# Capítulo 6

# Comparação de Métodos de Computação Evolucionária para o Problema da Mochila Multidimensional

Jonas Krause, Jelson André Cordeiro e Heitor Silvério Lopes*

**Resumo:** O Problema 0-1 de Múltiplas Mochilas (PMM) é um problema de otimização combinatória NP-completo bastante difundido na literatura devido à sua grande aplicabilidade no mundo real. O uso de algoritmos meta-heurísticos é uma prática comum para a resolução do PMM. Este capítulo apresenta os seguintes métodos de computação evolucionária para resolver o PMM: Algoritmos Genéticos (AG), uma versão binária da Evolução Diferencial (EDB), o algoritmo Colônia de Abelhas Artificiais (CAA) e o Algoritmo do Morcego (AM), estes dois últimos discretizados do domínio real para o binário. São apresentados e discutidos os resultados da aplicação destes métodos para algumas instâncias de teste do PMM. Os resultados e a análise estatística mostram que o EDB é o melhor método dentre os analisados.

**Palavras-chave:** Problema da mochila, Algoritmos genéticos, Evolução diferencial, Colônia de abelhas artificiais, Algoritmo do morcego.

***Abstract:*** *The 0-1 Multiple Knapsacks Problem (MKP) is a NP-complete combinatorial optimization problem widely found in the literature due to its applicability to real-world problems. The use of metaheuristic algorithms is a common practice to solve the MKP. This paper presents the following evolutionary computation methods for solving the MKP: Genetic Algorithms (GA), a binary version of Differential Evolution (BDE), Artificial Bee Colony (ABC) and Bat Algorithm (BA), these last two discretized from continuous to binary domains. The results and discussion of the application of these methods for some benchmark instances of the MKP are presented. Results and statistical analysis show that BDE is the best method, when compared with the other algorithms.*

***Keywords:*** *Knapsack problems, Genetic algorithms, Differential evolution, Artificial bee colony, Bat algorithm.*

## 1. Introdução

A otimização de recursos é um dos principais objetivos nas mais diversas áreas da logística, transporte e produção (Chu & Beasley, 1998). A busca de métodos eficientes e rápidos de otimização visa o aumento direto do lucro das empresas e a diminuição do uso de matéria prima. Um problema desta natureza bastante conhecido na literatura recente é o Problema da Mochila (PM). As diversas variantes do PM podem ser facilmente adaptadas para problemas reais, como por exemplo, problemas de corte e de empacotamento (Egeblad & Pisinger, 2009). Assim, este problema tem sido foco de diversas pesquisas utilizando programação inteira (Beasley, 1985) e contínua (Mclay & Jacobson, 2007).

Dependendo das dimensões de uma instância do PM, o número de possíveis combinações cresce exponencialmente e demanda um enorme esforço computacional para testar todas as soluções viáveis. Assim, este problema é considerado NP-completo e, em virtude disto, métodos heurísticos tem sido usados para encontrar soluções suficientemente boas (ou eventualmente ótimas) para o problema. Geralmente associadas à evolução de soluções iniciais aleatórias, tais soluções suficientemente boas são a melhor aproximação possível da otimalidade.

A maioria dos métodos da área de Computação Evolucionária (incluindo aqui, aqueles denominados de Inteligência de Enxames) foram concebidos para a otimização de problemas definidos no espaço contínuo de suas variáveis. Poucos são aqueles criados ou adaptados para trabalhar com espaços discretos, notadamente binários. Baseando-se em um exaustivo estudo anterior (Krause et al., 2013), neste trabalho são descritos métodos para a adaptação destes algoritmos, permitindo que os memos possam ser aplicados ao PM e outros problemas de otimização discreta.

*Autor para contato: hslopes@utfpr.edu.br

Os métodos propostos neste capitulo utilizam Algoritimos Genéticos (AG) com codificação binária, uma versão modificada da Evolução Diferencial adaptada para processamento binário e a discretização de dois métodos de inteligência de enxames: o algoritmo Colônia de Abelhas Artificiais (CAA) e o Algoritmo do Morcego (AM). Estes métodos serão descritos em detalhes nas próximas seções.

## 2. Modelagem do Problema

O Problema da Mochila (PM) é um problema clássico de otimização combinatorial e consiste em organizar $n$ itens com diferentes restrições e lucros dentro de um recipiente limitado (mochila, caixa, contêiner, etc). O objetivo é maximizar o lucro resultante do somatório dos $n$ itens carregados no recipiente, respeitando-se a sua capacidade máxima.

O PM pode ser interpretado de acordo com as dimensões dos $n$ itens (Egeblad & Pisinger, 2009). O PM de uma única dimensão consiste na representação de um único valor para as restrições de cada item. Assim, todos os $n$ itens possuem um valor inteiro como restrição, geralmente associado ao seu custo de transporte. Esta versão uni-dimensional do PM é facilmente encontrada no corte de barras metálicas, canos, etc.

Para os itens com duas dimensões (altura e largura) este problema pode ser interpretado com um problema de corte, onde a mochila é um espaço bi-dimensional. No problema de corte, o objetivo também é maximizar o lucro encaixando o maior número de itens e consequentemente minimizando o espaço não utilizado. Considerando a mochila uma peça de tecido, uma chapa metálica ou de madeira para corte, a otimização desta matéria prima é fundamental para maximizar os lucros e minimizar o material de sobra. O PM de natureza bi-dimensional é encontrado, por exemplo, na indústria têxtil, metalúrgica, gráfica, etc.

O PM também pode ser tri-dimensional, quando utilizado para itens com restrições de altura, largura e profundidade. Assim, cada mochila deve ter as três dimensões máximas bem definidas e cada item é associado a um volume. Outras dimensões podem ser associadas à outras restrições, como a ordem em que cada item é colocado na mochila, itens que podem ser levados em diversas posições diferentes (em pé, deitado ou de lado), itens agrupados, etc. Esta versão do PM é encontrada, por exemplo, no carregamento de contêiners, navios e caminhões, ou, ainda no armazenamento industrial.

As variantes do PM também podem ser associadas à quantidade de cada item $n$ a ser levado (Martello & Toth, 1990). No Problema da Mochila Inteiro (PMI) não há limitação na quantidade de itens a serem levados, podendo o mesmo item ser levado várias vezes ou até encher toda a mochila. Caso o item só possa ser levado uma única vez, este problema passa a ser um problema binário e denominado como Problema 0-1 da Mochila. Esta variante é uma das mais estudadas em problemas de programação discreta, devido ao fato de poder representar uma gama muito grande de situações práticas. O Problema 0-1 da Mochila pode ser visto como um problema de programação inteira e como um subproblema de muitos outros problemas mais complexos. Uma variante mais completa do Problema 0-1 da Mochila considera a existência de múltiplas mochilas com restrições diferentes para cada item. Esta variante é conhecida como o Problema 0-1 de Múltiplas Mochilas, foco deste trabalho.

Algumas variantes apresentam condições adicionais, como por exemplo, a limitação da quantidade de itens a serem levados. Neste caso, o problema passa a ser chamado de Problema da Mochila Restrito. Outra variante busca selecionar um subconjunto de itens cuja soma total dos pesos dos itens escolhidos se aproxime ao máximo da capacidade da mochila. Esta variante é chamada de Problema da Soma de Subconjuntos. Pode-se também adicionar a condição de que o número total de itens selecionados seja o mínimo possível, criando outra variante do problema conhecida como Problema do Troco.

O Problema de Atribuição Generalizada (*Generalized Assignment Problem*) pode ser descrito utilizando a terminologia do Problema da Mochila. O problema consiste em associar cada item a exatamente uma mochila, visando maximizar o ganho total sem associar a nenhuma mochila um peso total que ultrapasse sua capacidade. O Problema *Bin-Packing* é outra variante do PM muito similar ao Problema de Atribuição Generalizada. Cada item é associado a uma mochila, porém o número total de mochilas usadas deve ser o mínimo possível.

Existe ainda uma variante que utiliza uma função objetivo não linear ou que envolve restrições não lineares. Esta variante é chamada de Problema da Mochila Quadrático. Outras variantes podem ser compreendidas como o Problema da Mochila Compartimentada. Neste caso são considerados diferentes compartimentos dentro de cada mochila e cada item é associado a um ou mais compartimentos, criando assim possíveis agrupamentos.

Como mencionado anteriormente, este trabalho tem como foco uma das variantes binárias mais conhecidas do PM, o Problema 0-1 de Múltiplas Mochilas (PMM), também conhecido na literatura como: 0-1 $Multi-KnapsackProblem$ (MKP), $MulticonstraintKnapsackProblem$, $MultipleKnapsackProblem$ ou 0/1 $MultidimensionalKnapsackProblem$ (Khuri et al., 1994).

O PMM possui um único valor associado a cada item $n$, portanto uni-dimensional. Por ter múltiplas mochilas, cada valor associado também depende da mochila $m$, montando um problema com variáveis

multidimensionais ($n \times m$). Krause et al. (2012) definem o problema como o transporte de $n$ itens ou cargas a ser realizado em $m$ mochilas diferentes (ou meios diferentes de transporte). Suas respectivas restrições $W_{ij}, (i = 1, 2, ..., n)$ e $(j = 1, 2, ..., m)$ são representadas por uma matriz com $n$ linhas e $m$ colunas. Cada item $n$ pode ter sua restrição diferenciada dependendo em qual meio de transporte $m$ ele melhor se encaixa. Em outras palavras, cada item $n$ pode ser transportado de diferentes maneiras: avião, barco, *containers*, caminhões, etc. e cada meio de transporte possui uma restrição diferente $W_{ij}$ associada a cada item $n$.

O objetivo do PMM é maximizar o somatório do lucro $P$ que cada item $n$ levado possui, representado pela variável multidimensional $Pi$. Cada item $n$ é representado por uma variável binária $X_i$ que pode assumir o valor de zero (0) ou um (1), representando se o item $n$ será levado (1=SIM) ou não será levado (0=NÃO). A Equação 1 representa matematicamente este objetivo.

$$max(\sum(P_i.X_i)) \qquad i = 1, 2, ..., n \tag{1}$$

Porém, cada mochila $m$ possui uma capacidade máxima $C_j$ e seu limite deve ser respeitado. Consequentemente, o somatório das restrições $W_{ij}$ multiplicado pela variável binária $X_i$ deve ser menor ou igual ao valor de $C_j$ de cada mochila $m$. Tal limite máximo é representado matematicamente na Equação 2.

$$\sum(W_{ij}.X_i) \leq C_j \qquad j = 1, 2, ..., m \tag{2}$$

As variáveis $P_i$, $C_j$ e $W_{ij}$ são definidas como valores inteiros positivos. A forte interação entre as variáveis $W_{ij}$ e $C_j$ restringe os valores máximos de $W_{ij}$ e mínimo de $C_j$. Isto pode ser compreendido como: nenhum item $n$ deve ter a sua restrição $W_{ij}$ maior do que o tamanho de $C_j$ para o mesmo valor de $j$, como mostrado na Equação 3. Esta equação garante que nenhum item pode ter sua restrição $W_{ij}$ maior do que o tamanho total da mochila $j$.

$$W_{ij} \leq max(C_j) \tag{3}$$

A Equação 4 demonstra que toda capacidade $C_j$ deve ser maior que o valor mínimo de $W_{ij}$. Isto pode ser interpretado como a limitação de haver mochilas menores que o menor valor de restrição.

$$C_j \geq min(W_{ij}) \tag{4}$$

A codificação do vetor solução $X_i$ para o PMM pode ser representada por um vetor binário de dimensão $n$ para a utilização dos métodos de computação evolucionária. A melhor combinação binária do vetor solução $X_i$ indicará quais itens devem ser levados para que o maior lucro seja atingido e, consequentemente, a melhor solução para o problema. Tal combinação é uma das possibilidades criadas pelo vasto Espaço de Busca (EB) do problema, definido pelo número $n$ de itens e o número $m$ de mochilas, sendo representado matematicamente pela Equação 5.

$$EB = m.2^n \tag{5}$$

O número de itens $n$ do problema leva a uma equação exponencial de base dois e expoente $n$. Caso a instância do problema utilize uma única mochila, o EB é então definido como $2^n$. O número de mochilas $m$, quando maior do que 1, leva a um EB maior, elevando o grau de complexidade do problema.

## 3. Métodos

A busca da solução exata para problemas combinatoriais NP-completos é uma tarefa que exige um extraordinário esforço computacional, tornando-se inviável o uso de um método determinístico para instâncias com valores grandes de $n$ e $m$. Como alternativa, diversos algoritmos heurísticos podem ser utilizados para este tipo de problema. Tais métodos são, em geral, populacionais. Isto é, trabalham com diversas soluções possíveis ao mesmo tempo e, através de um processo iterativo, podem alcançar soluções de boa qualidade (eventualmente a solução ótima), dentro de um tempo computacional aceitável.

### 3.1 Algoritmo Genético

Algoritmo Genético (AG) é um método heurístico de busca e otimização introduzido por Holland (1975) e desde então vem sendo largamente utilizado para problemas reais de alta complexidade. Baseados na teoria da evolução de Darwin, os AGs buscam convergir uma população inicial aleatória para uma solução de boa qualidade usando conceitos de seleção natural e evolução. Transições probabilísticas atuam sobre os indivíduos

da população por diversas gerações objetivando a sua evolução de acordo com a sua capacidade de adaptação. AGs usam esta evolução como um processo inteligente de busca de soluções de boa qualidade para problemas difíceis.

O AG aplicado no PMM busca encontrar a melhor combinação de $n$ itens a serem levados, respeitando suas restrições do problema. Cada indivíduo da população inicial do AG é representado por um cromossomo de dimensão $n$, onde cada elemento deste vetor solução é uma variável binária representando o $n$-ésimo item.

Utilizando-se de uma população inicial aleatória, o AG avalia como cada indivíduo se adapta ao problema através da função de $fitness$, associada ao objetivo do problema. Esta função fornece um valor considerado a qualidade do indivíduo. A partir disto, os indivíduos de melhor qualidade têm maiores chances de serem selecionados para o processo reprodutivo. A seleção é realizada pelo método do torneio estocástico. A geração de novos indivíduos ocorre através da aplicação de operadores genéticos de *crossover* e mutação aos indivíduos anteriormente selecionados. Os novos indivíduos constituem uma nova população e são avaliados pela função de $fitness$ e o processo se repete por um número pré-definido de gerações.

Dependendo como os indivíduos foram codificados e como as restrições do problema foram tratadas, é possível que indivíduos inválidos sejam gerados ao longo do processo evolutivo. Tais indivíduos podem representar soluções promissoras mas que não satisfazem todas as restrições de valores do problema. Entretanto, é usual permitir indivíduos inválidos na população, penalizando seu valor de $fitness$ proporcionalmente à violação das restrições das mochilas. Isto é feito com a expectativa de que tais indivíduos contenham material genético útil para as próximas gerações. O Algoritmo 1 mostra o pseudocódigo do AG canônico.

**Algoritmo 1** Algoritmo Genético (AG)

```
1:  Parâmetros: D, P, CR, MU
2:  Gerar população inicial x_i ∈ [0, 1]
3:  Avalia f(x_i) para todos os indivíduos
4:  fim = FALSO
5:  while NÃO (fim) do
6:    for P/2 do
7:      Selecione 2 indivíduos da população atual pelo método do Torneio
8:      Aplique crossover com probabilidade CR gerando 2 filhos
9:      Aplique mutação aos filhos com probabilidade MU
10:     Calcule o fitness dos filhos
11:     Coloque os filhos na nova população
12:   end for
13:   Substitua a população antiga pela nova
14:   if condição terminal alcançada then
15:     fim = VERDADEIRO
16:   end if
17: end while
18: Pós-processamento
```

Inicialmente são determinados os parâmetros $D$, $P$, $CR$ e $MU$, representando respectivamente a dimensão $D$ do cromossomo, o número de indivíduos da população $P$, a taxa de *crossover* $CR$ e a taxa de mutação $MU$ (linha 1). Logo após é gerado aleatoriamente uma população de indivíduos (linha 2) e calculada sua função de $fitness$ (linha 3), que representa a adequabilidade do indivíduo para a solução do problema. É definido um bloco repetitivo de comandos (linhas 5-17) para criar e selecionar uma nova população (linhas 7-11) até que seja alcançada uma condição terminal definida. A seleção de dois indivíduos da população atual (linha 7) simula a reprodução gerando dois filhos através do método de *crossover*, que consiste em dividir cada um dos cromossomos pais em duas partes e associá-las entre si gerando novos indivíduos. Aplica-se a mutação (linha 9) que é uma mudança aleatória em um dos elementos do cromossomo. Depois calcula-se o valor de $fitness$ dos indivíduos filhos gerados (linha 11). Estes indivíduos filhos são incorporados à nova população, que substitui a população antiga (linha 13). O Algoritmo 1 finaliza quando sua condição terminal é alcançada. Esta condição pode ser um número pré-determinado de gerações ou quando o melhor indivíduo gerado até então satisfaz um critério de qualidade mínimo.

### 3.2 Evolução Diferencial Binária

O algoritmo de Evolução Diferencial Binária (EDB) é um algoritmo meta-heurístico baseado em populações de vetores proposto por Krause et al. (2012). A EDB é uma variante da Evolução Diferencial (ED) original

(Price et al., 2005) que utiliza o processo de mutação do AG e foi adaptado para problemas com representação binária.

A ED foi introduzida por Storn & Price (1995) para a otimização de funções matemáticas multidimensionais no espaço contínuo. Entretanto, diferentemente de outros métodos, a ED não utiliza o gradiente da função sendo otimizada, permitindo, assim, sua utilização para problemas complexos, não diferenciáveis, ruidosos, etc. Por esta razão, a ED tem sido aplicada com sucesso para uma vasta gama de problemas, como por exemplo, projetos de filtros digitais (Storn, 1996), problemas inversos de transferência radiativa (Lobato et al., 2011), dobramento de proteínas (Kalegari & Lopes, 2010), e otimização de cadeia de suprimentos (Falcone et al., 2008).

Em geral, as variantes da ED são caracterizadas por diferentes processos de mutação e *crossover* da população. A mutação acontece quando um indivíduo é perturbado pela diferença de outros dois indivíduos e pelo processo de mutação. Tal mutação pode ser aleatória, utilizando o melhor indivíduo, ou ainda utilizando as duas técnicas juntas. Por outro lado, o processo de *crossover* acontece de duas maneiras diferentes, binomialmente ou exponencialmente.

A EDB utilizada neste trabalho é baseada na variante mais utilizada da ED: $DE/rand/1/bin$, o que consiste em utilizar uma mutação diferencial aleatória a cada indivíduo da população e um *crossover* binomial.

Sendo a ED original concebida para espaços contínuos (variáveis reais), esta variante do problema teve sua codificação adaptada para o espaço discreto (variáveis binárias). A codificação do indivíduo pode ser interpretada como um vetor binário, analogamente à codificação tradicioinal do AG. Porém, como o processo de mutação da ED utiliza a diferença de dois vetores, na EDB foi adaptada como a troca de um ou mais $bit(s)$ do vetor. Esta modificação cria um algoritmo com uma capacidade maior de busca global, criando soluções não testadas anteriormente e possibilitando uma maior diversidade. Este processo de mutação foi inspirado no AG e adaptado para a sua utilização dentro da ED original. Todo o processo de *crossover* foi mantido como na ED original.

Utilizando uma codificação binária, todos os indivíduos da população inicial gerada aleatoriamente são vetores binários sendo possíveis soluções sendo evoluídas para o problema.

Na EDB, uma população de $N$ vetores, cada um com dimensão $D$, interage entre si. Cada vetor binário $\vec{x} = [x_{i1}, x_{i2}, \ldots, x_{iD}]$ é uma possível solução para o problema, e é avaliado pela função de $fitness$ $f(\vec{x}_i)$ $(i = 1, \ldots, N)$. O pseudocódigo da EDB é mostrado no Algoritmo 2.

**Algoritmo 2** Evolução Diferencial Binária (EDB)

```
1:  Parâmetros: D, N, PM, PR
2:  Gerar população inicial x_i
3:  Calcular função fitness f(x_i) de cada indivíduo
4:  while condição de parada não atingida do
5:    for i = 1 to N do
6:      if rndreal(0,1) < PR ou j = j_rand then {Perturbação de y}
7:        if rndreal(0,1) < PM < PM then {Mutação}
8:          InverterBit(y_j)
9:        end if
10:     end if
11:   end for
12:   Calcular f(y)
13:   if f(y) > f(x_i) then {Atualiza solução corrente}
14:     x_i recebe y
15:   end if
16:   Encontrar a melhor solução corrente x*
17: end while
18: Apresentar resultados
```

O algoritmo tem os seguintes parâmetros: $D$ como número de variáveis do problema (no presente trabalho corresponde ao número de *bits*), $N$ como número de indivíduos (vetores binários) da população, $PM$ como parâmetro de mutação e $PR$ como taxa de perturbação (todos definidos na linha 1).

Inicialmente gera-se uma população aleatória $i$ (linha 2) e calcula-se a função de $fitness$ para cada indivíduo (linha 3). Uma estrutura de repetição é executada até ser atingida a condição de parada (linhas 4-5), esta estrutura consiste em criar novos indivíduos através dos processos de perturbação (mutação e *crossover*). Dentro desta estrutura são selecionados os índices $s$ e $j$ aleatórios, onde $rndint$ e $rndreal$ são números aleatórios inteiros e reais respectivamente. O valor de $s$ corresponde a um número inteiro entre 1 e

o número de indivíduos da população $N$ ou igual do valor corrente de $i$ (linha 6). O valor de $j$ corresponde a um número inteiro entre 1 e a dimensão $D$ (linha 7).

O vetor solução teste $\vec{y}$ recebe o vetor do indivíduo $\vec{x}_i$ (linha 8) que sofrerá uma perturbação em seus $D$ elementos caso um valor aleatório real entre 0 e 1 seja menor do que a taxa de perturbação $PR$, ou o índice $j$ seja igual ao índice $j_{rand}$ (linha 10). Esta perturbação será uma mutação (inversão de *bit*) caso um valor aleatório real entre 0 e 1 seja menor que a taxa de mutação $PM$ (linha 11-12). Caso contrário, é aplicado um *crossover* na solução teste $j$ (linhas 13-14).

O valor de adaptabilidade ou função $fitness$ da solução teste é calculada após as perturbações (linha 18). Caso o valor do $fitness$ do novo indivíduo seja maior que o valor de $fitness$ do indivíduo antigo (linha 19), a solução $i$ recebe a nova solução (linha 20). Deste vetor de soluções é encontrada a melhor solução corrente $\vec{x}$ (linha 23). Esta estrutura é repetida até que a condição de parada seja atingida (linha 24), normalmente definida como um número pré-determinado de iterações do algoritmo. O Algoritmo 2 finaliza mostrando os resultados da melhores soluções $\vec{x}$ (linha 25).

### 3.3 Colônia de Abelhas Artificiais

O algoritmo de Colônia de Abelhas Artificiais (*Artificial Bee Colony* - CAA) é inspirado no comportamento de coleta de alimento pelas abelhas (Karaboga & Akay, 2009). As abelhas têm como objetivo a descoberta de locais que sejam fontes de alimento com quantidade elevada de néctar. A função $fitness$ avalia os possíveis locais encontrados pelas abelhas. Existem três tipos de abelhas: as abelhas escoteiras voam aleatoriamente no espaço de busca sem orientação específica; as abelhas empregadas exploram a vizinhança de sua localização para selecionar uma solução aleatória para ser perturbada; e as abelhas espectadoras que selecionam probabilisticamente uma solução para explorar a sua vizinhança.

Caso a quantidade de néctar de uma nova fonte encontrada seja maior do que a anterior na sua memória, a abelha se desloca para a nova posição e esqueça a anterior. Se uma solução não for melhorada por um número pré-determinado de iterações, a fonte de alimento é abandonada pela correspondente abelha empregada e esta se torna uma abelha escoteira. O algoritmo CAA equilibra exploração (busca global) e intensificação (busca local) usando as abelhas empregadas e espectadoras para a primeira tarefa e as abelhas escoteiras para a segunda.

Originalmente, o CAA foi concebido para trabalhar com variáveis contínuas (Karaboga & Akay, 2009) e a sua aplicabilidade tem sido muito ampla, principalmente nas áreas de engenharia e computação, por exemplo: na determinação da estrutura de proteínas (Benítez et al., 2012), no planejamento da recarga de reatores nucleares (Lima et al., 2011), e no reconhecimento de imagens faciais (Chidambaram & Lopes, 2010). O Algoritmo 3 apresenta o pseudocódigo do CAA.

**Algoritmo 3** Colônia Artificial de Abelhas (CAA)

1: Parâmetros: $CS, Limite, MCN, D$
2: Inicializa abelhas $\vec{x}_i$ ($i = 1, \ldots,$ CS/2)
3: **while** g < MCN **do**
4: Gera modificações $v_i^g$; Se melhorou $x_i^g = v_i^g$ e $Limite = 0$, senão, $Limite = Limite + 1$
5: Calcula probabilidades $p_i^g$; Se $p_i^g > rand(0,1)$ recruta seguidora
6: Seleção probabilística das $x_i^g$ soluções mais promissoras
7: Gera modificações $v_i^g$; Se melhorou $x_i^g = v_i^g$ e $Limite = 0$, senão, $Limite = Limite + 1$
8: Substitui soluções em que $Limite$ atingiu a contagem e faz $Limite = 0$
9: Memoriza melhor solução encontrada até o momento
10: g = g + 1
11: **end while**
12: Pós-processamento

O algoritmo começa definindo os parâmetros: população da colônia ($CS$), quantidade de variáveis ($D$), número máximo de gerações ($MCN$) e o limite para abandono de uma fonte de alimento ($Limite$) (linha 1). A população inicial é criada e dividida igualmente entre abelhas escoteiras e empregadas (linha 2). Uma estrutura de repetição (linhas 3 a 11) é utilizada com um contador ($g$) até o número máximo de ciclos ($MCN$). Esta estrutura gera modificações (linha 4) buscando melhores soluções, calcula a probabilidade da abelha se tornar uma seguidora (linha 5) e seleciona as soluções mais promissoras (linha 6). Gera novamente modificações (linha 7), substitui soluções que atingiram o limite (linha 8), memoriza a melhor solução encontrada (linha 9) e atualiza o contador (linha 10).

Utilizando o pseudocódigo do CAA para espaços contínuos, este artigo utiliza uma das técnicas de discretização proposta em Krause et al. (2013) para converter o vetor solução de real para binário. Uma

função sigmóide foi utilizada para converter cada dimensão do indivíduo, transformando ele em um indivíduo binário. Assim o algoritmo continua processando e posicionando as abelhas com valores reais, porém a fonte de alimento (objetivo do problema) discretizada para binário passa a ser uma possível solução para o problema.

### 3.4 Algoritmo do Morcego

O algoritmo do morcego (*BatAlgorithm* – AM) é inspirado no processo de eco-localização desempenhado pelos morcegos durante o seu vôo para detectar presas e evitar obstáculos e foi criado por Yang (2010b). A eco-localização se baseia na emissão de ondas ultrassônicas e a correspondente medição do tempo gasto para estas ondas voltarem à fonte após serem refletidas pelo alvo (presa ou obstáculo). A taxa de pulso e amplitude dos sons emitidos pelos morcegos variam com a estratégia de caça. Quando identificada uma presa, a taxa de pulso ($r$) é acelerada e a amplitude ($A$) é aumentada para evitar a perda da presa. Por outro lado, quando a presa está sob domínio, a amplitude diminui.

No modelo computacional do AM, cada morcego representa uma possível solução para o problema codificado sob a forma de um vetor. Uma população de morcegos então se move no espaço de busca do problema, atualizando continuamente a frequência, velocidade e posição de cada elemento buscando encontrar a solução ótima. A cada nova interação, cada morcego é atualizado seguindo a melhor solução encontrada pela população. Além da atualização da posição, existe o controle de exploração e intensificação como nos outros algoritmos de computação evolucionária. A exploração e a intensificação são realizadas, respectivamente, pela variação da amplitude e da taxa de pulso.

Embora o criador do algoritmo disponibilize uma versão do seu pseudocódigo (Yang, 2010b,a), muitos detalhes para a sua implementação são omitidos. Posteriomente, Cordeiro et al. (2012) publicaram uma versão mais completa do pseudocódigo, que é apresentado a seguir no Algoritmo 4.

**Algoritmo 4** Algoritmo do morcego (AM)

1: Parâmetros: $n, \alpha, \lambda$
2: Inicializa morcegos $\vec{x}_i$
3: Avalia $f(\vec{x}_i)$ para todos os morcegos
4: Atualiza melhor morcego $\vec{x}_*$
5: **while** critério de parada não atingido **do**
6:  **for** $i = 1$ **to** $n$ **do**
7:  $f_i = f_{min} + (f_{max} - f_{min})\beta, \beta \in [0, 1]$
8:  $\vec{v_i}^{t+1} = \vec{v_i}^{t} + (\vec{x_i}^{t} - \vec{x_*}^{t})f_i$
9:  $\vec{x}_{temp} = \vec{x_i}^{t} + \vec{v_i}^{t+1}$
10:  **if** $rand < r_i$ , $rand \in [0, 1]$ **then** {Faz busca local}
11:  $\vec{x}_{temp} = \vec{x}_* + \epsilon A_m, \epsilon \in [-1, 1]$
12:  **end if**
13:  Realiza perturbação em uma dimensão de $\vec{x}_{temp}$
14:  **if** $rand < A_i$ **or** $f(\vec{x}_{temp}) \leq f(\vec{x}_i)$ , $rand \in [0, 1]$ **then** {Aceita solução temporária}
15:  $\vec{x}_i = \vec{x}_{temp}$
16:  ${r_i}^{t+1} = 1 - exp(-\lambda t)$
17:  ${A_i}^{t+1} = \alpha {A_i}^{t}$
18:  **end if**
19:  Atualiza melhor morcego $\vec{x}_*$
20:  **end for**
21: **end while**
22: Pós-processamento

O AM inica no instante $t = 0$ e todos os $n$ morcegos $\vec{x}_i$ $(i = 1, \ldots, n)$ são inicializados com taxa de pulso $r_i = 0$, velocidade $\vec{v}_i = 0$, amplitude $A_i = 1$, frequência $f_i = 0$ e posição $\vec{x}_i$ aleatória (linha 2). O ciclo principal representa a evolução da população no tempo (linhas 5-21). O primeiro passo no interior do ciclo é atualizar a posição temporária $\vec{x}_{temp}$ até ser aceita. Para isto, a frequência $f_i$ é atualizada (linha 7), onde $f_{min}$ e $f_{max}$ são, respectivamente, os limites inferiores e superiores da função de $fitness$. O valor de $\beta \in [0, 1]$ é um número aleatório extraído de uma distribuição uniforme. A nova frequência $f_i$ é utilizada para determinar a nova velocidade $\vec{v_i}^{t+1}$ (linha 8), onde $\vec{x}_*$ é a melhor solução encontrada até o instante $t$. Com a nova velocidade $\vec{v_i}^{t+1}$, é possível determinar a nova posição temporária $\vec{x}_{temp}$ (linha 9).

Na linha 10 é realizada a busca local, que pode ser implementada de diversas maneiras. Um passeio aleatório (*random walk*) pode ser usado tanto para exploração quanto intensificação, dependendo do tamanho do passo. Outra maneira é utilizar o operador de mutação não uniforme. Nesta implementação foi utilizado

um terceiro método sugerido por Yang (2010b) (linha 11), onde $\epsilon \in [-1, 1]$ é um número aleatório extraído de uma distribuição uniforme e $A_m$ a média da amplitude de todos os morcegos em um dado instante $t$. Ainda na linha 11, o valor de $\vec{x}_{temp}$ é atualizado pela busca local, desconsiderando o valor anterior da posição e velocidade. Na linha 13 uma das dimensões de $\vec{x}_{temp}$ (um dos elementos deste vetor), escolhida dentre $d$ dimensões é modificada aleatoriamente dentro dos limites da função de avaliação.

Se a condição (linha 14) for verdadeira e $rand \in [0, 1]$ um número aleatório extraído de uma distribuição uniforme, a solução temporária $\vec{x}_{temp}$ é aceita (linha 15) e ocorre o aumento da taxa de pulso (linha 16). Caso $t \to \infty$ e $r_i \to 1$, a busca local se intensifica com o passar do tempo. Outro valor atualizado é a amplitude $A$ (linha 17). Para controlar a diminuição gradual de $A$, dois métodos (linear e geométrico) foram propostos por Yang (2010a). Para o método linear, a equação é $A = A_0 - \beta t$, onde $A_0$ é a amplitude inicial, $t$ é o número da interação e $\beta$ é taxa de diminuição, tal que $\beta = (A_0 - A_f)/t_f$, sendo $A_f$ a amplitude final e $t_f$ o número máximo de interações. Assim, $A \to 0$ quando $t \to \infty$. Para o método geométrico $A$ diminui com uma taxa de diminuição $0 < \alpha < 1$, utilizando a equação $A = A_0\alpha^t$, $t = 1, 2, ..., t_f$.

Nesta implementação do AM foi utilizado o segundo método para a diminuição gradual de $A$, pois tem a vantagem de não precisar especificar o número máximo de interações. É necessário apenas determinar o valor de $\alpha$ e o valor inicial de $A$. Foi observado que quando a diminuição for suficientemente lenta, o valor encontrado ao final da execução do algoritmo tende a se aproximar do ótimo global (Yang, 2010a).

O ciclo principal continua até que a sucessão evolutiva da população de morcegos atinja o critério de parada estabelecido (linha 5), geralmente um número máximo de iterações. Os valores de $\alpha$ e $\lambda$ são os parâmetros do AM que serão analisados na Seção 4.3.

Assim como o CAA, o AM também foi discretizado utilizando uma função sigmóide. Isto mantêm os morcegos buscando sua presa em espaços contínuos, porém a solução encontrada é discretizada para o espaço discreto, tornando válida a possível solução.

## 4. Experimentos

Este capítulo utiliza implementações em ANSI C para todos os métodos propostos. Para o AG, foi utilizado o software de domínio público GALLOPS [1](*Genetic ALgorithm Optimized for Portability and Parallelism System*), versão 3.2.4. A EDB utilizou a mesma implementação original proposta em Krause et al. (2012), de acordo com o pseudocódigo do Algoritmo 2. Para o CAA foi utilizado o código implementado por Karaboga & Akay (2009)[2] e descrito no Algoritmo 3. O AM, introduzido por Yang (2010b) e adaptado por Cordeiro et al. (2012), utiliza o pseudocódigo do Algoritmo 4.

### 4.1 Codificação e Discretização

Por ser um algoritmo que trabalha com codificação discreta, o AG pode representar as possíveis soluções do problema utilizando valores inteiros ou binários. Portanto, a implementação deste algoritmo para o PMM não necessitou de adaptações. Seu vetor solução (cromossomo) foi codificado em binário para representar quais itens devem ser levados na mochila. Cada posição do vetor $X_i$ representa a presença do item ($X_i = 1$) ou não ($X_i = 0$).

A EDB é uma adaptação do algoritmo original de ED que foi concebido para otimização em espaços contínuos. Por utilizar valores reais contínuos, a ED teve que ser modificada para trabalhar com indivíduos binários. Portanto, a EDB busca a evolução da sua população em espaços discretos. A mudança de espaço contínuo para discreto tem por consequência um algoritmo adaptado. Assim, o vetor solução só aceita valores binários e o processo de mutação foi modificado para o mesmo processo do AG, invertendo um ou mais bit(s) do vetor solução. A alteração da codificação do algoritmo é uma das técnicas para o uso de algoritmos contínuos em problemas discretos. O uso desta técnica geralmente resulta em versões discretas do algoritmo original, como é o caso da EDB.

Uma técnica bastante utilizada é a discretização somente do vetor solução (Krause et al., 2013). Assim o algoritmo continua trabalhando em espaços contínuos, mas cada solução é convertida para valores discretos. Este foi o caso tanto do CAA quanto do AM utilizados neste capítulo. Em ambos os casos uma função sigmoidal foi utilizada para transformar cada elemento do vetor de números reais para valores binários $0, 1$. Com esta estratégia, ambos os algoritmos conservam suas características originais, com processamento no espaço contínuo, mas buscando soluções no espaço discreto.

[1] http://garage.cse.msu.edu/software/galopps/
[2] http://mf.erciyes.edu.tr/abc/

### 4.2 *Benchmarks*

*Benchmarks* são instâncias de teste do problema para as quais se conhece o valor ótimo. Utilizando estes *benchmarks*, cada algoritmo é testado e a relevância de seus resultados se dá pela proximidade da solução encontrada com o valor ótimo conhecido.

As diversas instâncias disponíveis na literatura oferecem uma grande variedade de testes. Cada *benchmarks*tem seu espaço de busca como função do úmero de itens $n$ e mochilas $m$ (veja Equação 5). Algumas instâncias variam também as restrições $W_{ij}$ e a capacidade das mochilas $C_j$, criando problemas com diferentes níveis de complexidade.

A Tabela 1 lista os *benchmarks* utilizados para testes dos algoritmos. Eles foram escolhidos por configurarem diferentes espaços de busca, como função de variados itens $n$ e mochilas $m$. Assim, os métodos utilizados serão testados em instâncias com diferentes níveis de complexidade. O valor ótimo conhecido também é apresentado, tais valores são o lucro máximo possível conhecido para cada instância. Todos os *benchmarks* utilizados foram criados por Freville & Plateau (1990) e estão disponíveis na biblioteca digital do Jornal da Sociedade de Pesquisa Operacional [3] (Beasley, 1990).

Tabela 1. *Benchmarks* para teste dos algoritmos (Freville & Plateau, 1990).

| **Instância** | $m$ | $n$ | **EB** | **Ótimo** |
|---|---|---|---|---|
| PB1 | 4 | 27 | $4 \times 2^{27}$ | 3090 |
| PB2 | 4 | 34 | $4 \times 2^{34}$ | 3186 |
| PB4 | 2 | 29 | $2 \times 2^{29}$ | 95168 |
| PB5 | 10 | 20 | $10 \times 2^{20}$ | 2139 |
| PB6 | 30 | 40 | $30 \times 2^{40}$ | 776 |
| PB7 | 30 | 37 | $30 \times 2^{37}$ | 1035 |

### 4.3 Parâmetros de Controle dos Algoritmos

Cada um dos algoritmos meta-heurísticos apresentados na Seção 3 possui diversos parâmetros que controlam, por exemplo, a quantidade de indivíduos, o critério de parada e as probabilidades de modificação dos indivíduos. O comportamento de cada algoritmo pode ser fortemente influenciado pela escolha destes parâmetros. Os parâmetros utilizados nesta comparação são aqueles usuais na literatura (Goldberg, 1989; Krause et al., 2012; Karaboga & Akay, 2009; Yang, 2010b). É possível que um ajuste mais refinado dos parâmetros possa gerar resultados melhores, no entanto, isto está fora do escopo deste trabalho.

Com o intuito de fazer uma comparação o mais justa possível, os parâmetros similares dos quatro métodos foram identicamente configurados. Todos os métodos apresentaram um total de 30 mil avaliações (100 indivíduos x 300 gerações). Isto faz com que o esforço computacional seja o mesmo para todos. O AG e a EDB utilizam a mesma probabilidade de mutação de 5%. Os demais parâmetros são individuais de cada método. A Tabela 2 lista os parâmetros configurados. A quantidade de indivíduos, vetores, abelhas e morcegos que constitui a população de cada método e a quantidade de gerações/iterações. As taxas de mutação, *crossover* e perturbação, quando aplicáveis, e os valores de $\alpha$ e $\lambda$ específicos do Algoritmo do Morcego.

Tabela 2. Parâmetros de controle dos algoritmos.

| **Parâmetro** | **AG** | **EDB** | **CAA** | **AM** |
|---|---|---|---|---|
| População | 100 | 100 | 100 | 100 |
| Gerações | 300 | 300 | 300 | 300 |
| Mutação | 0,05 | 0,05 | - | - |
| *Crossover* | 0,8 | - | - | - |
| Perturbação | - | 0,5 | - | - |
| $\alpha$ | - | - | - | 0,9 |
| $\lambda$ | - | - | - | 0,1 |

## 5. Resultados

A Tabela 3 apresenta os resultados para os *benchmarks*. Por serem métodos heurísticos com valores iniciais aleatórios, os resultados finais podem variar de acordo com a semente aleatória usada para gerar os primeiros indivíduos. Ou seja, os métodos apresentados podem gerar soluções diferentes a cada execução. Isto cria a necessidade de se avaliar o método proposto várias vezes, e não somente em uma execução. Assim, para cada um dos métodos foram calculados os valores de média e desvio-padrão (DP) de uma amostra de 100 soluções

[3] http://people.brunel.ac.uk/ mastjjb/jeb/orlib/files/mknap2.txt

(melhor resultado de cada execução independente com sementes aleatórias diferentes). A média e o DP são dados importantes e mostram a robustez de cada algoritmo.

Na tabela é apresentada a melhor solução encontrada (Melhor) entre as 100 rodadas de cada método. Esta solução é mostrada em negrito caso a melhor solução seja o valor ótimo conhecido do problema, identificando que o algoritmo conseguiu encontrar a melhor combinação de itens e mochilas daquela instância. Apresenta-se, também, a taxa de sucesso (Sucesso), calculada como a porcentagem de vezes que um método é capaz de chegar à solução ótima conhecida.

Tabela 3. Resultados encontrados pelo AG, EDB, CAA e AM.

| | **AG** | | | **EDB** | | |
|---|---|---|---|---|---|---|
| Instância | Média±DP | Melhor | Sucesso | Média±DP | Melhor | Sucesso |
| PB1 | 3036,91±27,23 | **3090** | 6,00% | 3075,79±12,97 | **3090** | 98,00% |
| PB2 | 3150,82±32,55 | **3186** | 23,00% | 3183,76±8,41 | **3186** | 100,00% |
| PB4 | 91711,67±1421,59 | **95168** | 11,00% | 94702,39±926,28 | **95168** | 48,00% |
| PB5 | 2097,60±24,59 | **2139** | 8,00% | 2132,88±8,16 | **2139** | 100,00% |
| PB6 | 723,81±17,44 | 765 | 0,00% | 767,33±13,08 | **776** | 45,00% |
| PB7 | 965,84±21,64 | 1000 | 0,00% | 1033,01±2,73 | **1035** | 98,00% |
| | **CAA** | | | **AM** | | |
| Instância | Média±DP | Melhor | Sucesso | Média±DP | Melhor | Sucesso |
| PB1 | 2982,26±26,05 | 3026 | 0,00% | 3029,72±34,50 | **3090** | 4,00% |
| PB2 | 3054,18±31,81 | 3148,00 | 0,00% | 3109,89±45,49 | **3186** | 2,00% |
| PB4 | 84941,28±2040,17 | 89432 | 0,00% | 90828,86±2320,05 | **95168** | 9,00% |
| PB5 | 2093,73±19,30 | **2139** | 3,00% | 2086,95±24,00 | **2139** | 4,00% |
| PB6 | 598,81±32,48 | 672,00 | 0,00% | 712,94±41,71 | **776** | 2,00% |
| PB7 | 887,87±29,83 | 976 | 0,00% | 976,55±36,07 | **1035** | 1,00% |

Utilizando as informações do EB (Tabela 1) e os resultados encontrados, pode-se analisar o desempenho de cada algoritmo. Para a instância PB5, por exemplo, todos os algoritmos foram capazes de encontrar o valor ótimo. Devido às restrições a cada mochila e o menor EB dos *benchmarks* analisados, esta instância é a que apresenta a menor complexidade. De maneira oposta, a instância PB6 é a que tem o maior EB e para qual, de maneira geral, os algoritmos tiveram o pior desempenho.

O AG apresentou resultados muito bons, encontrando o valor ótimo para quase todos os *benchmarks* propostos (exceto PB6 e PB7). No entanto, a repetibilidade do método, observada pela taxa de sucesso, ainda não é satisfatória, especialmente para as instâncias mais complexas. Por outro lado, o EDB apresentou-se como o método mais eficiente, não só encontrando os valores ótimos para todas as instâncias, como também as maiores taxas de sucesso. Isto sugere que é um algoritmo robusto com uma boa escalabilidade para problemas combinatoriais com grandes EB. Possivelmente, a eficiência do EDB se deve à sua codificação binária, as estratégias de mutação e *crossover*.

A discretização do vetor solução dos dois algoritmos contínuos (CAA e AM) adapta os métodos para problemas binários e mantêm suas principais características individuais, o que pode ser bastante vangajoso. Porém, ao analisar os resultados do CAA, observa-se um desempenho muito ruim, com uma taxa de sucesso nula, exceto para PB5. Talvez um ajuste refinado dos parâmetros do algoritmo, incluindo o aumento do número de gerações, poderia torná-lo um pouco mais eficiente. Os resultados encontrados pelo AM indicam que o algoritmo encontrou o ótimo para todas as instâncias e com uma repetibilidade melhor do que a do CAA, porém muito aquém daquela do EDB. No entanto, os resultados sugerem que o AM seja um método promissor. De maneira geral, os métodos AG, CAA e AM necessitam de ajustes nos seus parâmetros, de modo a ter um melhor equilíbrio entre busca global e local. Além disto, certamente estes métodos requerem uma quantidade maior de avaliações para obter melhores resultados, pelo menos para os *benchmarks* utilizados.

Para testar a relevância dos dados encontrados, dois testes estatísticos foram aplicados. O teste *Shapiro-Wilk* para determinar se a distribuição observada se aproxima de uma distribuição normal e o teste *Kruskal-Wallis* para comparar mais de duas amostras que são independentes ou não relacionadas, para o caso de amostras que não tenham distribuição normal.

O teste *Shapiro-Wilk* testa a hipótese de que a amostra veio de uma população com distribuição normal, verificando se os dados são normais ou ordinais. Neste teste todos os resultados encontrados de cada método em cada instância rejeitam esta hipótese, portanto não é assumida uma distribuição normal em nenhum dos resultados encontrados. Sendo assim, as distribuições observadas possuem dados do tipo ordinal.

O teste *Kruskal-Wallis* é um método não-paramétrico que analisa a variância dos dados por *ranking*, este é um método não-paramétrico para testar se os dados ordinais são provenientes de uma mesma distribuição. Estabelecendo o nível de significância em 5%, o teste *Kruskal-Wallis* dos resultados encontrados rejeita a

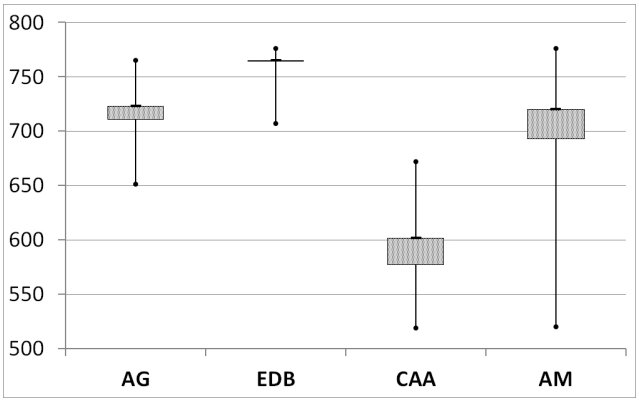


Figura 1. Boxplot dos resultados de cada método para a instância PB6.

hipótese dos valores serem provenientes da mesma distribuição populacional. Tais resultados confirmam que pelo menos uma das amostras são diferentes das outras e, consequentemente, os resultados encontrados podem ser considerados significativos.

Para corroborar a análise estatística, o gráfico *boxplot* da Figura 1 compara os resultados obtidos. As extremidades dos intervalos neste gráfico demonstram o máximo e mínimo encontrado por cada método. Os retângulos horizontais representam os primeiros e segundos quartis das amostras analisadas. Tais quartis não se sobrepõem quando se compara o EDB com os demais algoritmos, significando que o mesmo é, comparativamente o melhor método, sendo diferente dos demais.

## 6. Conclusões

A busca de novos métodos de solução para problemas NP-completos tem sido o foco de estudos de pesquisadores no mundo inteiro. As aplicações do PMM em problemas do mundo real criam a necessidade de algoritmos cada vez mais rápidos e eficientes. Os resultados encontrados apontam as meta-heurísticas como alternativas eficazes e interessantes para este problema. O método de discretização para os algoritmos com representação contínuua apresentou-se eficiente, criando a possibilidade do uso de outros algoritmos de mesma natureza na solução do PMM.

Os algoritmos apresentados neste capítulo buscam melhores alternativas para problemas binários. A comparação entre os métodos concebidos para espaços binários, a adaptação da codificação e a discretização do vetor solução buscam compreender as limitações de cada método e seu comportamento em espaços diferentes. O AG e sua codificação binária apresentaram resultados satisfatórios em quase todas as instâncias, apesar de não ter sido adaptado e ter utilizado os parâmetros sugeridos na literatura. A EDB também alcançou o valor ótimo na maioria das instâncias e mostrou ser um método promissor para problemas binários, principalmente por ter atingido uma alta taxa de sucesso, de maneira geral. O CAA discretizado não alcançou bom desempenho, necessitando de outras adaptações e possíveis ajustes de parâmetros. O AM encontrou a solução ótima para todas as instâncias, graças à sua forte busca global, mas sua repetibilidade ainda é inferior à EDB, sendo, então, um método bastante promissor caso seus parâmetros de controle sejam melhor ajustados.

O uso de outros métodos de discretização e a codificação de outros algoritmos projetados para espaços contínuos serão foco de trabalhos futuros. Com o sucesso do AM discretizado, o uso de processamento em espaço contínuo para gerar soluções discretizadas para problemas combinatoriais passa a ser uma diretriz de estudos. A hibridização dos métodos também é uma tendência em métodos heurísticos, gerando, assim, outras vertentes para trabalhos futuros.

## Referências

Beasley, J.E., Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, 36:297–306, 1985.

Beasley, J.E., OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

Benítez, C.V.; Parpinelli, R.S. & Lopes, H.S., Parallelism, hybridism and coevolution in a multi-level ABC-GA approach for the protein structure prediction problem. *Concurrency and Computation*, 24(6):635–646, 2012.

Chidambaram, C. & Lopes, H.S., An improved artificial bee colony algorithm for the object recognition problem in complex digital images using template matching. *International Journal of Natural Computing Research*, 1(2):54–70, 2010.

Chu, P.C. & Beasley, J.E., A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, 1998.

Cordeiro, J.A.; Parpinelli, R. & Lopes, H., Análise de sensibilidade dos parametros do *Bat Algorithm* e comparação de desempenho. In: *Anais do IX Encontro Nacional de Inteligência Artificial – ENIA*. Curitiba, PR: SBC, 2012.

Egeblad, J. & Pisinger, D., Heuristic approaches for the two- and three-dimensional knapsack packing problem. *Computers in Operations Research*, 36(4):1026–1049, 2009.

Falcone, M.A.; Lopes, H.S. & Coelho, L.S., Supply chain optimisation using evolutionary algorithms. *International Journal of Computer Applications in Technology*, 31(3/4):158–167, 2008.

Freville, A. & Plateau, G., Hard 0-1 multiknapsack test problems for size reduction methods. *Investigación Operativa*, 1:251–270, 1990.

Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, USA: Addison-Wesley, 1989.

Holland, J.H., *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. Ann Arbor, USA: University of Michigan Press, 1975.

Kalegari, D.H. & Lopes, H.S., A differential evolution approach for protein structure optimisation using a 2D off-lattice model. *International Journal of Bio-Inspired Computation*, 2(3/4):242–250, 2010.

Karaboga, D. & Akay, B., A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation*, 214:108–132, 2009.

Khuri, S.; Back, T. & Heitkotter, J., The zero/one multiple knapsack problem and genetic algorithms. In: *Proceedings of the 1994 ACM Symposium on Applied Computing*. ACM Press, p. 188–193, 1994.

Krause, J.; Cordeiro, J.A.; Parpinelli, R.S. & Lopes, H.S., A survey of swarm algorithms applied to discrete optimization problems. In: Yang, X.S.; Cui, Z.; Xiao, R. & Gandomi, A.H. (Eds.), *Swarm Intelligence and Bio-Inspired Computation*. New York, USA: Elsevier, 2013, a ser publicado.

Krause, J.; Parpinelli, R. & Lopes, H., Proposta de um algoritmo inspirado em evolução diferencial aplicado ao problema multidimensional da mochila. In: *Anais do IX Encontro Nacional de Inteligência Artificial – ENIA*. Curitiba, PR: SBC, 2012.

Lima, A.M.M.; Nicolau, A.S.; Oliveira, I.M.S.; Medeiros, J.A.C.C.; Silva, M.H. & Schirru, R., Computação evolucionária aplicada ao problema da recarga de reatores nucleares. In: Lopes, H.S. & Takahashi, R.H.C. (Eds.), *Computação Evolucionária em Problemas de Engenharia*. Curitiba, PR: Omnipax, p. 147–171, 2011.

Lobato, F.S.; Steffen Jr., V. & Silva Neto, A.J., Resolução de problemas inversos em processos difusivos e transferência radiativa usando o algoritmo de evolução diferencial. In: Lopes, H.S. & Takahashi, R.H.C. (Eds.), *Computação Evolucionária em Problemas de Engenharia*. Curitiba, PR: Omnipax, p. 173–195, 2011.

Martello, S. & Toth, P., *Knapsack problems: algorithms and computer implementations*. New York, USA: John Wiley & Sons, 1990.

Mclay, L.A. & Jacobson, S.H., Algorithms for the bounded set-up knapsack problem. *Discrete Optimization*, 4(2):206–212, 2007.

Price, K.; Storn, R.M. & Lampinen, J.A., *Differential Evolution: A Practical Approach to Global Optimization*. Natural Computing Series. New York, USA: Springer-Verlag, 2005.

Storn, R., Differential evolution design of an IIR-filter. In: *Proceedings of IEEE International Conference on Evolutionary Computation*. p. 268–273, 1996.

Storn, R. & Price, K., *Differential Evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces*. Technical Report TR-95-012, International Computer Science Institute, Berkeley University, Berkeley, USA, 1995.

Yang, X.S., *Nature-Inspired Metaheuristic Algorithms*. 2a edição. Frome, UK: Luniver Press, 2010a.

Yang, X.S., A new metaheuristic bat-inspired algorithm. In: Gonzalez, J.R. (Ed.), *Nature Inspired Cooperative Strategies for Optimization*. Berlin, Germany: Springer-Verlag, v. 284 de *Studies in Computational Intelligence*, p. 65–74, 2010b.

## Notas Biográficas

**Jonas Krause** é graduado em Matemática (Universidade Federal do Paraná, 2005) e especialista em Tecnologia da Informação (IBPEX, 2007). Atualmente é mestrando do Programa de Pós-Graduação em Engenharia e Informática (CPGEI) da Universidade Tecnológica Federal do Paraná (UTFPR).

**Heitor Silvério Lopes** é graduado em Engenharia Industrial Eletrônica e mestre em Engenharia Biomédica (Universidade Tecnológica Federal do Paraná – UTFPR, 1984 e 1990, respectivamente), e doutor em Engenharia Elétrica (Universidade Federal de Santa Catarina, 1996). Desde 2003 é bolsista de produtividade em pesquisa do CNPq na área de Ciência da Computação, e até o momento concluiu a orientação de 33 mestrandos e 6 doutorandos. Tem interesse em computação evolucionária e métodos bioinspirados com aplicações em problemas de engenharia, otimização, visão computacional e bioinformática, bem como computação de alto desempenho. Atualmente é Professor Associado IV da UTFPR, no câmpus de Curitiba.