com 100 dimensões . . . 49
Figura 13 – *Boxplot* do *Fitness* do *Knapsack Problem* com 250 dimensões . . . 50
Figura 14 – *Boxplot* do *Fitness* do *Knapsack Problem* com 500 dimensões . . . 51

LISTA DE TABELAS

Tabela 1 – Tabela de Resultados *One Max Problem* com 100 dimensões 42
Tabela 2 – Tabela de Resultados *One Max Problem* com 500 dimensões 42
Tabela 3 – Tabela de Resultados *One Max Problem* com 1000 dimensões . . . 43
Tabela 4 – Tabela de Resultados *One Max Problem* com 5000 dimensões . . . 44
Tabela 5 – Tabela de Resultados *Knapsack Problem* com 30 dimensões 47
Tabela 6 – Tabela de Resultados *Knapsack Problem* com 100 dimensões . . . 48
Tabela 7 – Tabela de Resultados *Knapsack Problem* com 250 dimensões . . . 49
Tabela 8 – Tabela de Resultados *Knapsack Problem* com 500 dimensões . . . 50

SUMÁRIO

1	**INTRODUÇÃO**	**12**
1.1	OBJETIVOS	13
1.2	ESTRUTURA DO TRABALHO	14
2	**ALGORITMOS BIO-INSPIRADOS**	**15**
2.1	ALGORITMOS EVOLUTIVOS	16
2.2	INTELIGÊNCIA DE ENXAME	16
3	**ALGORITMOS DE INTELIGÊNCIA DE ENXAME - PSO E FSS**	**18**
3.1	OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS – PSO	18
3.2	OTIMIZAÇÃO BINÁRIA POR ENXAME DE PARTÍCULAS – BPSO	22
3.2.1	Inicialização da População Binária	22
3.2.2	Avaliação dos *Fitness* Iniciais	23
3.2.3	Ciclo Principal do Algoritmo - *Main Loop*	23
3.3	*FISH SCHOOL SEARCH* – FSS	26
3.3.1	Deslocamento Individual	28
3.3.2	Movimento Coletivo Instintivo	29
3.3.3	Movimento Coletivo Volitivo	30
3.3.4	Busca por Cardume de Peixes Binária – BFSS	31
3.3.5	Inicialização da População Binária	32
3.3.6	Deslocamento Individual	32
3.3.7	Movimento Coletivo Instintivo	33
3.3.8	Movimento Coletivo Volitivo	34
3.4	BFSS MELHORADO E SIMPLIFICADO	36
3.4.1	Mudanças para o BFSS Melhorado	36
3.4.2	Mudanças para o BFSS Simplificado	37
4	**EXPERIMENTAÇÃO E RESULTADOS**	**40**
4.1	*ONE MAX PROBLEM*	40
4.1.1	100 Dimensões ($D = 100$)	41
4.1.2	500 Dimensões ($D = 500$)	42
4.1.3	1000 Dimensões ($D = 1000$)	43
4.1.4	5000 Dimensões ($D = 5000$)	44
4.2	$0/1$ *KNAPSACK PROBLEM*	45
4.2.1	30 Dimensões ($D = 30$)	47
4.2.2	100 Dimensões ($D = 100$)	48
4.2.3	250 Dimensões ($D = 250$)	49
4.2.4	500 Dimensões ($D = 500$)	50
5	**CONCLUSÕES E PERSPECTIVAS**	**52**
	REFERÊNCIAS	**53**
	GLOSSÁRIO	**56**
	APÊNDICES	**56**
	APÊNDICE A – DADOS DOS VALORES DE VOLUME DA MOCHILA, ITENS E VALORES UTILIZADOS PARA AS SIMULAÇÕES DO *KNAPSACK PROBLEM*	**57**
A.1	DADOS PARA *KNAPSACK PROBLEM* COM DIMENSÃO $Dim = 30$	57
A.2	DADOS PARA *KNAPSACK PROBLEM* COM DIMENSÃO $Dim = 100$	57

A.3 DADOS PARA *KNAPSACK PROBLEM* COM DIMENSÃO $Dim = 250$ 57
A.4 DADOS PARA *KNAPSACK PROBLEM* COM DIMENSÃO $Dim = 500$ 58

1 INTRODUÇÃO

Os algoritmos de otimização bio-inspirados são métodos da área de Inteligência Computacional (IC). Diferentemente da Inteligência Artificial (IA), que possui foco em desenvolver modelos computacionais inspirados no raciocínio humano, essa área tem como princípio a observação da natureza para resolução de problemas (EBERHART; KENNEDY, 1995). Ao longo do tempo, diversos tipos de algoritmos foram desenvolvidos, tendo em vista a vastidão de comportamentos que podem ser observados na natureza, desde plantas e animais até sistemas imunológicos (CASTRO, 2006).

A Computação Natural, computação inspirada na natureza, vem como uma alternativa às soluções já existentes para determinados problemas, isto é, a sua aplicação é mais conveniente quando se possui um tarefas de alta complexidade, com grande quantidade de variáveis e possíveis soluções, uma vez que, se o problema não necessita desse tratamento, provavelmente possui uma solução específica. Sendo assim, tornou-se interessante entender melhor o funcionamento dos algoritmos deste tipo, para que num momento de aplicação, utilize-se corretamente o método que apresenta maiores vantagens para aquela situação (CASTRO, 2006).

A importância deste estudo é que, com o avanço acelerado dos processos de produção, avanço da Indústria 4.0, coleta e tratamento de dados com sistemas de armazenamento em nuvem, surgimento de novas máquinas cada vez mais velozes e complexas, dentre tantas outras tecnologias que são empregadas todos os dias nas fábricas, se torna imprescindível que muitos fatores e características sejam otimizados. Isto tem a finalidade de não gerar "gargalos no processo", e/ou de ganhar tempo de produção, tornando todo procedimento mais confiável e preciso, assim maximizando lucros, ou ao menos, minimizando dispêndios. Muitos desses processos podem ser otimizados com a utilização de meta-heurísticas, e por isso, deseja-se realizar uma comparação entre alguns algoritmos dessa área (MENEZES; FREITAS; PARPINELLI, 2016).

Seguindo esse intuito, os algoritmos bio-inspirados ganharam destaque nos últimos anos. O primeiro exemplo é o Algoritmo Genético (AG), que possui inspiração na teoria Darwiniana da Evolução das Espécies (DARWIN, 1859), utilizando opera-

dores de seleção, mutação e cruzamento (HOLLAND, 1975). Também se destaca a Otimização por Enxame de Partículas (*Particle Swarm Optmization* – PSO), que tem como base o comportamento grupal de espécies de animais, como enxame de abelhas (KENNEDY; EBERHART, 1997). Ainda, pode-se citar a Busca por Cardumes de Peixes (*Fish School Search* – FSS) inspirado no processo de alimentação dos referidos animais (FILHO et al., 2008). Um exemplo dentre os algoritmos de maior destaque para problemas específicos é a Otimização por Colônia de Formigas (*Ant Colony Optimization* – ACO), muito utilizado para resolução de tarefas de otimização inteira (STÜTZLE; DORIGO et al., 1999).

1.1 OBJETIVOS

O principal objetivo deste trabalho é realizar testes de desempenho entre algoritmos para otimização binária. Sendo assim, foram escolhidos os algoritmos *Binary Particle Swarm Optimization* (BPSO), *Binary Genetic Algorithm* (BGA), *Boolean Differential Evolution* (BDE) e *Binary Fish-School Search* (BFSS), pois são destaques da literatura e que oferecem bons parâmetros para comparação quando implementados para otimização de funções de *benchmark*. Dois problemas principais foram escolhidos como estudos de caso: *OneMax Problem* e 0/1 *Knapsack Problem*.

Para alcançar o objetivo principal são necessários alguns passos. Dessa forma pode-se listar como objetivos específicos:

- Implementar satisfatoriamente os algoritmos *Binary Fish-School Search* (BFSS), *Improved* BFSS, *Simplified* BFSS, BPSO, BGA e BDE para solução do *Knapsack Problem* e *One-Max Problem*;
- Realizar a simulação de cada um dos problemas propostos com diversas dimensões, aplicando os algoritmos para solucioná-los, e assim obtendo vasta gama de dados para comparação de resultados;
- Apresentar o desempenho de cada um dos algoritmos para cada dimensão escolhida a partir de análise estatística;
- Discutir melhorias que podem ser implementadas nos algoritmos, com a finalidade de proporcionar trabalhos futuros com melhor desempenho na sua utiliza-

ção.

1.2 ESTRUTURA DO TRABALHO

A estrutura desse trabalho parte da explicação sobre a inspiração biológica para algoritmos, com um breve histórico sobre o assunto e a diferenciação entre os Algoritmos Evolutivos e os de Inteligência de Enxame, sendo tal questão discutida no Capítulo 2. Em seguida, apresenta-se o Capítulo 3, mais enfático sobre algoritmos de Inteligência de Enxame, principalmente o *Fish-School Search*.

A partir daí, é apresentado o Capítulo 4 sobre os problemas que serão testados com a implementação dos algoritmos, juntamente com a análise dos resultados obtidos, e por fim, a conclusão no Capítulo 5.

2 ALGORITMOS BIO-INSPIRADOS

Os métodos bio-inspirados, como o próprio nome sugere, são algoritmos que utilizam a natureza como fonte de inspiração (ou metáfora) para o desenvolvimento de novas técnicas computacionais. Pertencem à área da Computação Natural, e são utilizados para resolver problemas complexos das mais diversas áreas da ciência, como Computação e Engenharia. Essa inspiração pode ocorrer de diversas formas, desde o ponto de vista comportamental dos animais, o modo como procuram alimento, defendem-se de predadores, migram de habitat, até o ponto de vista da genética, pelo modo como ocorrem mutação de cromossomos, a reprodução celular e a seleção natural. Assim, pode-se observar dois grupos de algoritmos: os de inteligência de enxame e os evolutivos (CASTRO, 2006).

Realiza-se então a emulação desses comportamentos ou evoluções por meio de uma linguagem de programação, com a finalidade de criar uma população de agentes de busca (partículas ou indivíduos), que percorrerão um determinado espaço de busca, simulando o comportamento utilizado como inspiração. Muitas vezes esse espaço de busca é a representação gráfica de uma função matemática, e os agentes buscam os valores de máximos ou mínimos nesse espaço, como desejado. Pode-se exemplificar tal tarefa supondo-se haver uma função matemática que represente o dispêndio de energia para realização de um trabalho. Neste caso, deseja-se encontrar o valor de mínimo da mesma. Entretanto, caso seja elaborada uma função que represente a eficiência de uma máquina, o valor de máximo é o que se procura. Pode-se ainda utilizar esses algoritmos para encontrar valores específicos, como na otimização de parâmetros de um controlador proporcional integral derivativo (PID), onde é necessário encontrar os valores dos ganhos K_p, K_i e K_d (PUCHTA et al., 2016), para realizar processos de seleção de variáveis (KIRA; RENDELL et al., 1992), para realizar simulação como apoio à tomada de decisão (SCHRAMM, 2009), dentre muitas outras aplicabilidades desses algoritmos.

2.1 ALGORITMOS EVOLUTIVOS

Os algoritmos evolutivos (AE) surgiram com pesquisas nas décadas de 60 e 70, como podem ser observados nos trabalhos de Rechenberg (1973), Fogel, Owens e Walsh (1966), Schwefel (1965) e Holland (1975). Estes possuem sua inspiração no princípio da evolução e seleção natural proposto por Charles Darwin no livro “A Origem das Espécies” (DARWIN, 1859). Sendo assim, é comum encontrar nos AEs: processos de seleção de indivíduos, por métodos como torneio ou roleta, simulando uma seleção natural, na qual o mais forte e adaptado tende a sobreviver; métodos de troca de informação genética (*crossover*), na qual os indivíduos geram novos agentes; e métodos de mutação, nos quais características dos indivíduos são mudadas aleatoriamente. Todos esses processos são introduzidos no algoritmo como uma forma de realizar o deslocamento dos agentes na função custo, assim performando uma varredura no espaço de busca por soluções (FILHO, 2018).

2.2 INTELIGÊNCIA DE ENXAME

Quando o termo Inteligência de Enxame (*Swarm Intelligence*) surgiu no início da década de 90, era usado para se referir a sistemas robóticos celulares, nos quais um conjunto de agentes interagiam sob um regime pré-determinado (BENI; WANG, 1993). Hoje este termo é utilizado para caracterizar todos os algoritmos que possuem inspiração biológica no comportamento coletivo de animais.

A inteligência coletiva é uma propriedade de sistemas compostos por agentes pouco inteligentes e com capacidade individual limitada, porém capazes de apresentar comportamentos coletivos complexos, levando a um alto grau de auto-organização (CASTRO, 2006).

Na natureza, pode-se observar várias espécies que apresentam comportamentos coletivos interessantes, podendo ser citadas as alcateias de lobos, os bandos de aves, colônias de insetos, cardumes de peixes e enxames de abelhas, dentre os quais alguns se destacam pela facilidade de serem interpretados e emulados (CASTRO, 2006).

Numa visão geral, percebe-se nos grupos de animais comportamentos complexos para realizar busca por alimento e proteção, ou seja, eles seguem seus instintos

de sobrevivência. Essas necessidades fazem com que os animais percorram a natureza de forma a sempre conseguir manter a continuidade de sua espécie, pois com as condições ideais de alimentação num habitat com poucos predadores, se reproduzem e mantêm um ciclo de vida. Porém, vê-se também que, quando um grupo se torna muito grande, o alimento se torna escasso, e então parte deste grupo parte em busca de uma novo local. O mesmo vale para o surgimento de predadores, que fazem com que necessitem fugir e se esconder. Do ponto de vista computacional, essas são observações de problemas que os animais precisam resolver para que se mantenham vivos. Observar a resolução desses problemas leva à inspiração dos algoritmos de inteligência de enxame (FILHO, 2016).

3 ALGORITMOS DE INTELIGÊNCIA DE ENXAME - PSO E FSS

Este capítulo irá apresentar os dois algoritmos de inteligência de enxame que foram escolhidos para estudo. Discutir-se-á os seus desenvolvimentos, a respectiva inspiração biológica e funcionamento. Além disso, será apresentada a forma como esses algoritmos são utilizados para otimização binária.

3.1 OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS – PSO

A Otimização por Enxame de Partículas (*Particle Swarm Optimization* – PSO) é um algoritmo inspirado na obervação ampla dos comportamentos coletivos de aves e abelhas, e foi proposto inicialmente por Eberhart e Kennedy (1995). Nele, utilizam-se apenas dois parâmetros para realização da busca: a posição em que a partícula se encontra e a velocidade com que ela se movimenta, sendo essas características atualizadas a cada ciclo do algoritmo. Essa atualização ocorre por meio de comunicação entre as partículas, avaliando qual dentre elas possui uma melhor solução para o problema, e uma avaliação da própria partícula com os resultados obtidos por ela anteriormente, ou seja, é realizada uma combinação entre as experiências individual e social das partículas (PUCHTA et al., 2016).

Para que a melhor solução seja encontrada, algumas limitações são atribuídas ao comportamento da partícula. A velocidade é controlada com o objetivo de não ocorrer grandes saltos no espaço de busca, ou que o deslocamento seja tão pequeno a ponto da partícula não alterar suas soluções ao decorrer das iterações, uma vez que a sua variação interfere diretamente na alocação da nova posição. Quando se trata da posição, a direção à qual a partícula vai seguir também é definida, fazendo com que todas se aproximem da melhor posição já encontrada até a iteração corrente. Para isso, cada uma armazena a lembrança da melhor posição por ela já visitada, chamada de *personal best* (*pbest*) e cada enxame possui aquela que alcançou a melhor posição entre todas, chamada *global best* (*gbest*). Para determinar o que é “melhor”, faz-se uma avaliação e dá-se uma “nota” para o desempenho do indivíduo, chamado de *fitness*. Este é o parâmetro que irá apontar qual partícula está mais próxima da melhor resposta possível até o momento (PUCHTA et al., 2016).

O processo para a atualização da velocidade e da posição está descrito matematicamente nas Equações (1) e (2), respectivamente (PUCHTA et al., 2016).

$$\mathbf{v}_{(t+1)} = \omega_t \cdot \mathbf{v}_t + rand_1 \cdot k_1 \cdot (\mathbf{pbest} - \mathbf{r}_t) + rand_2 \cdot k_2 \cdot (\mathbf{gbest} - \mathbf{r}_t) \tag{1}$$

sendo

- $\mathbf{v}_{(t+1)}$: velocidade futura da partícula;
- $\mathbf{v}_t$: velocidade atual da partícula;
- ω_t: constante de inércia;
- $\mathbf{r}_t$: posições de todas as partículas vindos do *looping* anterior;
- k_1: coeficiente cognitivo;
- k_2: coeficiente social;
- $rand_1$ e $rand_2$: coeficientes gerados aleatoriamente a partir de uma distribuição uniforme, no intervalo compreendido entre 0 e 1;
- $\mathbf{pbest}_p$: melhor posição da partícula sendo analisada (personal best);
- $\mathbf{gbest}_p$: melhor posição global dentre todas partículas (global best).

e

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_t \tag{2}$$

em que

- $\mathbf{x}_{t+1}$: futura posição da partícula;
- $\mathbf{x}_t$: posição atual da partícula;
- $\mathbf{v}_t$: velocidade da partícula.

O coeficiente de inércia ω foi uma mudança proposta por Shi e Eberhart (1998), parâmetro que não existia no algoritmo original. Assim, após estudos, determinou-se que alguns valores obtinham melhores resultados, pois a inércia influencia diretamente a variação das respostas. Então, os autores propuseram uma limitação na variável, estando ela no intervalo [$\omega_{max} = 0{,}9$; $\omega_{min} = 0{,}4$]. Além disso,

sugeriram um decremento linear deste parâmetro, o que melhorava significativamente a resposta final, uma vez que no início a partícula percorreria um espaço maior, e com o passar das iterações dá passos menores na área de busca, o que permite refinar a busca.

Uma sugestão para o decremento linear do valor do coeficiente de inércia é apresentado em função do número de iterações, como na Equação 3:

$$\omega_{t+1} = \omega_{max} - \frac{(\omega_{max} - \omega_{min})}{(it_{max})} \cdot it; \tag{3}$$

em que:

- ω_{t+1} coeficiente de inércia para a próxima iteração;
- ω_{max} é o coeficiente de inércia máximo;
- ω_{min} é o coeficiente de inércia mínimo;
- it_{max} e it são o número máximo de iterações determinado e a iteração atual, respectivamente.

No mesmo trabalho, encontra-se a sugestão da utilização para os coeficientes cognitivo e social iguais a dois, fazendo com que o espaço de busca se torne cada vez mais reduzido, facilitando que o enxame encontre a melhor posição global (PUCHTA et al., 2016). A figura 1 apresenta o funcionamento do PSO.

Muitas funções são utilizadas para teste dos algoritmos de otimização. Aqui será dado destaque para a Função de Rastrigin [1], pois esta apresenta muitos picos e vales, ou seja, muitos pontos de máximo e mínimo locais (ÁVILA et al., 2002). Uma representação gráfica dessa função pode ser observada na Figura 2, onde os pontos vermelhos representam agentes de busca distribuídos aleatoriamente em sua superfície. A figura apresentada foi gerada a partir da Equação (4):

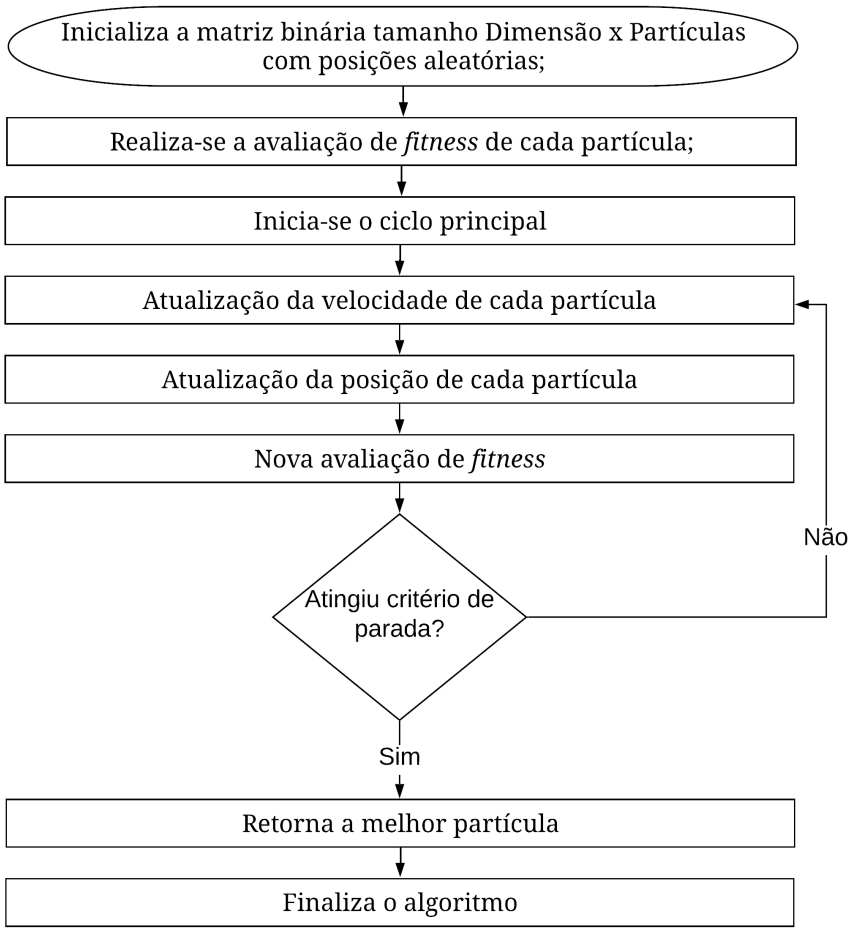
$$z = f(x,y) = 20 + x^2 + y^2 - 10 \cdot (\cos(2 \cdot \pi \cdot x) + \cos(2 \cdot \pi \cdot y)) \tag{4}$$

em que x e y são coordenadas das partículas no espaço (posição).

Para essa função, deseja-se encontrar o valor de mínimo, e se conhece o mínimo global da função, o qual se encontra em $x = y = 0$, resultando em $z = 0$. Dessa

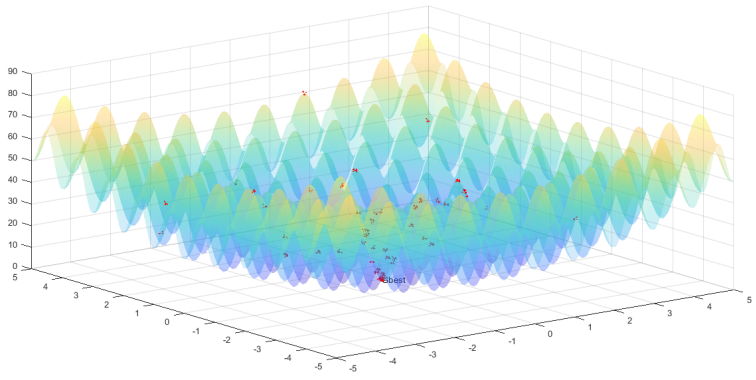
[1] Função de Rastrigin - função de teste utilizada em problemas de teste de otimização. Possui vários mínimos locais que são distribuídos regularmente e é altamente multimodal.

Figura 1 – Fluxograma de funcionamento do algoritmo PSO



Fonte: Autoria Própria

Figura 2 – Função de Rastrigin no intervalo [-5,5]



Fonte: Autoria Própria

forma, uma função de *fitness* que pode ser empregada é apresentada na Equação 5:

$$fit(z) = \frac{1}{1+z} \tag{5}$$

sendo fit a função que calcula o *fitness*. Quanto maior o *fitness*, melhor a solução, ou

seja, mais próximo do valor ótimo da função, que está em uma posição onde $z = 0$.

Dos processos descritos no fluxograma, vale ressaltar que a inicialização aleatória das partículas é feita restritamente dentro do espaço de busca, limitando o espaço de busca dentro da região factível.

3.2 OTIMIZAÇÃO BINÁRIA POR ENXAME DE PARTÍCULAS – BPSO

O algoritmo de Otimização Binária por Enxame de Partículas (*Binary Particle Swarm Optmization* – BPSO) foi proposto por Kennedy e Eberhart (1997) e é utilizado para realizar otimizações no domínio binário. No BPSO, os agentes são vetores binários, em que a quantidade de uns e zeros vai determinar se a solução encontrada para o problema proposto é satisfatória. As propriedades do agente permanecem as mesmas, ou seja, ele possui uma velocidade e uma posição, e o seu deslocamento depende da mesma equação da velocidade, Equação (1). Porém, é realizada a transformação dos resultados para o domínio binário. Para ilustrar melhor cada procedimento e o funcionamento do algoritmo binário cada passo será descrito separadamente.

3.2.1 Inicialização da População Binária

A população de partículas binárias é inicializada aleatoriamente como um conjunto de vetores binários ou como uma matriz binária (possuem somente zeros e uns). No primeiro caso, cada vetor é uma partícula e cada elemento é chamado de dimensão (bit), sendo que cada dimensão vai representar uma característica a ser avaliada. O segundo caso se diferencia do primeiro apenas pelo modo como é armazenado, pois cada linha da matriz é uma partícula e cada coluna é um bit. Exemplificando: se é desejado inicializar uma população de N partículas, cada uma terá D dimensões binárias. Dado que R é uma matriz binária, para cada dimensão D em cada partícula N, sorteia-se um bit zero ou um, gerando a matriz R de forma randômica.

O resultado esperado para esse processo é encontrado na matriz $R_{N,D}$:

$$R_{N,D} = \begin{bmatrix} 0 & 1 & 0 & \dots & R_{1,(D-1)} & R_{1,D} \\ 0 & 1 & 1 & \dots & R_{2,(D-1)} & R_{2,D} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ R_{N,1} & R_{N,2} & R_{N,3} & \dots & R_{N,(D-1)} & R_{N,D} \end{bmatrix},$$

tal que todos os valores de R são bits (0,1).

3.2.2 Avaliação dos *Fitness* Iniciais

Após a inicialização da população de agentes binários, deve-se realizar a avaliação inicial de cada um para dar início aos procedimentos de busca, sendo calculado o seu *fitness*. A forma como vai ser feita tal avaliação depende do problema proposto. No Capítulo 4 será apresentada a forma como foram feitas as análises de *fitness* para o *One Max Problem* e *Knapsack Problem*.

3.2.3 Ciclo Principal do Algoritmo - *Main Loop*

Antes de começar o ciclo principal, vale lembrar que é necessário que se declare as constantes que foram definidas anteriormente, como os coeficientes cognitivo e social, coeficiente de inércia máximo e mínimo, variáveis auxiliares, como a quantidade máxima de iterações do algoritmo (veja a Seção 3.1).

Os procedimentos que seguem são basicamente os mesmos apresentados no fluxograma da Figura 1, porém é necessário que a velocidade e a posição sejam binárias. Existem duas formas de realizar esse processo: a primeira e mais simples, utiliza funções de transferência e mantém os demais parâmetros como no PSO clássico. Aqui serão apresentados os exemplos das funções Sigmóide (*S-Shaped*) e *V-Shaped*, que podem ser observadas no trabalho de Siqueira et al. (2018).

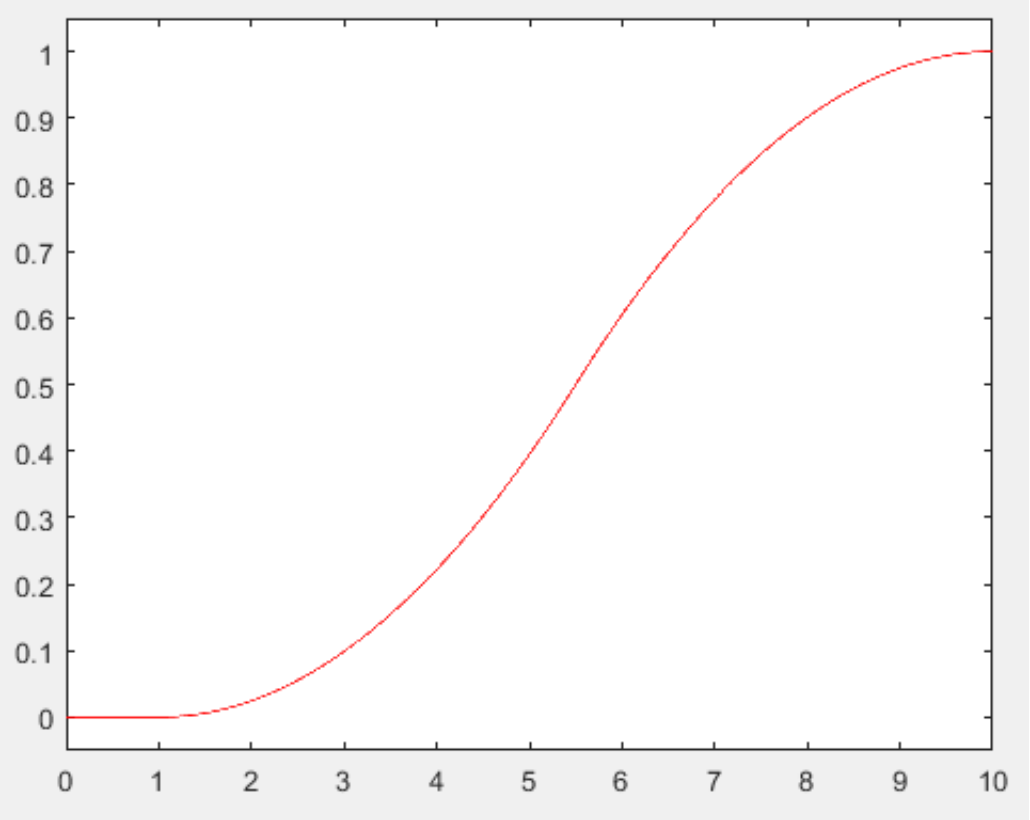
A função de transferência *S-Shaped* possui, como o próprio nome sugere, formato semelhante à letra S. É gerada a partir da equação (6).

$$S(\mathbf{v_p}) = \frac{1}{1 + e^{-\beta \cdot \mathbf{v_p}}} \tag{6}$$

onde β é um coeficiente que define a inclinação da curva, ou seja, se será mais acentuada ou mais suavizada, neste trabalho foi utilizado $\beta = 1$, e $\mathbf{v}_p$ é a velocidade da partícula, que deve definir os limites laterais, sendo o esquerdo igual a zero e o di-

reito a velocidade máxima da partícula. Na Figura 3 é mostrado um exemplo visual, no intervalo de 0 a 10, de como a curva se comporta:

Figura 3 – Função Sigmóide



Fonte: Autoria própria.

No trabalho de Siqueira et al. (2018) há duas propostas para o cálculo da posição binária utilizando o resultado da função Sigmóide, descritas nas equações 7 e 8:

$$x_{p,D}^{t+1} = \begin{cases} gbest_D^t & \text{se } rand < S(\mathbf{v}_p) \\ x_{p,D}^t & \text{, caso contrário.} \end{cases} \tag{7}$$

ou

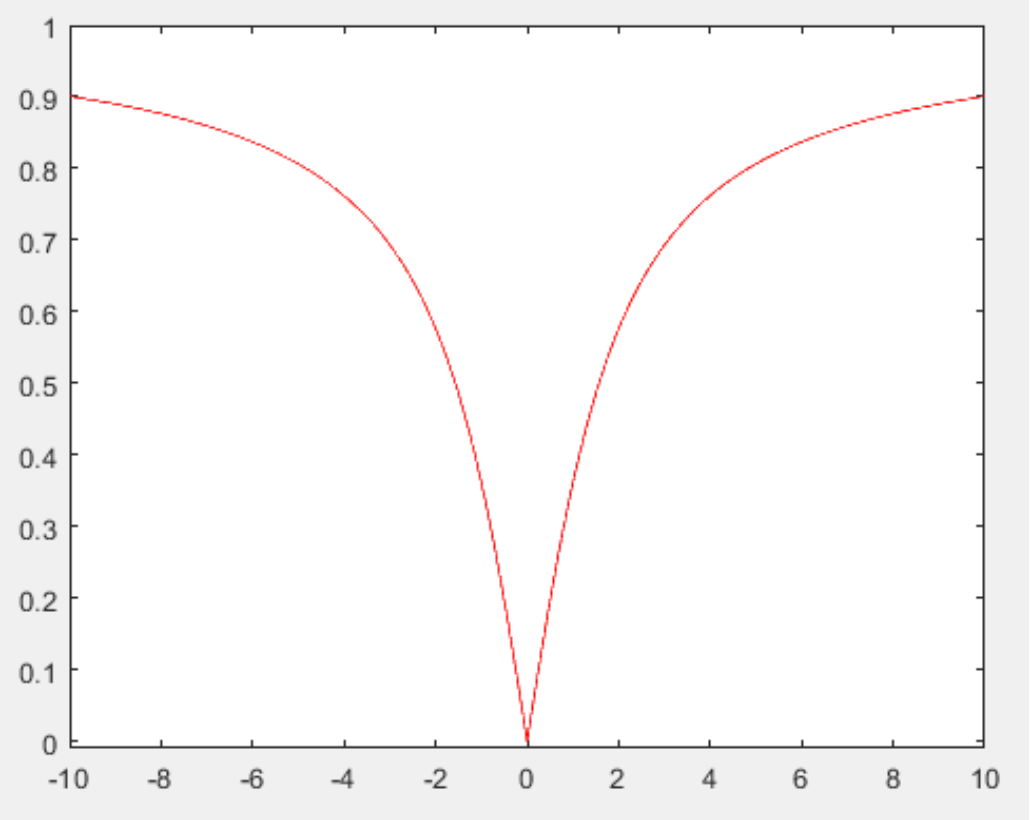
$$x_{p,D}^{t+1} = \begin{cases} \overline{x_{p,D}^t} & \text{se } rand < S(\mathbf{v}_p) \\ x_{p,D}^t & \text{, caso contrário.} \end{cases} \tag{8}$$

- $x_{p,D}^{t+1}$ é o bit da dimensão D, pertencente à partícula p, na futura iteração $t+1$;
- $x_{p,D}^{t}$ é o bit da dimensão D, pertencente à partícula p, na atual iteração t;
- $\overline{x_{p,D}^t}$ é operação lógica *not* sobre o bit;
- $gbest_D^t$ é o valor do bit na dimensão D pertencente ao $\mathbf{gbest}$;

- $rand$ valor aleatório gerado a partir de uma distribuição uniforme, no intervalo de 0 a 1;
- $S(\mathbf{v}_p)$ é o resultado da função sigmóide.

A função *V-Shaped*, como pode ser vista na Figura 4, possui formato semelhante à letra V, e segue os mesmos princípios de função de transferência que a função sigmóide.

Figura 4 – Função *V-Shaped*



Fonte: Autoria Própria.

Assim, pode-se realizar a atualização da posição utilizando a Equação (7) ou (8), apenas alterando a função S para a função V, gerada pela equação (9). Com esse procedimento, os demais processos permanecem iguais aos descritos.

$$V(\mathbf{v}_p) = \left| \frac{2}{\pi} \cdot \arctan\left(\frac{2}{\pi} \cdot \mathbf{v}_{p,d} \right) \right| \tag{9}$$

A segunda forma de tratar o algoritmo binário é fazer uso de uma variável auxiliar, chamada velocidade intermediária, sem utilizar funções de transferência. Para que isso seja possível, é necessário calcular tal variável por meio da Equação (14). Seguindo o descritivo presente no trabalho de Siqueira et al. (2018), define-se variáveis

auxiliares $d_{p,D}^{bit,pbest}$ e $d_{p,D}^{bit,gbest}$:

$$d_{p,D}^{1,pbest} \text{ e } d_{p,D}^{0,pbest} = \begin{cases} r_1 k_1 \text{ e } -r_1 k_1 & \text{se } pbest_D = 1 \\ -r_1 k_1 \text{ e } r_1 k_1 & \text{se } pbest_D = 0 \end{cases} \tag{10}$$

$$d_{p,D}^{1,gbest} \text{ e } d_{p,D}^{0,gbest} = \begin{cases} r_2 k_2 \text{ e } -r_2 k_2 & \text{se } gbest_D = 1 \\ -r_2 k_2 \text{ e } r_2 k_2 & \text{se } gbest_D = 0 \end{cases} \tag{11}$$

Logo, as velocidades são calculadas com as equações (12) e (13):

$$v_{p,d}^0 = \omega v_{p,d}^0 + d_{p,D}^{0,pbest} + d_{p,D}^{0,gbest} \tag{12}$$

e

$$v_{p,d}^1 = \omega v_{p,d}^1 + d_{p,D}^{1,pbest} + d_{p,D}^{1,gbest} \tag{13}$$

Então, tem-se v' como na equação (14):

$$v'_{p,D} = \begin{cases} v_{p,d}^1 & \text{se } x_{p,D} = 0 \\ v_{p,d}^0 & \text{se } x_{p,D} = 1 \end{cases} \tag{14}$$

Por fim, o deslocamento é realizado com a Equação (15)

$$x_{p,D}^{t+1} = \begin{cases} \overline{x_{p,D}^t} & \text{se } rand > v'_{p,D} \\ x_{p,D}^t & \text{ao contrário.} \end{cases} \tag{15}$$

Como os demais algoritmos utilizados para comparação serão puramente binários, uma vez que os problemas de *benchmark* que serão usados para os testes são inicializados de forma binária, optou-se por utilizar a segunda forma apresentada, sem utilização de funções de transferência.

3.3 *FISH SCHOOL SEARCH* – FSS

O algoritmo de Busca por Cardume de Peixes (*Fish School Search* – FSS) é inspirado no comportamento alimentar de cardumes. Foi apresentado em 2008, como pode ser visto nos trabalhos de Filho et al. (2008), e desde então tem se mostrado um algoritmo muito eficiente, além de ter sofrido diversas melhorias nos últimos anos, gerando variações de sua versão original (SANTANA et al., 2019). Isso se dá principalmente por conta de nuances que podem ser observadas no comportamento dos

variados cardumes, pois existem milhares de espécies de peixes cada uma apresenta suas especificidades. Outros trabalhos usam premissas parecidas como em Luo, Chen e Guo (2005) e Hersovici et al. (1998).

O FSS apresenta na literatura bons resultados para resoluções de problemas de otimização multimodal, pois possui funcionalidade para evitar mínimos locais a partir do movimento de expansão ou contração do cardume (SANTANA et al., 2019). Além disso, a movimentação no algoritmo de FSS é mais complexa que a observada no PSO, já que outros parâmetros são levados em consideração. No PSO, somente a velocidade e a posição da partícula são usados para os cálculos. No FSS, a relação entre cada peixe com o cardume é mais profunda: os peixes tendem a interagir mais fortemente uns com os outros, aumentando ou reduzindo a dispersão, deslocando-se individualmente, mas também respeitando um movimento coletivo.

Outro parâmetro importante neste algoritmo é o peso do peixe. Como a inspiração biológica baseia-se em como este animal busca alimento, o valor de *fitness* é avaliado a partir do ganho ou perda de peso, uma vez que se ele engordou significa que encontrou mais alimento, e se emagreceu ele se distanciou do ponto com mais alimento.

Dada a apresentação acima, podemos sumarizar os principais parâmetros do FSS da seguinte forma:

- Posição individual: assim como no PSO, a posição do peixe é a característica que indicará se a resposta para o problema proposto foi encontrada, isto é, que ele encontrou a melhor posição possível dentro do espaço de busca;

- Peso (*weight*): o peso do peixe é um valor numérico, previamente definido, atribuído a todos os peixes no início do algoritmo, e será atualizado com o passar das iterações a partir da variação do seu valor de *fitness*, no processo chamado *Feeding Operator* (Operador de Alimentação);

- Posição coletiva ou baricentro: seguindo a inspiração biológica sobre o cardume, tem-se a posição coletiva ou baricentro, que será determinada a partir das posições individuais de cada peixe e seu respectivo peso, de forma a permitir a expansão ou contração do cardume tendo este parâmetro como referência;

- Passo (*step*): assim como no PSO em que há o valor da inércia influenciando o quanto a partícula pode se locomover, no FSS existe o valor de passo, que segue o mesmo princípio de não permitir saltos no espaço de busca. Serão definidos dois valores de *step* no algoritmo, o individual e o coletivo, e ambos sofrem decremento linear com o passar das iterações.

Vale mencionar que, como se trata de um algoritmo de busca em uma região limitada e utiliza cardumes de peixe como inspiração, o espaço de busca neste algoritmo é definida como um aquário.

A partir daí, pode-se definir todas as movimentações que o cardume pode efetuar. No PSO, cada agente se movia tomando por referência o melhor indivíduo do grupo, usando somente a equação da velocidade para realizar o deslocamento. Já no FSS, existem algumas imposições a mais, sendo elas: Deslocamento Individual, Movimento Coletivo Instintivo e Movimento Coletivo Volitivo.

3.3.1 Deslocamento Individual

O deslocamento individual é o primeiro efetuado no algoritmo. Realizado por todos os peixes no início de todas as iterações, depende da posição atual do peixe e do valor de *Individual Step* previamente definido. Para que seja adicionada fortuidade no processo de busca, o valor de *step* é sempre multiplicado por um valor aleatório gerado a partir de uma distribuição uniforme entre 0 e 1. A posição dos peixes após o deslocamento individual é tal que não extrapole as bordas do aquário (FILHO et al., 2008). A Equação 16 mostra como esse processo é realizado matematicamente:

$$\mathbf{x}_p^{t+1} = \mathbf{x}_p^t + (step_{ind} \cdot rand) \tag{16}$$

tal que:

- $\mathbf{x}_p^{t+1}$ é a posição futura do peixe p, pós deslocamento;
- $\mathbf{x}_p^t$ é a posição atual do peixe;
- $step_{ind}$ é o valor de passo individual;

Logo após o deslocamento individual se realiza a operação de alimentação do cardume. O operador de alimentação nada mais é que uma forma matemática

de recompensar os peixes que obtiveram sucesso na procura. É a partir do peso W que, no movimento coletivo volitivo, será avaliado o baricentro do cardume. A sua atualização ocorre de acordo conforme a Equação (17):

$$W_{t+1}^{p} = W_{t}^{p} + \frac{\Delta f_p}{max(|\Delta f|)} \tag{17}$$

na qual:

- W_{t+1}^{p} é o futuro peso do peixe p, pós alimentação;
- W_{t}^{p} é o peso atual do peixe;
- Δf_p é a variação de *fitness* de cada peixe da iteração anterior para a atual, como na Equação (18);
- $max(|\Delta f|)$ é o valor da maior variação de *fitness* dentre todos os peixes, aplicando-se o comando max sobre o resultado da Equação (18).

$$\Delta f_p = f_p^t - f_p^{t-1} \tag{18}$$

3.3.2 Movimento Coletivo Instintivo

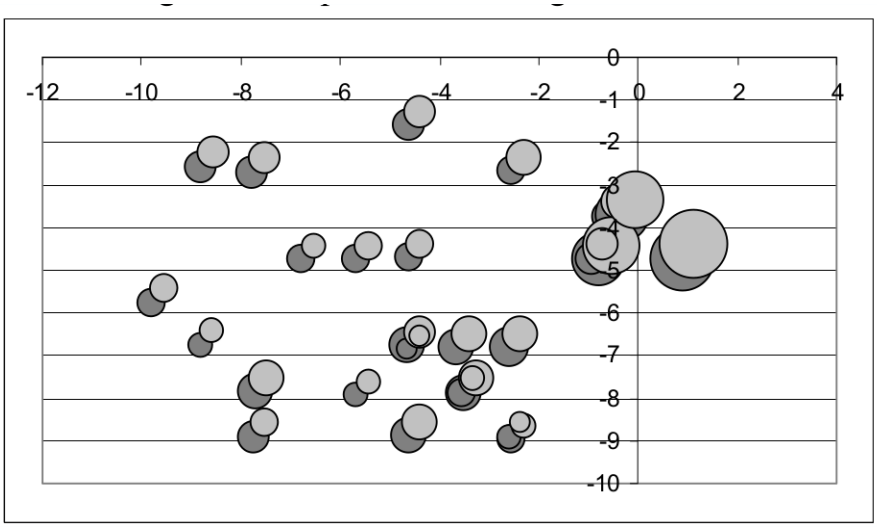
O segundo deslocamento realizado é o movimento coletivo instintivo, que visa atrair o cardume para próximo aos peixes que obtiveram os melhores resultados após o deslocamento individual. Esse processo de movimentação utiliza uma média ponderada do deslocamento individual pela variação de *fitness* apresentada pelo peixe, fazendo com que o cardume se desloque em direção àqueles que apresentaram melhor resposta. A Figura 5 mostra de como ocorre o deslocamento dos peixes:

Na Figura 5 pode-se observar que a direção seguida pela maioria dos peixes foi a daqueles que possuíram o melhor resultado na busca de alimentos, ou seja, aqueles com maiores pesos após o deslocamento individual. O peso é representado pelo diâmetro dos círculos, de modo que quanto maiores os diâmetros, maior o peso do peixe. Na Equação (19) é apresentado como ocorre este movimento:

$$\mathbf{x}_p^{t+1} = \mathbf{x}_p^t + \frac{\sum_{p=1}^{N}(\Delta \mathbf{x}_p^{ind} \cdot \Delta f_p)}{\sum_{p=1}^{N} \Delta f_p} \tag{19}$$

sendo:

Figura 5 – Movimento Coletivo Instintivo ilustrado antes (círculos cinza escuro) e depois (círculos cinza claro) de sua ocorrência.



Fonte: Retirado de Filho et al. (2008)

- $\mathbf{x}_p^{t+1}$ é a posição futura do peixe p, pós movimento;
- $\mathbf{x}_p^t$ é a posição atual do peixe;
- p é o índice para cada peixe;
- N é o total de peixes no cardume;
- $\Delta\mathbf{x}_p^{ind}$ é o deslocamento individual performado pelo peixe;
- Δf_p é a variação de *fitness* do peixe entre a iteração anterior e a atual.

3.3.3 Movimento Coletivo Volitivo

O movimento coletivo volitivo é o último performado. Volição é o processo cognitivo pelo qual um indivíduo se decide a praticar uma ação em particular. É definida como um esforço deliberado, sendo uma das principais funções psicológicas humanas. Processos volitivos podem ser aplicados conscientemente e podem ser automatizados como hábitos no decorrer do tempo (CORNO; KANFER, 1993).

Esse movimento é baseado na avaliação de desempenho do cardume como um todo, a partir da variação de peso apresentada pelos outros deslocamentos. É este que vai proporcionar a expansão ou contração do cardume e ocorre de maneira simples: se o peso do cardume (soma do peso dos peixes) aumentou, significa que que o cardume está se aproximando do melhor resultado. Neste caso, a distância entre

os peixes diminui (se aproximam do baricentro do cardume), chamado processo de contração. Caso contrário, se o peso do cardume diminuiu, significa que os peixes se afastaram do melhor resultado. Sendo assim, o cardume se expande com o objetivo de aumentar a área de procura. Além do aquário, o processo de expansão ou contração conta com mais uma limitação que é o *Volitive Step*, que segue as mesmas funções do *Individual Step* e coeficiente de inércia. O processo de cálculo do baricentro e do movimento volitivo está descrito respectivamente nas Equações (20), (21) (contração do cardume) e (22) (expansão do cardume).

$$\mathbf{Bari}_{(t)} = \frac{\sum_{p=1}^{N}(\mathbf{x}_p^t \cdot W_p^t)}{\sum_{p=1}^{N} W_p^t} \tag{20}$$

De forma que:

- $\mathbf{Bari}_{(t)}$ é o baricentro do cardume;
- $\mathbf{x}_p^t$ é a posição atual de cada peixe;
- W_p^t é o peso de cada peixe;

e

$$\mathbf{x}_p^{t+1} = \mathbf{x}_p^t - step_{vol} \cdot rand \cdot [\mathbf{x}_p^t - \mathbf{Bari}_{(t)}] \tag{21}$$

$$\mathbf{x}_p^{t+1} = \mathbf{x}_p^t + step_{vol} \cdot rand \cdot [\mathbf{x}_p^t - \mathbf{Bari}_{(t)}] \tag{22}$$

- $\mathbf{x}_p^{t+1}$ é a posição futura do peixe p, pós deslocamento;
- $step_{vol}$ é o valor de passo volitivo (*volitive step*);

A atualização para os valores de $step$ ocorrem da mesma forma que na Equação (3).

3.3.4 Busca por Cardume de Peixes Binária – BFSS

O algoritmo de Busca por Cardume de Peixes Binária (*Binary Fish School Search* – BFSS) foi proposto por Sargo et al. (2014) e trouxe ajustes em relação ao FSS original para otimização no domínio binário. Este novo método realiza passos semelhantes aos já descritos, porém adaptados ao problema binário.

3.3.5 Inicialização da População Binária

A inicialização da população binária segue os mesmos passos descritos na Seção 3.2.1. É gerada uma matriz de zeros e uns em que cada linha da é um peixe (agente de busca) e cada coluna é uma dimensão (*bit* 0 ou 1).

3.3.6 Deslocamento Individual

Uma adaptação foi realizada no deslocamento individual para tratar vetores binários. Basicamente, o valor de $step_{ind}$, que anteriormente era utilizado para controlar o deslocamento no espaço de busca, será um parâmetro que determina a probabilidade das dimensões de cada peixe terem seu valor alterado (*flip bit*). O deslocamento é na verdade a mudança do conjunto de dimensões selecionadas como prováveis soluções ao problema binário proposto. O deslocamento individual binário está descrito na Equação (23):

$$x_{p,D}^{t+1} = \begin{cases} \overline{(x_{p,D}^{t})} & \text{, se } rand(0,1) < step_{ind}^{t}. \\ x_{p,D}^{t} & \text{, caso contrário.} \end{cases} \tag{23}$$

sendo:

- $x_{p,D}^{t+1}$ o *bit* da dimensão D, pertencente ao peixe p, na futura iteração $t+1$;
- $x_{p,D}^{t}$ o *bit* da dimensão D, pertencente ao peixe p, na atual iteração t;
- $\overline{(x_{p,D}^{t})}$ o *bit* negado (operação lógica *not*) da dimensão D, pertencente ao peixe p, na atual iteração t;
- $rand$ valor aleatório gerado a partir de uma distribuição uniforme, no intervalo de 0 a 1;
- $step_{ind}^{t}$ é o valor do passo individual para a iteração t.

Assim como na versão original, logo após o deslocamento individual é realizada a operação de alimentação. Na versão binária, este processo não sofreu alterações, logo, basta efetuar o processo descrito na Equação (17).

3.3.7 Movimento Coletivo Instintivo

No BFSS, cria-se primeiramente um vetor M que calcula a relação entre a posição e o *fitness* do peixe a partir da Equação (24) (SANTANA et al., 2019):

$$M_{(D)} = \frac{\sum_{p=1}^{N}(x_{p,D}^{t} \cdot \Delta f_p)}{\sum_{p=1}^{N} \Delta f_p} \tag{24}$$

Como pode ser observado, o valor de Δx_p^{ind} foi substituído por $x_{p,D}^{t}$. Onde antes era obtido o valor do quanto o peixe se movimentou para cálculo do novo deslocamento, agora há um vetor com valores no intervalo [0,1], de tamanho igual ao número de peixes. Como o propósito do movimento coletivo instintivo é levar o cardume em direção aos agentes que obtiveram melhores resultados no deslocamento individual, é necessária a conversão do vetor que possui valores no dado intervalo para zeros e uns.

Aqui se usa um novo operador $tresh_c$ (*collective treshold*), um limitador para que se tenha um parâmetro de conversão do vetor M em um vetor binário. De acordo com Sargo et al. (2014), $tresh_c$ será multiplicado pelo maior valor de M, isto é, com o valor de maior relação entre posição e *fitness*, com o objetivo de que pelo menos uma dimensão seja sempre escolhida para manter a habilidade de busca deste deslocamento. A Equação (25) mostra como este processo é realizado:

$$M_{(D)}^{bit} = \begin{cases} 0 & \text{se } M_{(D)} < tresh_c \cdot max(M_{(D)}) \\ 1 & \text{se contrário.} \end{cases} \tag{25}$$

em que:

- $M_{(D)}^{bit}$ é o vetor M binário;
- $max(M_{(D)})$ é a função que retorna o maior valor do vetor M.

Agora, possuindo o vetor M binário, pode-se realizar o deslocamento, que ocorre da seguinte forma: escolhe-se apenas uma dimensão (um *bit*) de cada peixe, de forma aleatória, tal que este seja diferente do *bit* dessa mesma dimensão em M^{bit}. Isto é, compara-se cada *bit* de $x_{p,D}$ com $M_{(D)}^{bit}$ e sorteia-se um que apresente $x_{p,D} \neq M_{(D)}^{bit}$. Este *bit* sorteado em $x_{p,D}$ receberá o valor de M^{bit}, ou seja, isto faz com que o agente se mova em direção à posição com relação posição-*fitness* mais forte, como

na Equação 26:

$$x_{p,D_{\text{raffled}}}^{t+1} = M_{D_{\text{raffled}}}^{bit} \tag{26}$$

sendo que "D_{raffled}" é a dimensão sorteada na condição dada.

3.3.8 Movimento Coletivo Volitivo

Assim como os deslocamentos anteriores, o movimento coletivo volitivo também necessita de adaptações para o seu desenvolvimento binário. Como seu objetivo é a contração ou expansão do cardume, isto é, aproximar ou distanciar os agentes do baricentro, torna-se necessário que esse seja também binário.

O processo realizado para esta conversão é similar à forma como ocorreu o cálculo do vetor M no movimento anterior. Primeiro, aplica-se a Equação (20) e se obtém $Bari_D$, vetor de baricentro. Seguindo o mesmo processo de conversão apresentado anteriormente, tem-se a Equação (27):

$$\mathbf{Bari}_{(D)}^{bit} = \begin{cases} 0 & \text{se } \mathbf{Bari}_{(D)} < tresh_v \cdot max(\mathbf{Bari}_{(D)}) \\ 1 & \text{ao contrário.} \end{cases} \tag{27}$$

sendo que:

- $\mathbf{Bari}_{(D)}^{bit}$ é relativo ao vetor $\mathbf{Bari}$ binário na dimensão D;
- $max(\mathbf{Bari}_{(D)})$ é a função que retorna o maior valor do vetor $\mathbf{Bari}$;
- $tresh_v$ é o *volitive treshold*, limitador de conversão.

Com o intuito de facilitar o processo de aproximação ou expansão, pode-se criar um vetor anti-baricentro, que será exatamente o vetor de baricentro invertido, como segue na Equação (28):

$$\mathbf{AntiBari}_D^{bit} = \overline{\mathbf{Bari}_D^{bit}}; \tag{28}$$

Assim, ao aproximar-se do anti-baricentro, o agente está na verdade se afastando do baricentro.

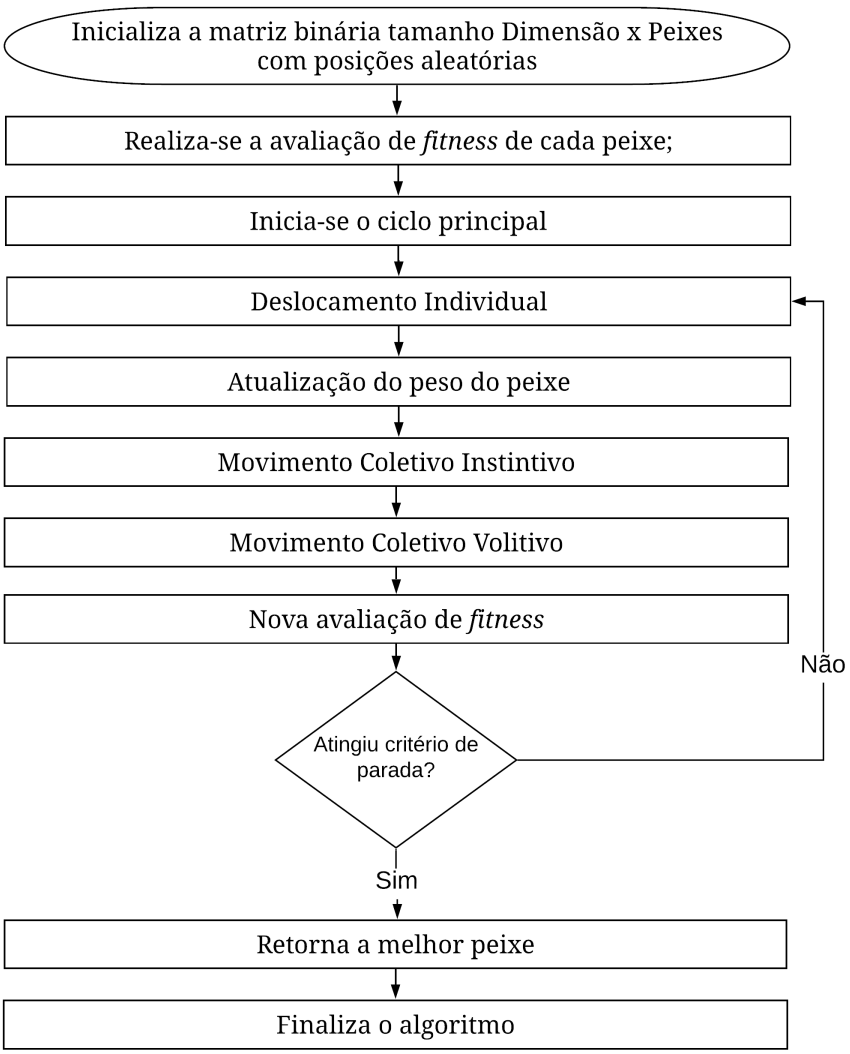
A partir daí, basta seguir os mesmos procedimentos apresentados para o deslocamento instintivo, isto é, compara-se cada *bit* $\mathbf{x}_{p,D}$ com $\mathbf{Bari}_{(D)}^{bit}$ e sorteia-se um em que $\mathbf{x}_{p,D} \neq \mathbf{Bari}_{(D)}^{bit}$. A aproximação ao baricentro se dá caso o cardume obtenha

sucesso, isto é, se o peso do cardume aumentou. Caso contrário, se o cardume perdeu peso, este irá se aproximar do anti-baricentro. Este processo fica mais visível na Equação (29):

$$\mathbf{x}_{p,(D^{\text{raffled}})}^{t+1} = \begin{cases} \mathbf{Bari}_{D^{\text{raffled}}}^{t} & \text{se o peso aumentou;} \\ \mathbf{AntiBari}_{D^{\text{raffled}}}^{t} & \text{se o peso diminuiu.} \end{cases} \tag{29}$$

Em ambos os movimentos coletivos, o motivo pelo qual escolhe-se inverter apenas uma dimensão por iteração é propiciar a suavidade da convergência, de forma que os vetores não evitem posições que poderiam ser consideradas como bons resultados. De forma a deixar mais visível o funcionamento do BFSS, a Figura 6 apresenta

Figura 6 – Fluxograma de funcionamento do algoritmo BFSS



Fonte: Autoria própria.

um fluxograma com os passo-a-passo descritos.

3.4 BFSS MELHORADO E SIMPLIFICADO

Nesta seção serão apresentadas duas variações do BFSS, a proposta de sua versão melhorada – *Improved Binary Fish School Search* - IBFSS, e a simplificada – *Simplified Binary Fish School Search* - SBFSS. A partir de pequenas mudanças no código, pode-se realizar mudanças significativas no desempenho do algoritmo. Como todos os processos permanecem muito similares ao apresentado na seção 3.3.4, apenas as principais alterações serão mostradas.

3.4.1 Mudanças para o BFSS Melhorado

O algoritmo de busca *Improved* BFSS foi proposto por Carneiro e Bastos-Filho (2016) e trouxe algumas alterações interessantes ao algoritmo. Por exemplo, não se utiliza valores de limitação – *tresholds*, acelerando os processos dos deslocamentos coletivos.

Seguindo o passo-a-passo do algoritmo, inicia-se pela Inicialização da População. A primeira sugestão de mudança foi que a população inicial fosse de vetores binários com dimensões majoritariamente iguais a zero (0). A premissa é que, quando aplicados a problemas de seleção de variáveis, a capacidade de exploração no aquário (espaço limitado para o cardume) é melhorada quando começando com uma quantidade menor de itens selecionados. Dado isso, a inicialização da população se deu com apenas 25% de chance da dimensão ser igual a 1 (um), como na Equação (30):

$$x_{p,D} = \begin{cases} 1 & \text{se } rand < 0.25 \\ 0 & \text{caso contrário} \end{cases} \tag{30}$$

O deslocamento individual deixou de utilizar o valor de $step_{ind}$ como parâmetro de comparação para realizar a inversão de *bits*, e passou a ser a quantidade de *bits* que serão invertidos. Dessa forma, o $step_{ind}$ é uma porcentagem do número de dimensões, que é decrementado linearmente, de forma que pelo menos 1 (um) *bit* seja sempre invertido para que o deslocamento não seja nulo, resultando na perda de habilidade de busca. Após o deslocamento individual, o operador de alimentação do peixe é realizado fazendo as devidas atualizações nos pesos dos peixes, e não sofreu alterações, sendo realizado igualmente às versões anteriores do BFSS.

O movimento coletivo instintivo usa novamente o valor de $\Delta x_{p,D}$ para realizar a movimentação do agente de busca. Embora o vetor seja binário, o processo utiliza o cálculo apresentado na Equação (31):

$$\Delta x_{p,D}^{t} = \begin{cases} 1 & \text{se } x_{p,D}^{t} = 1 \text{ e } x_{p,D}^{t-1} = 0 \\ 0 & \text{se } x_{p,D}^{t} = x_{p,D}^{t-1} \\ -1 & \text{se } x_{p,D}^{t} = 0 \text{ e } x_{p,D}^{t-1} = 1 \end{cases} \tag{31}$$

Assim, o agente realizará o deslocamento somente a partir das dimensões que sofreram alterações.

Seguindo, realiza-se um processo similar ao BFSS, utilizando o vetor $M = \Delta x_{p,D}^{t}$, e, comparando-o com $x_{p,D}^{t}$, marca-se as dimensões em que $x_{p,D}^{t} \neq M_D$. Então, uma porcentagem de dimensões dentre as marcadas terão seu valor invertido, e não somente uma dimensão como anteriormente, como mostra a Equação 32:

$$x_{p,D_{marked}}^{t+1} = \overline{x_{p,D_{marked}}^{t}} \tag{32}$$

sendo que "D_{marked}"são as dimensões marcadas na condição dada.

O movimento coletivo volitivo segue de forma similar. O processo de cálculo do baricentro é o mesmo do BFSS, porém o deslocamento não se dá em uma única dimensão, e sim em uma porcentagem delas, de modo que $x_{p,D}^{t} \neq Bari_D$. Esse percentual vai depender do valor de $step_{vol}$ que continua sendo decrementado linearmente com o passar das iterações.

3.4.2 Mudanças para o BFSS Simplificado

A versão simplificada do BFSS foi proposta por Santana et al. (2019) e, como o próprio nome sugere, tem objetivo de simplificar o algoritmo. Nele não serão necessários parâmetros como $steps$ e $tresholds$, facilitando seu desenvolvimento, entendimento e implementação.

A inicialização da população é igual à do BFSS inicial, com probabilidades iguais das dimensões serem zero ou um. O deslocamento individual não mais depende do valor de $step_{ind}$. Aqui, a movimentação se dá pela inversão de duas dimensões escolhidas aleatoriamente. O operador de alimentação permanece o mesmo que em todos os outros casos.

O movimento coletivo instintivo realiza um processo de contagem de quantos zeros e quantos uns existem nas dimensões de cada indivíduo e, para cada um deles, se soma o valor da variação de *fitness*, de forma a demonstrar o quão representativa é a quantidade de uns ou de zeros no valor de *fitness*. Esse processo fica melhor ilustrado no Algoritmo 1.

Algoritmo 1 – Vetor Instintivo

para *cada peixe p* **faça**
  para *cada dimensão D* **faça**
    se $x_{p,D} == 1$ **então**
      $cont\_ones_D \leftarrow$ cont_ones$_D + \Delta f$
    senão,
      $cont\_zeros_D \leftarrow$ cont_zeros$_D + \Delta f$
    finaliza se
  finaliza para
finaliza para

Fonte: Retirado de Santana et al. (2019)

Por fim, o movimento ocorre de acordo com a equação (33):

$$x_{p,D}^{t+1} = \begin{cases} 1 & \text{se } cont\_zeros_D < cont\_ones_D \\ 0 & \text{se contrário} \end{cases} \tag{33}$$

onde $cont\_zeros_D < cont\_ones_D$ representa que a relevância da quantidade de números um sobre o fitness é maior que a de quantidade de zeros. O movimento coletivo

Algoritmo 2 – Novo Cálculo do Baricentro

requer Aplicar o torneio binário para seleção dos três peixes;
para *cada dimensão D* **faça**
  para *cada peixe p* **faça**
    se $x_{p,D} == 1$ **então**
      $weight\_ones_D \leftarrow$ weight_ones$_D + W_p$
    senão,
      $weight\_zeros_D \leftarrow$ weight_zeros$_D + W_p$
    finaliza se
  finaliza para
  se $weight\_zeros_D < weight\_ones_D$ **então**
  $Bari_D^{t+1} \leftarrow 1$
  senão,
  $Bari_D^{t+1} \leftarrow 0$
  finaliza se
finaliza para

Fonte: Retirado de Santana et al. (2019)

volitivo segue o mesmo princípio do instintivo, porém utilizando o valor do peso no lugar do valor de *fitness*. Além disso, decidiu-se que o cálculo do baricentro poderia ser realizado com o peso de apenas três peixes do cardume, escolhidos a partir de

um torneio binário. O novo processo de cálculo do baricentro e o movimento coletivo volitivo podem ser observados no Algoritmo 2 e na equação (34), respectivamente.

$$x_{p,D}^{t+1} = \begin{cases} 1 & \text{se } cont\_zeros_D < cont\_ones_D \\ 0 & \text{se contrário} \end{cases} \tag{34}$$

onde $weight\_zeros_D < weight\_ones_D$ representa que a relevância da quantidade de números um sobre o peso é maior que a de quantidade de zeros.

4 EXPERIMENTAÇÃO E RESULTADOS

Neste capítulo está descrita a aplicação dos algoritmos apresentados para a resolução do *One max Problem*, problema binário que consiste na busca por preencher um vetor binário com a maior concentração de *bits* um, e o Problema da Mochila, mais conhecido pelo seu nome original em inglês, *Knapsack Problem*, que consiste no empacotamento de uma mochila de volume inteiro V com N itens que possuem valores e volumes variados (ROSS; TSANG, 1989). Serão abordados os resultados obtidos a partir da simulação destes problemas com diversas dimensões e sua análise estatística, discutindo e comparando o desempenho dos métodos.

Os resultados do Algortimo Genético Binário (*Binary Genetic Algorithm*) e da Evolução Diferencial Booleana (*Boolean Differential Evolution*) apresentados foram retirados do trabalho "Algoritmos Evolutivos para Otimização Binária", (Santos, Walace R. Lopes, 2019 (em processo de publicação)), que utilizou a mesma base de dados e foi desenvolvido paralelamente com o trabalho aqui apresentado. Ressalta-se apenas que os resultados de tal trabalho não apresentam a média de iterações até a convergência, não sendo possível realizar tal análise para estes casos.

Todos os algoritmos foram escritos e testados no software MATLAB® versão R2017B, utilizando sistema operacional Microsoft Windows 10 Professional-64bit, tendo como hardware um Processador Intel® Core™ i5-7400 e 8Gb Memória RAM DDR4 - 2400MHz em *single channel*.

4.1 *ONE MAX PROBLEM*

O primeiro problema escolhido para realizar as simulações dos algoritmos binários foi o *One Max Problem*, que consiste em obter a maior concentração de *bits* um possíveis dentro de um vetor ou matriz binários, caracterizando-se assim, um problema de maximização. Este problema foi escolhido por ser de fácil compreensão, e por ser simples sua implementação, uma vez que basta realizar a avaliação de *fitness* diretamente dos vetores binários da população de agentes de busca.

Para dar validade aos algoritmos, o problema foi implementado em quatro variações, com dimensões iguais a 100, 500, 1000 e 5000, onde cada uma foi simulada

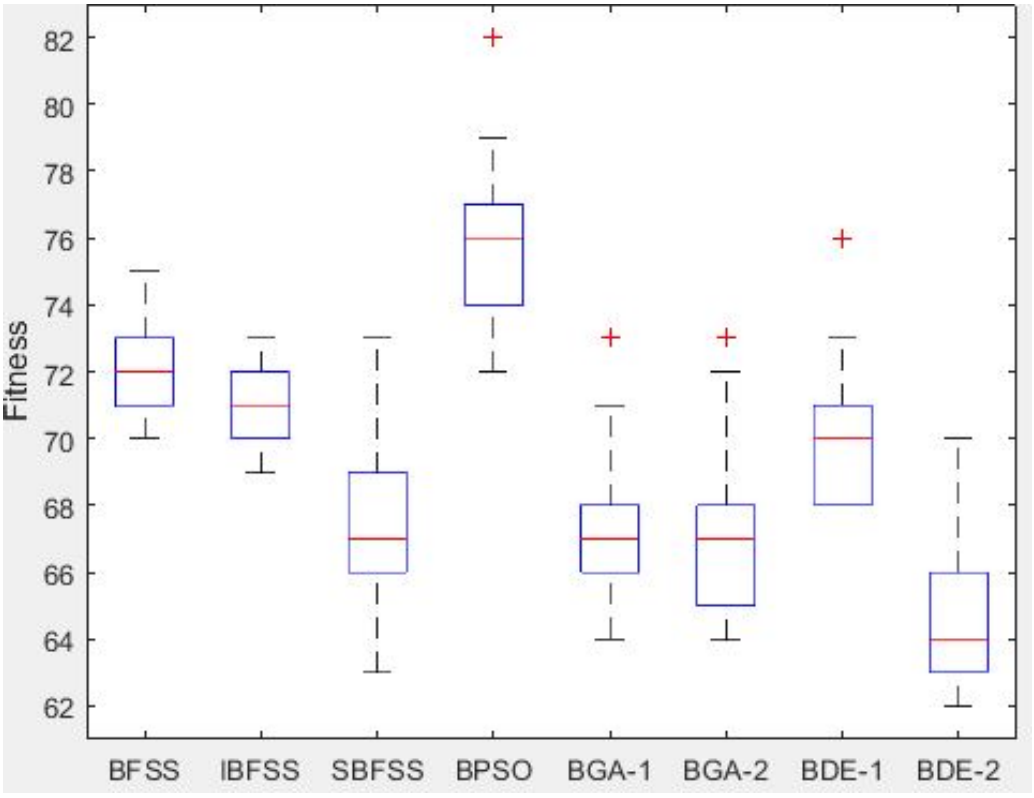
50 vezes utilizando uma população de 30 agentes de busca, com valor de iterações máximas iguais a 5000, sendo esse também utilizado como critério de parada do algoritmo. O cálculo do *fitness* é dado pela a soma da quantidade de dimensões iguais a um em cada agente, isto é, aquele que apresentar a maior concentração de *bits* um possuirá melhor *fitness*.

Nas Tabelas a seguir serão apresentados dados como o maior e menor *fitness* encontrados dentre as 50 simulações para determinado algoritmo, assim como a média e o desvio padrão de todos os *fitness* encontrados em todas as simulações. Além disso, também será apresentada uma média de quantas iterações o algoritmo necessitou para convergir em cada caso, com a ideia de mostrar a agilidade de uns em relação aos demais.

4.1.1 100 Dimensões ($D = 100$)

O resultado obtido para a simulação do *One Max Problem* com 100 dimensões pode ser observado na Figura 7, que apresenta o gráfico tipo boxplot, e na Tabela 1.

Figura 7 – *Boxplot* do *Fitness* do *One Max Problem* com 100 dimensões



Fonte: Autoria própria.

Tabela 1 – Tabela de Resultados *One Max Problem* com 100 dimensões

Algoritmo	Maior Fitness	Menor Fitness	Média	Desvio Padrão	Média Iterações
BFSS	75	70	72,02	1,10	1739,22
IBFSS	73	69	70,90	1,11	1370,82
SBFSS	73	63	67,52	2,12	1848,22
BPSO	82	72	75,56	1,81	1853,58
BGA-1	73	64	67,08	1,86	-
BGA-2	73	64	67,06	2,05	-
BDE-1	76	68	70	1,84	-
BDE-2	70	62	64,6	1,84	-

Fonte: Autoria própria.

Aqui podemos notar que o BPSO apresentou melhores resultados se comparado aos demais algoritmos, com valor médio de *fitness* próximo aos 75.6% (0.756) e melhor respota encotrada igual a 82%. A média de iterações até a convergência dentre todos os métodos aqui dispostos foi próxima, a menos do caso do IBFSS que atingiu o estado estacionário antes. Por possuir poucas dimensões, nota-se que os resultados do maior *fitness* encontrado foram relativamente próximos, sendo iguais nos casos do IBFSS, SBFSS, BGA-1 e BGA-2.

4.1.2 500 Dimensões ($D = 500$)

Da mesma forma, o resultado obtido para a simulação do *One Max Problem* com 500 dimensões é apresentado na Figura 8 e na Tabela 2.

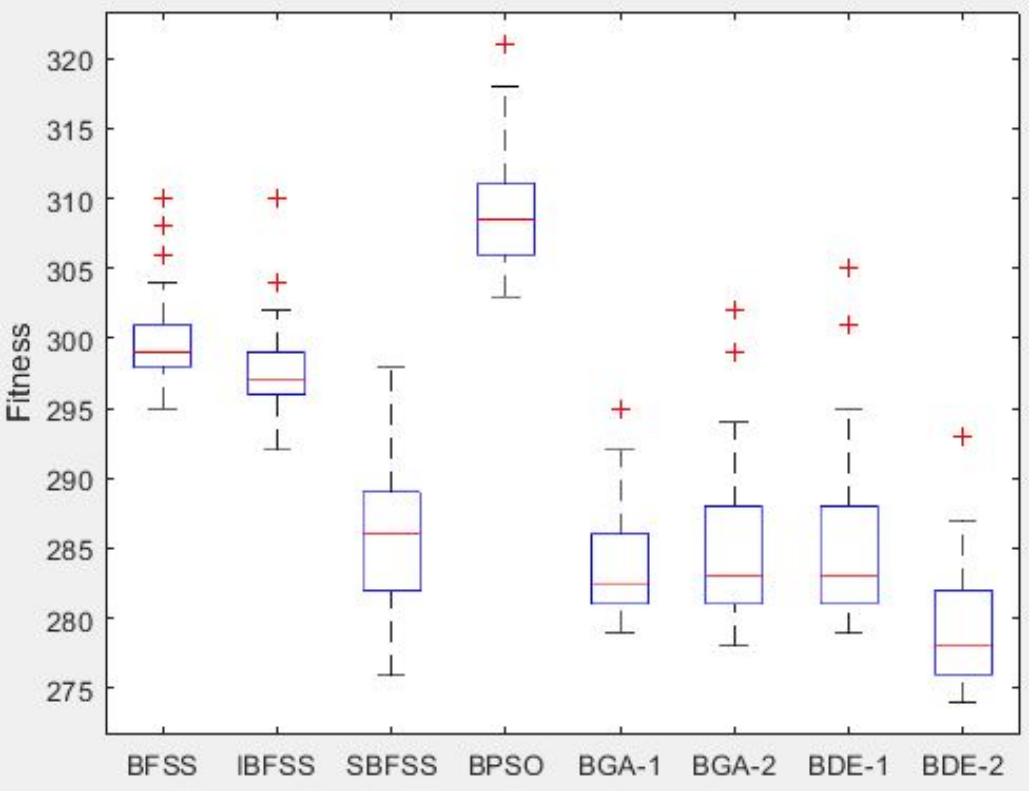
Tabela 2 – Tabela de Resultados *One Max Problem* com 500 dimensões

Algoritmo	Maior Fitness	Menor Fitness	Média	Desvio Padrão	Média Iterações
BFSS	310	295	300,04	3,33	2154,62
IBFSS	310	292	297,28	3,06	1439,88
SBFSS	298	276	286,10	4,87	1648,10
BPSO	321	303	309,04	4,08	2334,38
BGA-1	295	279	283,68	3,44	-
BGA-2	302	278	284,82	5,47	-
BDE-1	305	279	284,96	5,64	-
BDE-2	293	274	279,16	4,19	-

Fonte: Autoria própria.

Novamente, o algoritmo de BPSO se mostrou muito superior aos demais. Aqui, os algoritmos IBFSS e BFSS obtiveram o mesmo melhor resultado, e ambos também apresentaram melhores resultados que os algoritmos evolutivos.

Figura 8 – *Boxplot* do *Fitness* do *One Max Problem* com 500 dimensões



Fonte: Autoria própria.

4.1.3 1000 Dimensões ($D = 1000$)

O resultado obtido para a simulação do *One Max Problem* com 1000 dimensões encontra-se na Figura 9 e na Tabela 3. Para 1000 e 5000 dimensões não há simulações para os algoritmos BGA e BDE, pois são duas dimensões muito superiores as normalmente utilizadas para simulação e que tomam demasiado tempo para serem simuladas, e aqui foram utilizadas por curiosidade em saber qual seria as respostas apresentadas pelos algoritmos.

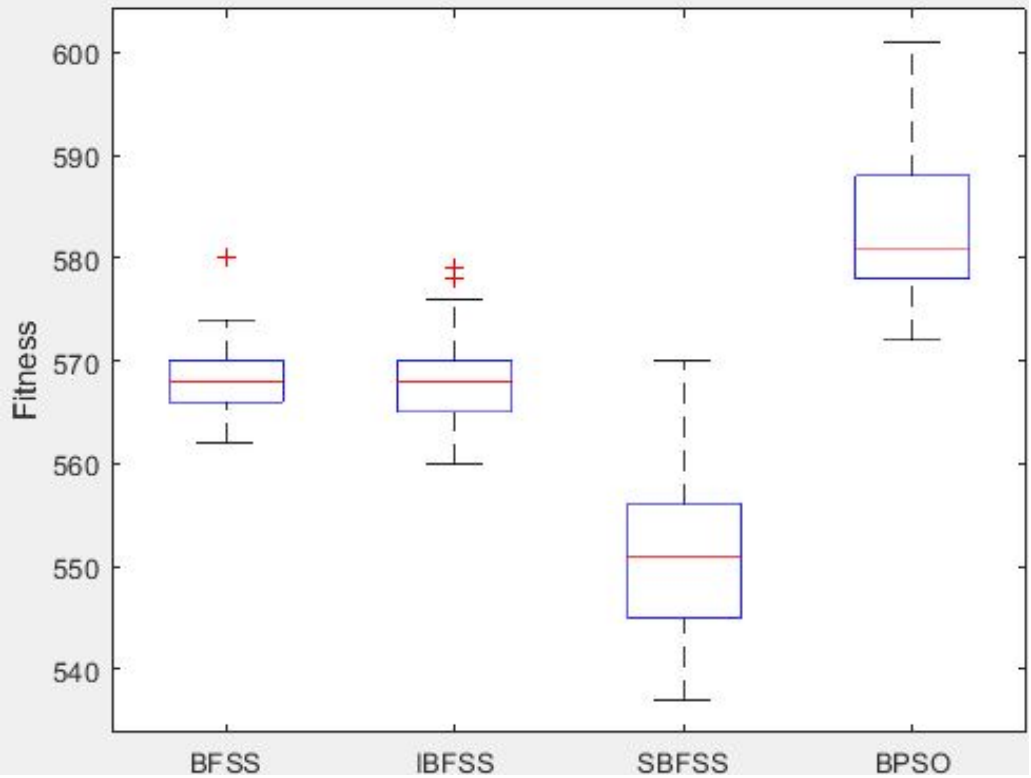
Tabela 3 – Tabela de Resultados *One Max Problem* com 1000 dimensões

Algoritmo	Maior Fitness	Menor Fitness	Média	Desvio Padrão	Média Iterações
BFSS	580	562	568,48	3,29	2253,74
IBFSS	579	560	568,46	4,44	1819,90
SBFSS	570	537	551,08	7,68	1065,30
BPSO	601	572	583,36	7,32	2707,46

Fonte: Autoria própria.

Nota-se que com o aumento do número de dimensões o valor de desvio padrão começa também a aumentar, pois como a área de busca é maior, as variações tendem a ser maiores. Nesse caso, pode-se observar que o BFSS é o que possui me-

Figura 9 – *Boxplot* do *Fitness* do *One Max Problem* com 1000 dimensões



Fonte: Autoria própria.

nor desvio, se mostrando convergente para um mesmo ponto. Em conformidade com os casos anteriores o BPSO continua atingindo as melhores performances.

4.1.4 5000 Dimensões ($D = 5000$)

Por fim, o resultado obtido para a simulação do *One Max Problem* com 5000 dimensões encontra-se na Figura 10 e na Tabela 4.

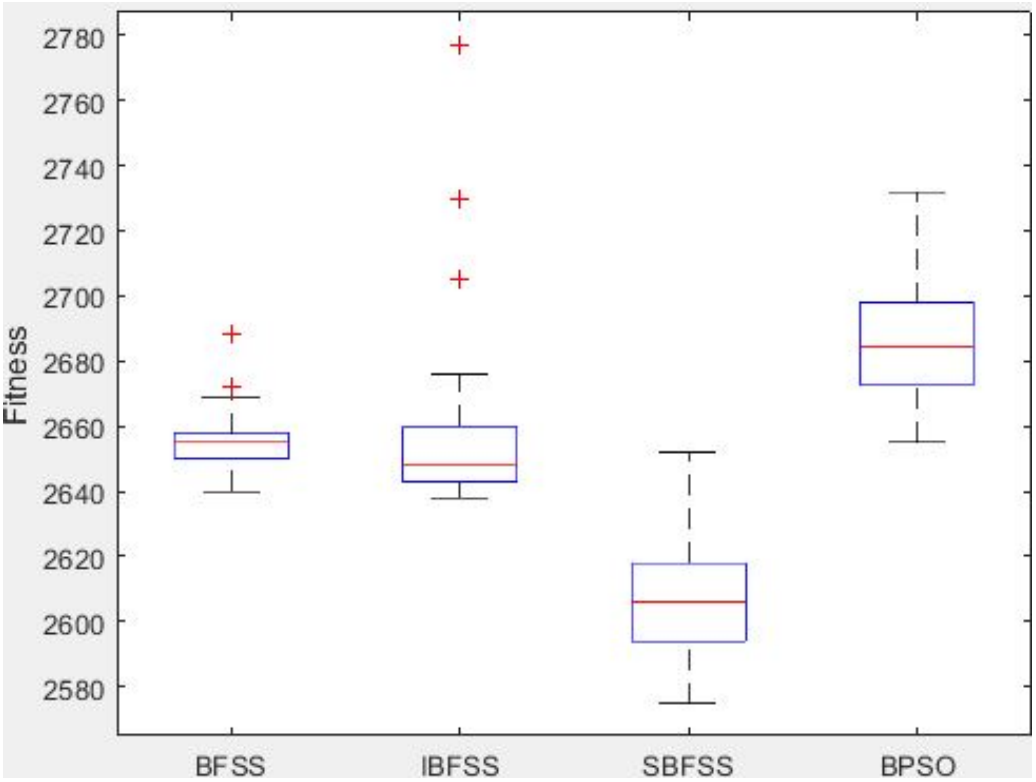
Tabela 4 – Tabela de Resultados *One Max Problem* com 5000 dimensões

Algoritmo	Maior Fitness	Menor Fitness	Média	Desvio Padrão	Média Iterações
BFSS	2688	2640	2655,82	8,83	2071,28
IBFSS	2777	2638	2656,20	24,07	1719,74
SBFSS	2652	2575	2605,50	16,36	1648,1
BPSO	2732	2655	2687,30	18,10	2334,38

Fonte: Autoria própria.

A análise com 5000 dimensões foi testada com o intuito de avaliar até que ponto os algoritmos conseguiriam ser utilizados, já que esse número de dimensões é alto, e supera o normalmente utilizado para teste. É interessante observar que o IBFSS obteve melhor resultado geral, inclusive maior que o BPSO, e o BFSS manteve

Figura 10 – *Boxplot* do *Fitness* do *One Max Problem* com 5000 dimensões



Fonte: Autoria própria.

o seu valor de desvio padrão muito abaixo dos demais. Na média, o BSPO superou os inspirados em cardumes de peixes para o problema proposto, mostrando que os pontos destoantes (melhores resultados) desses não representam que sejam melhor que o BPSO.

4.2 $0/1$ *KNAPSACK PROBLEM*

O Problema da Mochila, mais conhecido pelo seu nome original em inglês *Knapsack Problem*, é um problema *benchmark* muito utilizado para testar técnicas de seleção de variáveis (*feature selection*). A ideia por trás dele é simples: preencher o volume de uma mochila com itens de tal forma que se obtenha o maior valor possível.

Cada item/variável possui um valor e um volume, os quais devem ser selecionados de forma que não extrapolem o volume máximo da mochila. Nesse caso, o valor do *fitness* será a soma do valor de todos os itens escolhidos. Deve ser realizado um processo de penalização do *fitness* caso a soma do volume ultrapassasse o máximo permitido, fazendo com que o agente desse conjunto não possa ser escolhido como o resultado factível. O objetivo é que, ao invés de se tentar todas as combinações pos-

síveis para encontrar o melhor preenchimento, utilize-se os algoritmos apresentados como forma de solucionar o problema quando o número de dimensões se torna muito grande e a solução matemática se torna trabalhosa.

O $0/1$ *Knapsack Problem* utiliza vetores binários, em que os *bits* iguais a 1 sinalizam que o item correspondente daquela dimensão foi selecionado, e os *bits* 0 indicam o contrário.

O problema também foi implementado em quatro variações, com 30, 100, 250 e 500 dimensões, em que estes representam o número de itens dados para seleção. Os parâmetros dos algoritmos de otimização foram os mesmos usados para o *One Max Problem*, isto é, população de 30 indivíduos, 50 vezes simulações independentes e número máximo de iterações iguais a 5000, sendo esse o critério de parada.

Os dados dos itens e o volume da mochila utilizados foram gerados aleatoriamente para os testes. O melhor resultado alcançável para cada caso (ótimo global), foi obtido através de um programa auxiliar, facilitando o processo de cálculo de *fitness* das simulações. Todos os dados estão presentes no Apêndice A.

Nas tabelas de resultados das simulações do *Knapsack Problem* é apresentado o *Fitness %*, que apresenta a porcentagem da melhor resposta encontrada em relação a melhor resposta alcançável presente no Apêndice A. O cálculo para o *Fitness %* é mostrado na Equação (35).

$$Fitness\% = \frac{Maior\ Fitness}{Melhor\ Fitness} \tag{35}$$

Para efeito de comparação de desempenho entre vários algoritmos, serão também usadas duas variações do *Binary Genetic Algorithm* e duas do *Boolean Differential Evolution*, conforme dados fornecidos a seguir:

- BGA-1 - genético clássico:
  - Processo de seleção realizado por roleta;
  - Taxa de crossover de 70%;
  - Taxa de mutação de 10%;
- BGA-2
  - Processo de seleção realizado por roleta;
  - Taxa de crossover de 100%;

  - Taxa de mutação de 10%;

- BDE-1
  - Tipo de mutação: Rand;
  - Quantidade de diferenças poderadas: 1;
  - Tipo de crossover: binomial.

- BDE-2
  - Tipo de mutação: Best;
  - Quantidade de diferenças poderadas: 1;
  - Tipo de crossover: binomial.

Estes foram os escolhidos do trabalho anteriormente citado no segundo parágrafo do Capítulo 4, por serem os que apresentam os resultados mais próximos aos apresentados pelos algoritmos de Inteligência de Enxame.

4.2.1 30 Dimensões ($D = 30$)

O resultado obtido para a simulação do *Knapsack Problem* com 30 dimensões pode ser observado na Figura 11 e na Tabela 5.

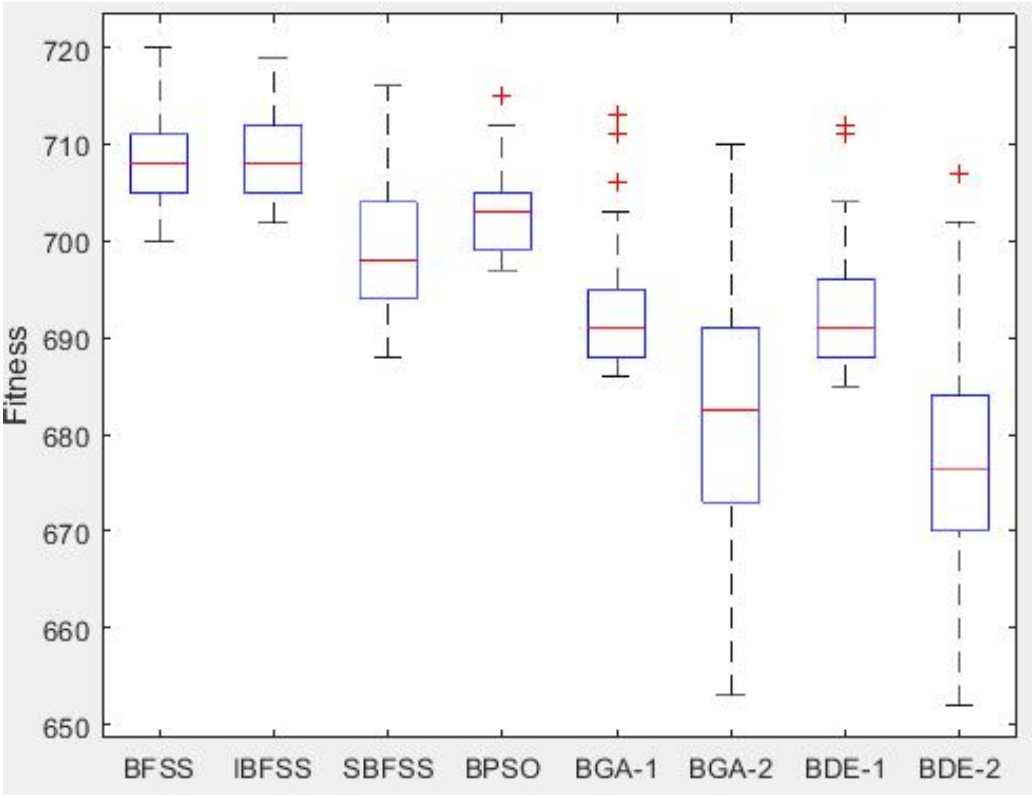
Tabela 5 – Tabela de Resultados *Knapsack Problem* com 30 dimensões

Algoritmo	Maior Fitness	Fitness %	Média	Desvio Padrão	Média Iterações
BFSS	720	98,77	707,80	4,27	2498,84
IBFSS	719	98,63	708,70	4,02	2414,4
SBFSS	716	98,22	699,52	6,98	2053,08
BPSO	715	98,08	702,70	4,18	2428,88
BGA-1	713	97,81	692,84	6,09	-
BGA-2	710	97,39	682,48	12,46	-
BDE-1	712	97,67	692,68	6,20	-
BDE-2	707	96,98	677,64	11,95	-

Fonte: Autoria própria.

Por se tratar de um problema de poucas dimensões, já esperava-se que os resultados fossem muito próximos do ótimo global. Sendo assim, todos apresentam *fitness* próximo a 100%, apenas com o objetivo de mostrar a convergência e funcionamento dos algoritmos.

Figura 11 – *Boxplot* do *Fitness* do *Knapsack Problem* com 30 dimensões



Fonte: Autoria própria.

4.2.2 100 Dimensões ($D = 100$)

As performances para o *Knapsack Problem* com 100 dimensões podem ser vistas na Figura 12 e na Tabela 6.

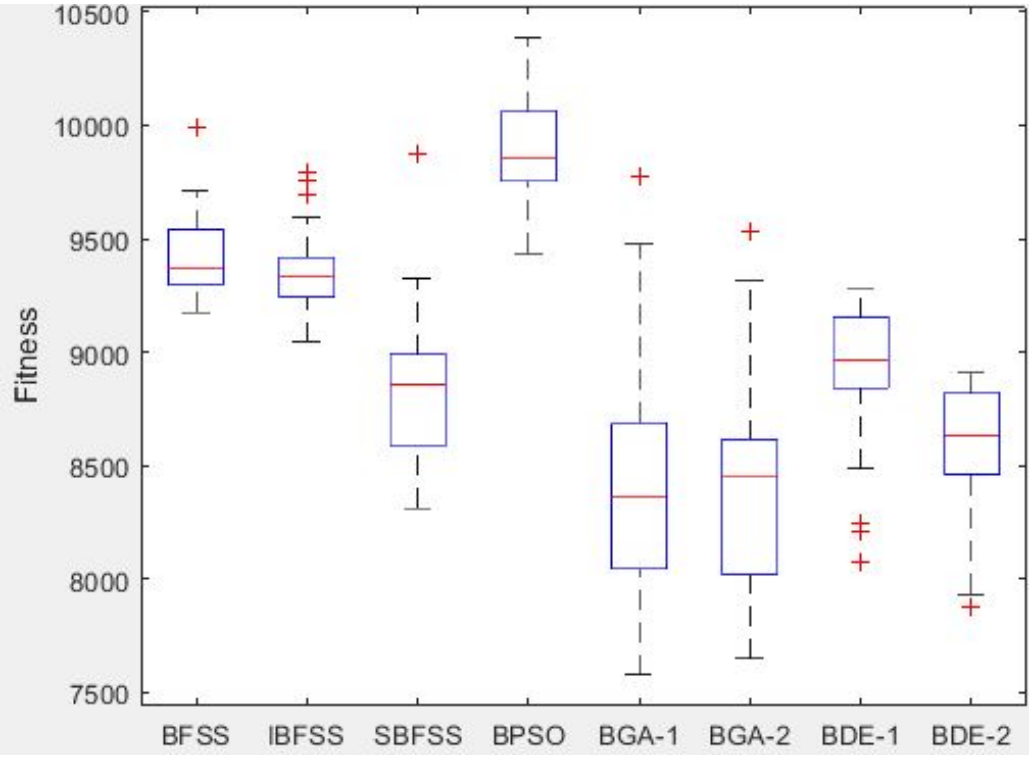
Tabela 6 – Tabela de Resultados *Knapsack Problem* com 100 dimensões

Algoritmo	Maior Fitness	Fitness %	Média	Desvio Padrão	Média Iterações
BFSS	9993	79,94	9409,28	164,95	2405,06
IBFSS	9790	78,32	9347,82	161,11	2144,34
SBFSS	9874	78,99	8837,36	287,89	1735,14
BPSO	10385	83,08	9897,3	215,32	2822,6
BGA-1	9778	78,22	8406,86	493,88	-
BGA-2	9532	76,26	8395,86	438,03	-
BDE-1	9284	74,28	8937,20	278,74	-
BDE-2	8914	71,32	8606,06	249,31	-

Fonte: Autoria própria.

Nesse caso, o algoritmo de BPSO obteve vantagem sobre os demais métodos, porém, pode-se observar que o maior *fitness* encontrado pelo algoritmo BFSS é maior que a média do BPSO, mostrando que algumas melhorias no seu desenvolvimento podem impactar no seu desempenho, tornando-o equiparável ao BPSO. No caso geral, os métodos de enxame se comportaram melhor que os evolutivos. As propostas

Figura 12 – *Boxplot* do *Fitness* do *Knapsack Problem* com 100 dimensões



Fonte: Autoria própria.

de DE, entretanto, chegaram a melhores performances que o SBFSS.

4.2.3 250 Dimensões ($D = 250$)

O resultado obtido para o problema proposto com 250 dimensões é apresentado na Figura 13 e na Tabela 7.

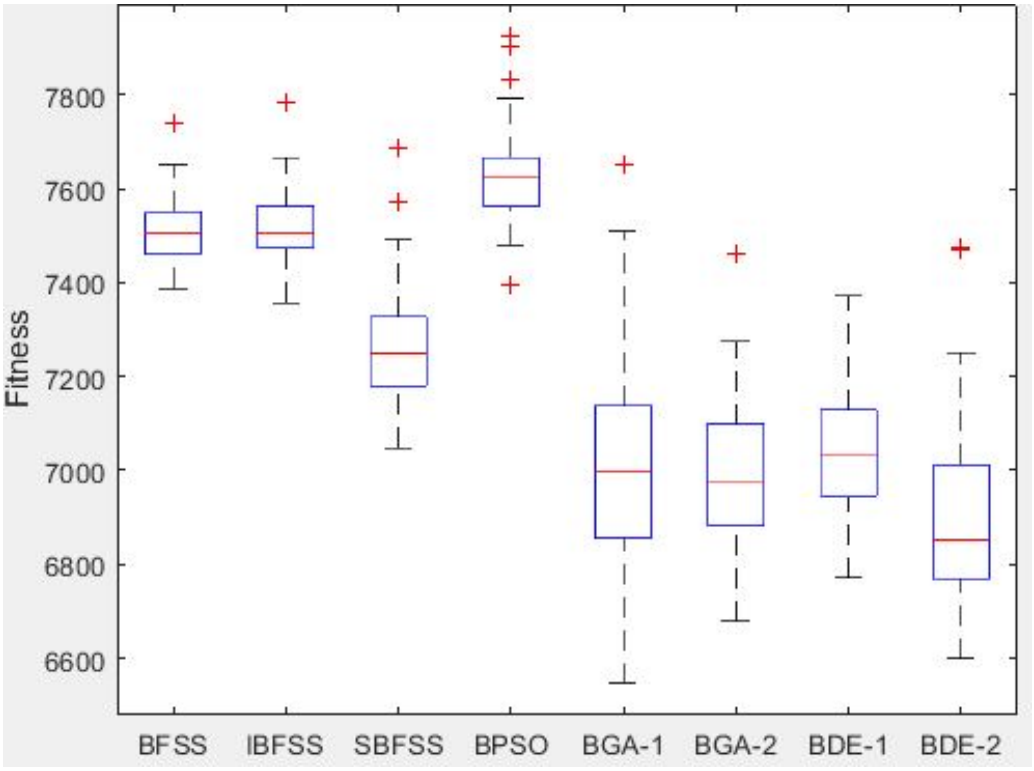
Tabela 7 – Tabela de Resultados *Knapsack Problem* com 250 dimensões

Algoritmo	Maior Fitness	Fitness %	Média	Desvio Padrão	Média Iterações
BFSS	7737	77,41	7504,86	67,27	2659,32
IBFSS	7782	77,86	7519,54	75,96	2379,40
SBFSS	7687	76,91	7264,78	126,96	1363,64
BPSO	7924	79,28	7631,28	104,16	2516,60
BGA-1	7652	76,56	6989,54	212,39	-
BGA-2	7461	74,65	6990,14	147,21	-
BDE-1	7371	73,75	7041,26	131,31	-
BDE-2	7473	74,77	6895,98	192,63	-

Fonte: Autoria própria.

Pode-se ver claramente que as variantes do BFSS obtiveram melhores resultados que todos os algoritmos evolutivos, mostrando que com o crescer do número de dimensões, tendem a apresentar melhores resultados em relação aos AEs, mesmo

Figura 13 – *Boxplot* do *Fitness* do *Knapsack Problem* com 250 dimensões



Fonte: Autoria própria.

que ainda atrás do BPSO, que se mantém o melhor. Aqui outro detalhe bem visível é a dispersão do BGA-1, que apresentou o valor acima dos demais.

4.2.4 500 Dimensões ($D = 500$)

O resultado obtido para a simulação do *Knapsack Problem* com 500 dimensões pode ser observado na figura 14 e na tabela 8.

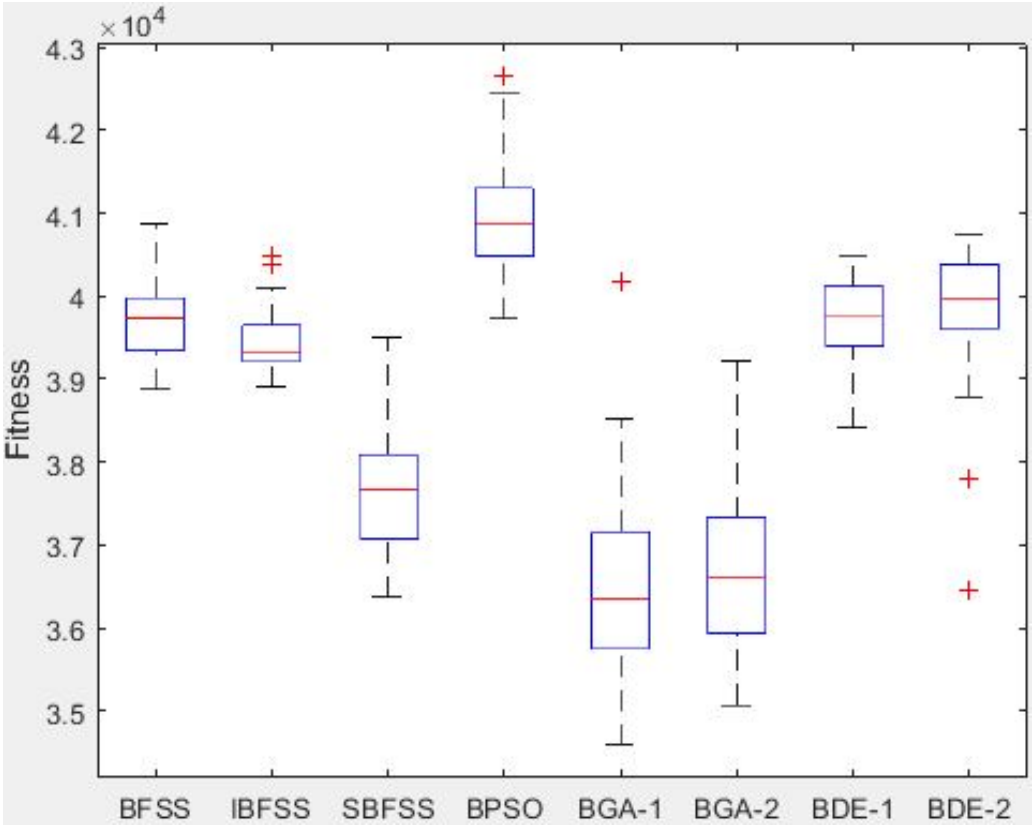
Tabela 8 – Tabela de Resultados *Knapsack Problem* com 500 dimensões

Algoritmo	Maior Fitness	Fitness %	Média	Desvio Padrão	Média Iterações
BFSS	40874	71,07	39695,02	421,54	2506,88
IBFSS	40485	70,39	39438,10	387,14	2020,88
SBFSS	39504	68,69	37672,40	713,40	1190,18
BPSO	42653	74,16	40941,76	680,61	2932,46
BGA-1	40187	69,87	36500,90	1033,28	-
BGA-2	39219	68,19	36658,50	914,28	-
BDE-1	40480	70,38	39719,04	514,29	-
BDE-2	40736	70,83	39843,28	770,74	-

Fonte: Autoria própria.

Por fim, a maior dimensão testada apresentou resultados peculiares. O BPSO manteve sua liderança, mostrando-se o algoritmo mais eficaz dentre os avaliados,

Figura 14 – *Boxplot* do *Fitness* do *Knapsack Problem* com 500 dimensões



Fonte: Autoria própria.

resultado esse já esperado, observando-se as experimentações dos artigos que foram utilizados como referência para este trabalho. Contudo, os algoritmos baseados na Evolução Diferencial, apresentaram resultados muito próximos aos BFSS, fato esse que não se observou nos casos anteriores, sinalizando assim a necessidade de serem reavaliados, podendo esses resultados serem frutos de um possível vício na série de seleções. Além disso, se faz necessário também testes em casos reais, para que seja comprovada a funcionalidade vista nos testes.

Os resultados gerais mostram que das propostas consideradas clássicas - BPSO e BGA - o primeiro foi o que alcançou melhores desempenhos gerais para as tarefas propostas. Dentre os modelos de FSS, o BFSS e o IBFSS alcançaram desempenhos geralmente próximos, superando a versão Simplificada. Mais ainda, os métodos de enxame conseguiram, no caso geral, chegar a resultados superiores aos evolutivos.

5 CONCLUSÕES E PERSPECTIVAS

Neste trabalho foram apresentadas versões binárias dos algoritmos Otimização por Enxame de Partículas (BPSO) e três variantes da Busca por Cardume de Peixes (FSS): o FSS binário (BFSS), BFSS Melhorado (IBFSS) e o BFSS simplificado (SBFSS). Como estudos de casos foram abordados o *One Max Problem* e o problema da mochila (*Knapsak Problem*), com variação no número de dimenões. Como forma de comparação, foram também incluídos os desempenhos de duas versões binárias do Algoritmo Genético (BGA) e da Evolução Diferencial (BDE).

Dado o desenvolvimento e implementação dos algoritmos e os resultados apresentados, alguns detalhes sobre o comportamento dos algoritmos foram observados, como o modo de movimentação dos agentes de busca. Há uma relevante diferença entre o BPSO e o BFSS, já que o último apresenta a movimentação de expansão e contração para evitar mínimos locais, e mesmo o BPSO não possuindo essa movimentação, atingiu melhores resultados.

Os resultados computacionais deixaram evidente que o BPSO se mostrou superior aos demais métodos, seguido pelos algoritmos BFSS, IBFSS e SBFSS. Assim, para estes casos, pode-se dizer que os algoritmos de inteligência de enxame foram superiores que os evolutivos, principalmente sobre os BGAs.

Pode-se também, futuramente, estudar o impacto que os valores dos dados da mochila e dos itens, juntamente com seus valores, volumes, e consequentemente a dispersão desses dados, têm sobre os resultados dos algoritmos, uma vez que o empacotamento por partes, isto é, separando os itens de forma ordenada, para então realizar o processo de otimização, pode se mostrar um método mais eficiente que o apresentado. Ademais, faz-se necessário avaliar os métodos acima em problemas reais de seleção de variáveis.

Outra proposta de pesquisa a ser realizada é sobre o operador de peso no BFSS, já que se deseja encontrar uma forma de manipular o peso de forma binária, e não mais utilizando valores reais.

REFERÊNCIAS

ÁVILA, Sérgio Luciano et al. Algoritmos genéticos aplicados na otimização de antenas refletoras. Florianópolis, SC, 2002. Citado na página 20.

BENI, Gerardo; WANG, Jing. Swarm intelligence in cellular robotic systems. In: **Robots and biological systems: towards a new bionics?** [S.l.]: Springer, 1993. p. 703–712. Citado na página 16.

CARNEIRO, Raphael F; BASTOS-FILHO, Carmelo JA. Improving the binary fish school search algorithm for feature selection. In: IEEE. **2016 IEEE Latin American Conference on Computational Intelligence (LA-CCI)**. [S.l.], 2016. p. 1–6. Citado na página 36.

CASTRO, Leandro Nunes De. **Fundamentals of natural computing: basic concepts, algorithms, and applications**. [S.l.]: Chapman and Hall/CRC, 2006. Citado 3 vezes nas páginas 12, 15 e 16.

CORNO, Lyn; KANFER, Ruth. Chapter 7: the role of volition in learning and performance. **Review of research in education**, Sage Publications Sage CA: Thousand Oaks, CA, v. 19, n. 1, p. 301–341, 1993. Citado na página 30.

DARWIN, Charles. **The Origin of Species; And, the Descent of Man**. [S.l.]: Modern library, 1859. Citado 2 vezes nas páginas 12 e 16.

EBERHART, Russell; KENNEDY, James. Particle swarm optimization. In: CITESEER. **Proceedings of the IEEE international conference on neural networks**. [S.l.], 1995. v. 4, p. 1942–1948. Citado 2 vezes nas páginas 12 e 18.

FILHO, Adailson de Castro Queiroz. **Uma ferramenta para análise de algoritmos de otimização bioinspirados**. 2016. Tese (Doutorado) — Universidade Federal de Pernambuco, 2016. Citado na página 17.

FILHO, Carmelo JA Bastos et al. A novel search algorithm based on fish school behavior. In: IEEE. **2008 IEEE International Conference on Systems, Man and Cybernetics**. [S.l.], 2008. p. 2646–2651. Citado 4 vezes nas páginas 13, 26, 28 e 30.

FILHO, Marco Antonio Itaborahy. **Análise de algoritmos genéticos e evolução diferencial para otimização de funções não-lineares multimodais**. 2018 — Universidade Tecnológica Federal do Paraná, 2018. Citado na página 16.

FOGEL, Lawrence J; OWENS, Alvin J; WALSH, Michael. J. 1966. **Artificial intelligence through simulated evolution**, 1966. Citado na página 16.

HERSOVICI, Michael et al. The shark-search algorithm. an application: tailored web site mapping. **Computer Networks and ISDN Systems**, Elsevier, v. 30, n. 1-7, p. 317–326, 1998. Citado na página 27.

HOLLAND, John H. Adaptation in natural and artificial systems ann arbor. **The University of Michigan Press**, v. 1, p. 975, 1975. Citado 2 vezes nas páginas 13 e 16.

KENNEDY, James; EBERHART, Russell C. A discrete binary version of the particle swarm algorithm. In: IEEE. **1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation**. [S.l.], 1997. v. 5, p. 4104–4108. Citado 2 vezes nas páginas 13 e 22.

KIRA, Kenji; RENDELL, Larry A et al. The feature selection problem: Traditional methods and a new algorithm. In: **Aaai**. [S.l.: s.n.], 1992. v. 2, p. 129–134. Citado na página 15.

LUO, Fang-fang; CHEN, Guo-long; GUO, Wen-zhong. An improved"fish-search"algorithm for information retrieval. In: IEEE. **2005 International Conference on Natural Language Processing and Knowledge Engineering**. [S.l.], 2005. p. 523–528. Citado na página 27.

MENEZES, Sandro Loiola; FREITAS, Rebeca Schroeder; PARPINELLI, Rafael Stubs. Mineraçao em grandes massas de dados utilizando hadoop mapreduce e algoritmos bio-inspirados: Uma revisao sistemática. **Revista de Informática Teórica e Aplicada**, v. 23, n. 1, p. 69–101, 2016. Citado na página 12.

PUCHTA, Erickson Diogo Pereira et al. **Controle PID gaussiano com otimização dos parâmetros das funções gaussianas usando algoritmo genético e PSO**. 2016. Dissertação (Mestrado) — Universidade Tecnológica Federal do Paraná, 2016. Citado 4 vezes nas páginas 15, 18, 19 e 20.

RECHENBERG, Ingo. Evolution strategy: Optimization of technical systems by means of biological evolution. **Fromman-Holzboog, Stuttgart**, v. 104, p. 15–16, 1973. Citado na página 16.

ROSS, Keith W; TSANG, Danny HK. The stochastic knapsack problem. **IEEE Transactions on communications**, IEEE, v. 37, n. 7, p. 740–747, 1989. Citado na página 40.

SANTANA, Clodomir J et al. Sbfss: Simplified binary fish school search. In: IEEE. **2019 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.], 2019. p. 2595–2602. Citado 5 vezes nas páginas 26, 27, 33, 37 e 38.

SARGO, João AG et al. Binary fish school search applied to feature selection: Application to icu readmissions. In: IEEE. **2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)**. [S.l.], 2014. p. 1366–1373. Citado 2 vezes nas páginas 31 e 33.

SCHRAMM, Fábio Kellermann. Projeto de sistemas de produção na construção civil utilizando simulação computacional como ferramenta de apoio à tomada de decisão. 2009. Citado na página 15.

SCHWEFEL, Hans-Paul. **Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik. Dipl.-Ing**. 1965. Tese (Doutorado) — thesis, 1965. Citado na página 16.

SHI, Yuhui; EBERHART, Russell. A modified particle swarm optimizer. In: IEEE. **1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)**. [S.l.], 1998. p. 69–73. Citado na página 19.

SIQUEIRA, Hugo et al. Double-swarm binary particle swarm optimization. In: IEEE. **2018 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.], 2018. p. 1–8. Citado 3 vezes nas páginas 23, 24 e 25.

STÜTZLE, Thomas; DORIGO, Marco et al. Aco algorithms for the traveling salesman problem. **Evolutionary algorithms in engineering and computer science**, p. 163–183, 1999. Citado na página 13.

APÊNDICES

APÊNDICE A – DADOS DOS VALORES DE VOLUME DA MOCHILA, ITENS E VALORES UTILIZADOS PARA AS SIMULAÇÕES DO *KNAPSACK PROBLEM*

Abaixo se encontra cada um dos casos utilizados para o *Knapsack Problem*

A.1 DADOS PARA *KNAPSACK PROBLEM* COM DIMENSÃO $Dim = 30$

- A melhor resposta, isto é, melhor *fitness* para essa mochila, é 729;
- O volume dessa mochila é de 629;
- Valores dos itens dados: [92 29 37 37 77 83 28 83 52 5 106 31 4 89 14 21 44 90 45 78 31 101 5 40 66 11 59 88 73 29];
- Volume dos itens dados: [82 26 42 36 70 77 36 77 55 10 96 23 2 82 14 22 50 69 39 71 44 96 2 51 65 9 50 91 65 25];

A.2 DADOS PARA *KNAPSACK PROBLEM* COM DIMENSÃO $Dim = 100$

- A melhor resposta, isto é, melhor *fitness* para essa mochila, é 12500;
- O volume dessa mochila é de 1159;
- Valores dos itens dados: [51 212 125 177 200 124 122 91 236 111 227 211 234 100 41 198 157 9 11 101 85 172 99 43 140 152 161 30 194 35 146 32 77 138 103 10 230 1 38 247 233 192 62 88 159 154 105 209 250 242 112 205 106 14 173 67 33 90 137 115 249 207 220 39 145 65 78 132 17 110 239 244 238 76 28 150 121 94 2 129 68 63 181 86 34 82 31 183 160 241 184 213 167 55 158 186 23 19 136 114];
- Volume dos itens dados: [16 24 24 6 14 10 9 5 1 22 22 4 17 9 12 4 21 5 24 17 13 2 7 17 9 17 22 7 10 14 11 9 6 3 25 2 7 4 15 7 25 3 18 14 14 19 24 3 24 22 7 2 14 20 10 8 5 12 10 11 10 20 18 3 18 7 23 4 20 18 24 20 14 2 5 20 15 23 22 17 8 5 22 8 16 2 23 10 11 4 8 9 13 11 14 22 3 9 18 16];

A.3 DADOS PARA *KNAPSACK PROBLEM* COM DIMENSÃO $Dim = 250$

- A melhor resposta, isto é, melhor *fitness* para essa mochila, é 9995;

- O volume dessa mochila é de 719;
- Valores dos itens dados: 82 91 13 92 64 10 28 55 96 97 16 98 96 49 81 15 43 92 80 96 66 4 85 94 68 76 75 40 66 18 71 4 28 5 10 83 70 32 96 4 44 39 77 80 19 49 45 65 71 76 28 68 66 17 12 50 96 35 59 23 76 26 51 70 90 96 55 14 15 26 85 26 82 25 93 35 20 26 62 48 36 84 59 55 92 29 76 76 39 57 8 6 54 78 94 13 57 47 2 34 17 80 32 53 17 61 27 66 69 75 46 9 23 92 16 83 54 100 8 45 11 97 1 78 82 87 9 40 26 81 44 92 19 27 15 14 87 58 55 15 86 63 36 52 41 8 24 13 19 24 42 5 91 95 50 49 34 91 37 12 79 39 25 41 10 14 95 96 58 6 24 36 83 2 5 17 65 74 65 46 55 30 75 19 69 19 37 63 79 9 93 78 49 44 45 31 51 52 82 80 65 38 82 54 36 94 88 56 63 59 21 31 48 24 85 20 23 18 23 44 32 93 44 19 91 98 44 12 26 41 60 27 61 72 23 12 30 32 43 51 9 27 81 3 93 74 49 58 24 46];
- Volume dos itens dados: [10 6 6 3 5 7 7 4 4 10 1 9 10 8 1 3 4 7 2 8 2 7 5 8 8 10 9 4 7 2 1 8 6 5 10 7 7 9 9 6 2 3 9 1 5 2 10 8 6 5 1 7 1 1 6 1 9 9 8 2 7 6 10 7 9 5 5 9 1 2 2 4 9 9 1 4 6 5 7 7 3 5 1 10 2 2 4 2 5 4 10 10 1 8 3 5 6 10 5 10 4 8 7 6 7 7 2 2 10 2 1 6 9 7 2 4 5 10 2 9 7 4 2 5 5 2 6 3 4 6 3 3 7 3 9 10 8 4 6 2 10 9 9 3 6 1 5 4 2 2 5 1 6 5 7 7 7 1 1 4 6 7 5 9 8 10 6 4 2 7 8 5 1 3 2 3 5 6 5 9 6 10 7 10 3 7 3 7 7 1 3 3 7 9 4 8 7 1 7 4 10 1 5 5 5 8 4 8 5 1 2 8 5 2 4 7 2 8 3 10 3 8 2 3 1 6 7 6 5 7 7 7 7 10 3 8 3 2 7 5 5 7 8 4 7 5 9 9 3 7];

A.4 DADOS PARA *KNAPSACK PROBLEM* COM DIMENSÃO $Dim = 500$

- A melhor resposta, isto é, melhor *fitness* para essa mochila, é 57514;
- O volume dessa mochila é de 3422;
- Valores dos itens dados: [82 162 129 49 173 154 163 85 188 54 134 39 177 87 182 204 40 58 138 169 171 32 3 87 60 140 175 51 154 176 92 115 186 89 85 235 132 142 164 197 175 205 22 17 208 138 90 226 128 159 59 173 32 78 247 243 49 171 231 189 34 172 91 220 94 157 141 91 33 228 173 176 168 116 83 48 130 39 83 207 81 103 170 206 137 75 82 132 155 103 78 172 83 86 137 13 115 26 239 173 239 20 128 18 102 104 231 47 232 245 84 214 161 84 45 133 228 188 157 240 37 181 141 194 184 227 47 56 108 168 242 244 131 125 162 188 115 144 242 220 126 184 119 229 83 142 226 83 53 76 107 215 108 20 215

120 110 139 131 241 166 141 130 112 239 141 244 166 46 84 87 212 114 77 124 146 77 136 211 248 99 110 27 21 185 59 40 240 250 152 178 95 180 4 139 169 210 236 138 168 23 156 189 70 7 10 100 45 208 180 230 141 88 90 116 209 90 27 112 143 92 34 33 3 191 106 107 46 134 117 232 244 106 93 143 165 157 8 238 250 249 217 189 0 51 6 134 136 89 152 63 86 120 101 191 65 222 54 27 206 119 203 48 121 145 24 171 210 155 192 157 1 44 29 186 31 138 153 144 243 176 99 248 57 224 145 11 4 124 152 224 204 41 7 12 244 130 187 104 108 222 180 202 102 182 133 70 91 185 24 9 122 160 160 18 228 20 92 216 237 48 222 64 15 92 91 67 93 15 236 186 111 56 103 205 173 223 198 249 24 42 232 120 25 63 49 97 174 136 160 87 247 70 223 37 239 106 227 222 216 229 183 86 142 155 9 240 135 22 38 146 124 151 12 131 31 64 35 182 219 111 226 17 141 245 22 15 250 231 221 42 230 181 70 141 112 233 109 122 171 19 21 195 56 125 173 176 68 6 85 52 63 9 187 95 146 218 79 155 235 135 182 127 224 118 49 203 29 8 191 160 186 106 241 104 164 76 223 208 133 3 29 59 155 191 91 54 54 242 52 235 9 233 207 18 135 10 186 189 69 144 92 10 91 250 195 187 16 80 188 65 198 208 198 173 14 161 227 250 40 196 44 82 17 188 138 74 165 9 243 78 165 199 76 114 123 171 151 6 30];

- Volume dos itens dados: [15 12 18 12 4 1 18 4 7 3 19 14 5 4 4 9 16 19 11 19 8 2 19 11 1 18 9 14 11 19 7 19 6 6 1 3 15 11 16 5 3 10 8 5 20 4 9 12 19 10 20 4 3 8 9 15 15 17 1 19 18 5 20 19 3 9 15 19 7 3 17 5 9 14 3 12 3 19 14 4 3 13 13 20 8 3 2 17 1 19 13 3 15 8 7 6 14 18 19 15 15 14 20 1 15 3 9 17 3 12 9 11 9 11 10 15 14 18 7 6 18 17 19 14 1 7 12 12 12 13 8 19 20 8 11 10 15 8 10 14 11 18 6 2 19 20 16 6 9 10 14 11 9 7 16 19 10 6 17 11 16 8 15 8 15 1 3 14 3 17 13 6 18 9 6 4 19 15 4 6 8 18 15 1 5 8 19 3 12 14 4 6 7 14 17 17 4 9 13 3 13 1 7 10 17 20 17 1 15 6 16 16 5 15 10 4 10 7 12 1 12 3 13 17 10 8 15 3 2 10 1 3 7 10 6 6 9 9 13 9 16 9 19 20 12 20 18 19 8 1 20 13 2 9 1 10 11 9 7 4 8 19 17 2 19 15 7 8 13 5 3 14 7 4 3 1 8 15 3 18 17 13 13 15 14 11 9 19 19 5 16 4 6 14 12 12 12 16 20 17 2 14 9 17 6 11 14 1 17 15 16 13 20 17 17 19 12 4 3 16 17 17 15 16 13 12 19 1 5 4 17 12 12 6 7 17 19 16 19 8 7 16 13 20 6 15 1 11 10 15 5 9 15 5 20 20 7 6 15 8 18 19 15 10 5 14 1 18 20 20 15 20 1 4 9 9 10 2 4 1 13 18 9 19 15 19 8 15 19 9 13 18 7 7 2 15 6 4 3 3 15 3 6 9 2 14 9 13 19 10 3 3 3 6 18 19 15 7 3 1 19 6 1 11 20 17 20 13 9 15 9 11 15 4 7 2 18 6 18 3 1 9 9 2 16 15 14 18 15 8 19 8 2 18 11 12

10 17 5 2 20 10 19 7 18 17 19 5 4 14 11 2 7 8 6 12 19 1 17 8 17 4 18 19 18 12 11 15 6 11 17 3 9 5 9 14 7 2 6 1];