Proposta de um algoritmo inspirado em Evolução Diferencial aplicado ao Problema Multidimensional da Mochila

Jonas Krause[1], Rafael S. Parpinelli[1,2], Heitor S. Lopes[1]

[1]Laboratório de Bioinformática – Universidade Tecnológica Federal do Paraná (UTFPR) Curitiba – PR – Brasil

[2]Departamento de Ciência da Computação – Universidade do Estado de Santa Catarina (UDESC) Joinville – SC – Brasil

jkfries@gmail.com, parpinelli@joinville.udesc.br, hslopes@utfpr.edu.br

Abstract. *The multidimensional knapsack problem is a classical optimization problem which has several discrete and stochastic mathematical methods to resolve it. This paper proposes a new algorithm inspired in Differential Evolution, the Binary Differential Evolution (BDE) and compares its results with those obtained using Genetic Algorithms (GAs). Each method was used in six known instances of the problem; the results found by both methods suggest the BDE as a promising, efficient and fast method for binary problems.*

Resumo. *O problema multidimensional da mochila é um problema clássico de otimização para o qual há vários métodos matemáticos discretos e estocásticos para a sua resolução. Este artigo propõe um novo algoritmo inspirado em Evolução Diferencial, a Evolução Diferencial Binária (EDB) e compara seus resultados com os resultados alcançados utilizando Algoritmos Genéticos (AGs). Cada método foi utilizado em seis instâncias conhecidas do problema, os resultados encontrados sugerem o EDB como um método promissor, eficiente e rápido para problemas binários.*

1. Introdução

O Problema da Mochila (PM) é um clássico problema de otimização combinatória que consiste em organizar *n* itens de diferentes pesos e lucros dentro de uma mochila, objetivando maximizar o lucro resultante da somatória dos *n* itens carregados e respeitar a capacidade máxima da mochila. [Martello e Toth 1990] abordam as diversas variantes do PM, desenvolvendo soluções e experimentos computacionais. Este artigo aborda uma das variantes mais aplicáveis e conhecidas do PM [Beasley e Chu 1998], o Problema Multidimensional da Mochila (PMM) ou *0-1 Multidimensional Knapsack Problem (MKP),* também referenciado na literatura como *Multiconstraint Knapsack Problem*, *Multi-Knapsack Problem*, *Multiple Knapsack Problem* ou *0/1 Multidimensional Knapsack Problem* [Khuri, Bäck, e Heitkoetter 1994].

O PMM é encontrado em diversos segmentos da indústria, tais como logística, transporte e produção [Beasley e Chu 1998]. Este problema está sendo estudado amplamente com o apoio da matemática e da computação, por ser um problema NP-completo não possui um algoritmo capaz de resolvê-lo em tempo polinomial. O uso de Algoritmos Genéticos tem sido uma alternativa viável para problemas deste tipo, como

abordado por [Hristakeva e Shrestha (2011)], [Taskiran (2010)], [Bortfeldt e Winter (2008)] e [Khuri, Bäck e Heitkoetter 1994].

Este artigo apresenta um novo método heurístico para a solução do PMM, chamado EDB – Evolução Diferencial Binária. O EDB é aplicado a várias instâncias do PMM e os resultados são comparados com um Algoritmo Genético – AG.

2. Algorítimo Genético (AG)

AG é uma técnica de busca e otimização introduzida por John Holland em 1975 [Holland 1975] e desde então vem sendo utilizado para problemas reais de alta complexidade. Os AGs buscam através de métodos heurísticos uma solução aceitável de maneira simples e eficiente.

Baseados na teoria da evolução de Darwin, os AGs buscam convergir uma população inicial aleatória para a melhor solução usando conceitos de seleção natural e evolução, ou seja, métodos probabilísticos que atuam sobre os indivíduos de uma espécie por gerações objetivando a sua evolução de acordo com a sua capacidade de adaptação. AGs usam esta evolução como um processo inteligente de otimização na busca de soluções de boa qualidade para problemas difíceis.

O AG consiste em encontrar a melhor solução para o problema baseado no valor de adaptação (função *fitness*) de seus indivíduos, representados por cromossomos. Métodos de *crossover*, mutação e seleção são utilizados pelo AG para gerar populações novas mais evoluídas e, consequentemente, melhores soluções para o problema.

Por se tratar de um método estocástico, AGs frequentemente geram soluções inválidas [Eberhart e Shi (1998)]. Porém, esta prática é comum em algoritmos evolucionários devido à necessidade de melhores buscas globais, assim o algoritmo pode passar por diferentes regiões do problema e apresentar soluções inválidas. Tais soluções possuem uma penalidade aplicada no seu valor de *fitness*, de modo que não influencie demasiadamente os outros indivíduos. Esta penalidade é calculada proporcionalmente em função do valor excedido dos limites das mochilas.

O Algoritmo 1 mostra o pseudocódigo do AG canônico. Inicialmente são determinados os parâmetros *D, P, CR* e *MU*, representando respectivamente a dimensão *D* do cromossomo, a população *P*, a taxa de *crossover CR* e a taxa de mutação *MU* (linha 2). Logo após é gerada uma população de cromossomos aleatórios (linha 3) e calculada sua função de *fitness* (linha 4), que consiste no valor da adaptação de cada indivíduo gerado. É definido um bloco repetitivo de comandos (linhas 6-9) para criar e selecionar uma nova população (linhas 10-14) até que seja alcançada uma condição terminal definida. A seleção de dois indivíduos da população atual (linha 8) 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 10) e calcula-se o valor de *fitness* dos cromossomos filhos gerados (linha 11). Estes cromossomos filhos são incorporados à nova população, que substitui a população antiga (linha 17). O Algoritmo 1 finaliza quando sua condição terminal é alcançada, geralmente associada a um número fixo de gerações.

Algoritmo 1 - Algoritmos Genéticos (AGs)

```
1:  INÍCIO
2:      Parâmetros D, P, CR, MU
3:      Gerar população inicial x_i ∈{0,1}
4:      Calcular função fitness de cada indivíduo
5:      fim = FALSO
6:      ENQUANTO NÃO (fim) FAÇA
7:          INÍCIO
8:              PARA ( P / 2) FAÇA
9:                  INÍCIO
10:                     Selecione 2 indivíduos da população atual
11:                     Aplique crossover gerando 2 filhos
12:                     Aplique mutação aos filhos
13:                     Calcule o fitness dos filhos
14:                     Coloque os filhos na nova população
15:                 FIM
16:         FIM
17:         Substitua a população antiga pela nova
18:         SE condição terminal alcançada
19:         ENTÃO fim = VERDADEIRO
20:     FIM
21: FIM
```

3. Evolução Diferencial Binária (EDB)

O EDB (*Binary Differential Evvolution – BDE*) é um algoritmo metaheurístico baseado em populações, inspirado na Evolução Diferencial (ED) padrão [Price, Storm e Lampinem (2005)] e adaptado para problemas binários. Mais precisamente, o EDB é a modificação da variante *DE/rand/1/bin* da ED, o que consiste em utilizar um indivíduo da população para gerar soluções teste, passando pelo menos uma vez pela estrutura de perturbação.

No EDB, uma população N de dimensão D interage entre si, cada vetor binário $\vec{x}_i = [x_{i1}, x_{i2}, \ldots, x_{iD}]$ é uma possível solução do problema, avaliado pela função *fitness* $f(\vec{x}_i)$ *(i = 1, ..., N)*. Assim como o ED padrão, o EDB combina a solução atual com uma solução aleatória através do *crossover*, porém a principal modificação feita no algoritmo *DE/rand/1/bin* da ED é a inserção de uma estrutura de mutação, onde a solução teste $\vec{y}$ sofrerá uma inversão no valor do *bit* em uma ou mais dimensões $\vec{y}_j$.

O pseudocódigo do EDB é mostrado no Algoritmo 2, onde *rndint* e *rndreal* são, respectivamente, números aleatórios inteiros e reais. O algoritmo tem os seguintes parâmetros: D como número de variáveis do problema ou número de *bits*, N como número de indivíduos da população, PM como parâmetro de mutação e PR como taxa

de perturbação (linha 1). Inicialmente gera-se uma população aleatória $\vec{x}_i$ (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 (linha 4-5), esta estrutura consiste em criar novos indivíduos através dos processos de perturbação (mutação e *crossover*).

Algoritmo 2 – Pseudocódigo da Evolução Diferencial Binária (EDB)

```
1:  Parâmetros: D, N, PM, PR
2:  Gerar população inicial x_i ∈ {0,1}
3:  Calcular função fitness f(x_i) de cada indivíduo
4:  ENQUANTO condição de parada não atingida FAÇA
5:      PARA i = 1 ATÉ N FAÇA
6:          Escolha aleatoriamente o índice de um indivíduo: s = rndint(1, N) com s ≠ i
7:          Escolha aleatoriamente o índice de uma variável: j_rand = rndint(1, D)
8:          Solução teste y recebe x_i
9:          PARA j = 1 ATÉ D FAÇA
10:             SE rndreal(0,1) < PR ou j = j_rand ENTÃO {Perturbação de y}
11:                 SE rndreal(0,1) < PM ENTÃO {Mutação}
12:                     InverterBit(y_j)
13:                 SENÃO {Crossover}
14:                     y_j = x_sj
15:                 FIM SE
16:             FIM SE
17:         FIM PARA
18:         Calcular f(y)
19:         SE f(y) > f(x_i) ENTÃO {Atualiza solução corrente}
20:             x_i recebe y
21:         FIM SE
22:     FIM PARA
23:     Encontrar a melhor solução corrente x*
24: FIM ENQUANTO
25: Apresentar resultados
```

Dentro desta estrutura são selecionados os índices *s* e *j* aleatórios, com *s* sendo um número inteiro entre 1 e o número de indivíduos da população *N* e diferente do valor corrente de *i* (linha 6) e *j* 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 ou 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 $\vec{y}_j$ (linha 13-14).

O valor de adaptação ou função *fitness* da solução teste $\vec{y}$ é calculado após as perturbações (linha18). Caso o valor do *fitness* do novo indivíduo seja maior que o valor de *fitness* do indivíduo antigo (linha 19), a solução $\vec{x}_i$ recebe a nova solução $\vec{y}$ (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), geralmente associada a um número fixo de gerações. O Algoritmo 2 finaliza mostrando os resultados da melhores soluções $\vec{x}_*$ (linha 25).

4. Descrição do Problema

O PMM é definido em [Martello e Toth 1990] e [Khuri, Bäck, e Heitkoetter 1994] como um problema de *n* objetos e *m* mochilas de capacidades específicas c_j ($j = 1, ..., m$). As variáveis binárias x_i ($i = 1, ..., n$) representam os itens selecionados para (1=SIM, 0=NÃO) serem levados nas *m* mochilas. Todo objeto possui um lucro p_i e um peso w_{ij} para cada mochila *m*. O objetivo é encontrar a melhor combinação de *n* objetos maximizando a somatória dos lucros p_i multiplicados pela variável binária x_i, representado matematicamente pela Equação (1). Suas restrições são as capacidades c_j de cada mochila. Portanto, o somatório dos valores de w_{ij} multiplicados por x_i deve ser menor ou igual a c_j, representado matematicamente pela Equação (2).

$$\sum_{i=1}^{n} p_i.x_i \tag{1}$$

$$\sum_{i=1}^{n} w_{ij}.x_i \leq c_j \quad (j = 1, ..., m) \tag{2}$$

Todos os valores de p_i, w_{ij} e c_j são definidos como inteiros positivos e obedecem as Equações (3) e (4).

$$w_{ij} \leq \max \{c_j\} \quad (i = 1, ..., n) \; (j = 1, ..., m) \tag{3}$$

$$c_j \geq \min \{w_{ij}\} \quad (i = 1, ..., n) \; (j = 1, ..., m) \tag{4}$$

A Equação (3) pode ser interpretada como o limite máximo que a variável w_{ij} pode assumir, sendo menor ou igual a c_j, ou seja, menor ou igual a qualquer c_j das *m* mochilas. Esta mesma analogia pode ser feita para a Equação (4), onde cada c_j é definida como menor ou igual ao mínimo dos pesos w_{ij}, limitando a capacidade mínima das mochilas ao menor peso dos *n* itens.

A dimensão ou espaço de busca do PMM depende diretamente dos valores de n e m. Uma função exponencial binária com expoente n monta todas as possibilidades de arranjo dos n elementos respeitando as capacidades de cada mochila m. Consequentemente para buscar a solução ótima, pode-se testar todas as 2^n possibilidades para cada mochila m pelo método da força bruta, ou seja, varrer o espaço de busca que cada instância cria de m x 2^n possibilidades.

5. Metodologia

A modelagem do PMM para o algoritmo EDB proposto consiste em evoluir um indivíduo inicial aleatório de n dimensões onde cada dimensão assume um valor binário x_i ($i = 1, ..., n$). Tais indivíduos são as possíveis soluções para o problema. Porém, devem respeitar as restrições c_j e não ultrapassar a carga máxima de cada mochila m.

Os indivíduos iniciais (população aleatória) são avaliados através de seu valor de *fitness*, que consiste na somatória do produto dos lucros p_i pela variável binária x_i menos as restrições, caso o indivíduo seja uma solução inválida, ou seja, que não respeite a capacidade máxima de cada mochila. A cada nova geração, estas soluções são evoluídas utilizando os processos de *crossover* e mutação descritos no Algoritmo 2, gerando uma nova população. Os resultados da função *fitness* destes novos indivíduos indicarão o quão próximo do valor ótimo cada possível solução se encontra.

Para o AG foi utilizada uma modelagem similar, onde a população de cromossomos representa os indivíduos do EDB. Com o intuito de fazer uma comparação justa, todos os parâmetros estabelecidos no EDB foram identicamente estabelecidos no AG. A Tabela 1 lista os parâmetros utilizados por cada um dos métodos testados.

Tabela 1 – Lista de parâmetros utilizados pelo AG e EDB.

Parâmetro	AG	EDB
População	100	100
Gerações	300	300
Mutação	0,1	0,1
Crossover	0,8	-
Perturbação	-	0,5

6. Experimentos

Este artigo utiliza o pseudocódigo descrito no Algoritmo 2, desenvolvido em ANSI C, para os experimentos com EDB. Nos experimentos com AG, utiliza o Galopps 3.2.4 (*Genetic ALgorithm Optimized for Portability and Parallelism System*), software flexível e genérico de AGs, escrito em ANSI C e baseado em Algoritmo Genético Simples (*Simple Genetic Algorithm – SGA*) de [Goldberg (1989)].

No intuito de testar o EDB com problemas que representem diferentes espaços de busca, as instâncias WEING1 [Weingartner e Ness (1967)], WEING7 [Weingartner e Ness (1967)], WEISH1 [Shi (1979)], WEISH14 [Shi (1979)], PB6 [Freville e Plateau (1990)] e SENTO1[Senyu e Toyada (1967)] foram selecionadas, todas disponíveis em

[Beasley (2005)] e descritas na Tabela 2, com o número *m* de mochilas, o número *n* de itens e o espaço de busca (m x 2^n). Os espaços de busca variam entre m x 2^{28} até m x 2^{105}, contudo as variações de *m* desenvolvem espaços de busca com mais ótimos locais, aumentando a complexidade da instância.

Tabela 2 – Instâncias utilizadas nos experimentos

Instância	*m*	*n*	Espaço de Busca
WEING1	2	28	2 x 2^{28}
WEING7	2	105	2 x 2^{105}
WEISH1	5	30	5 x 2^{30}
WEISH14	5	60	5 x 2^{60}
PB6	30	40	30 x 2^{40}
SENTO1	30	60	30 x 2^{60}

Ambos os algoritmos foram executados 100 vezes para cada instância, cada uma utilizando uma semente aleatória diferente e uniformemente distribuída, totalizando 1200 experimentos. Todos os experimentos foram executados utilizando um processador Intel Core i3, CPU 2.2 GHz.

7. Resultados e Análise

A Tabela 3 mostra as melhores soluções alcançadas pelo EDB e AG, todos executados com os parâmetros listados na Tabela 1. Cada instância descreve o valor ótimo conhecido (Opt), a média das 100 soluções encontradas (Média), o desvio padrão (±DP), o melhor resultado encontrado por cada método (Melhor) e a porcentagem (%) da diferença entre o valor Opt e o Melhor encontrado.

Tabela 3 – Comparação dos resultados alcançados pelo AG e EDB.

Instância	Opt	AG		EDB	
		Média ±DP (Melhor)	%	Méd ±DP (Melhor)	%
WEING1	141278	141277,6 ± 2,8 (141278)	0,00%	141277,4 ± 3,4 (141278)	0,00%
WEING7	1095445	1052993,9 ± 7279,3 (1078722)	1,52%	1087073,1 ± 1285,9 (1089841)	0,51%
WEISH1	4554	4544,4 ± 21,6 (4554)	0,00%	4554,0 ± 0,0 (4554)	0,00%
WEISH14	6954	6427,9 ± 124,6 (6723)	3,32%	6795,8 ± 53,1 (6914)	0,57%
PB6	776	700,8 ± 36,6 (770)	0,77%	616,3 ± 121,8 (773)	0,38%
SENTO1	7772	7094,0 ± 214,3 (7543)	2,94%	5395,5 ± 930,5 (7596)	2,26%

Ao se analisar os valores encontrados pelo AG e EDB na Tabela 3, verifica-se que ambos os métodos encontraram valores satisfatórios e muito perto dos ótimos conhecidos. Contudo, em todas as instâncias utilizadas o algoritmo EDB superou os valores ótimos encontrados pelo AG. Os resultados ainda apontam uma semelhança

entre os métodos, que atingiram o ótimo conhecido nas instâncias com os menores espaços de busca (WEING1 e WEISH1), conforme a Tabela 2.

A análise das médias e desvio padrão (Média ±DP) da Tabela 3 detalha o intervalo que abrange todas as soluções encontradas. Para uma melhor comparação destes resultados, foi utilizado um teste estatístico *t* de *Student* com um intervalo de 95% de confiança para a média $\bar{x}$ encontrada pelos métodos. A Tabela 4 apresenta estes intervalos.

Tabela 4 – Intervalos com 95% de confiança, teste estatístico *t Student*.

Instância	AG	EDB
WEING1	$141277{,}61 \leq \bar{x} \leq 141277{,}58$	$141277{,}42 \leq \bar{x} \leq 141277{,}37$
WEING7	$1053039{,}75 \leq \bar{x} \leq 1052948{,}04$	$1087081{,}20 \leq \bar{x} \leq 1087064{,}99$
WEISH1	$4544{,}53 \leq \bar{x} \leq 4544{,}26$	$4554 \leq \bar{x} \leq 4554$
WEISH14	$6428{,}68 \leq \bar{x} \leq 6427{,}11$	$6796{,}13 \leq \bar{x} \leq 6795{,}46$
PB6	$701{,}03 \leq \bar{x} \leq 700{,}56$	$617{,}06734 \leq \bar{x} \leq 615{,}53266$
SENTO1	$7095{,}35 \leq \bar{x} \leq 7092{,}64$	$5401{,}36 \leq \bar{x} \leq 5389{,}63$

Tais intervalos apontam o local da média dos 100 resultados encontrados, eles também demonstram diferença significativa entre as soluções encontradas pelo EDB e AG, pois em nenhuma instância analisada os intervalos deste teste estatístico se sobrepõem.

As variações dos resultados da Tabela 3 e os intervalos da Tabela 4 refletem o comportamento de cada algoritmo frente a instâncias com espaços de busca de tamanho e complexidade diferentes.

Para se comparar o EDB e o AG utilizados nestes experimentos, os Gráficos 1, 2, 3, 4, 5 e 6 mostram a convergência das soluções para o máximo obtido. Eles representam o valor da função *fitness* normalizada entre 0 e 1 da melhor solução encontrada a cada geração (AG) ou iteração (EDB).

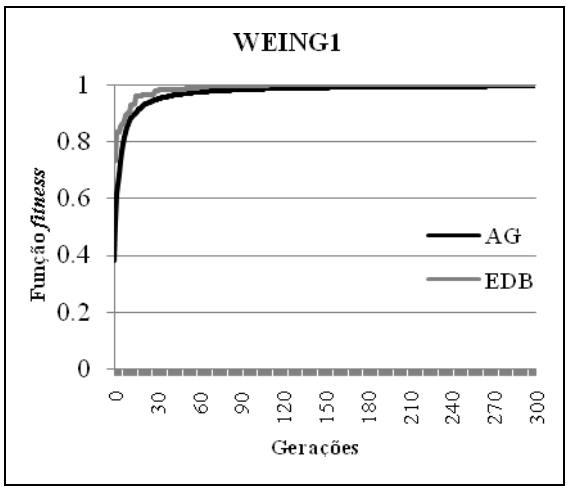

Gráfico 1 – Função *fitness* (WEING1)

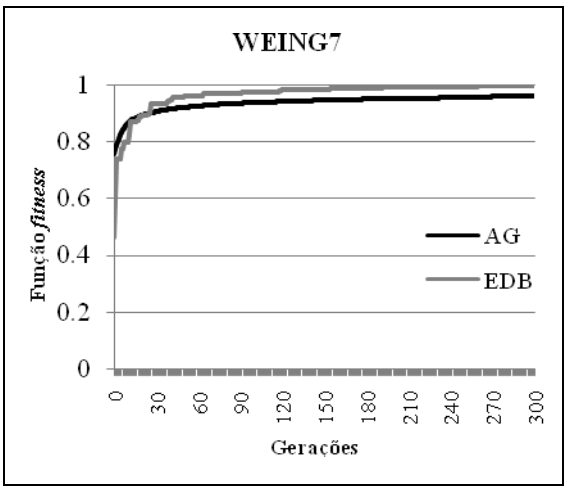

Gráfico 2 – Função *fitness* (WEING7)

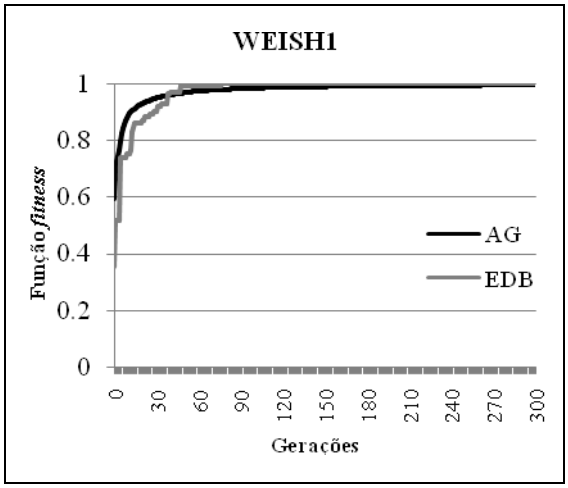

Gráfico 3 – Função *fitness* (WEISH1)

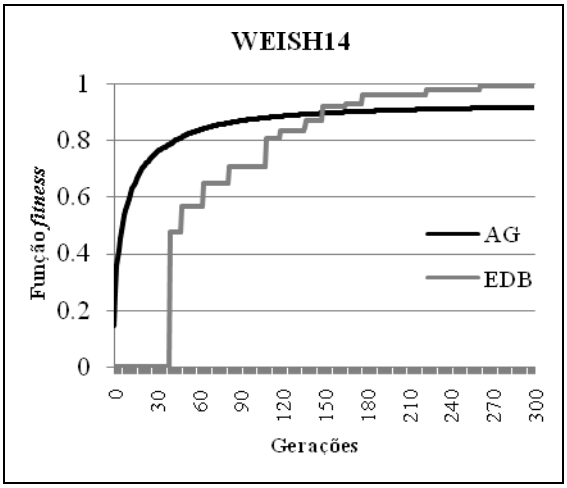

Gráfico 4 – Função *fitness* (WEISH14)

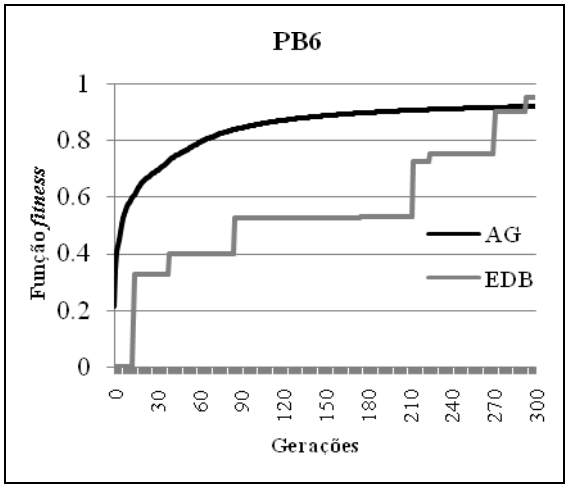

Gráfico 5 – Função *fitness* (PB6)

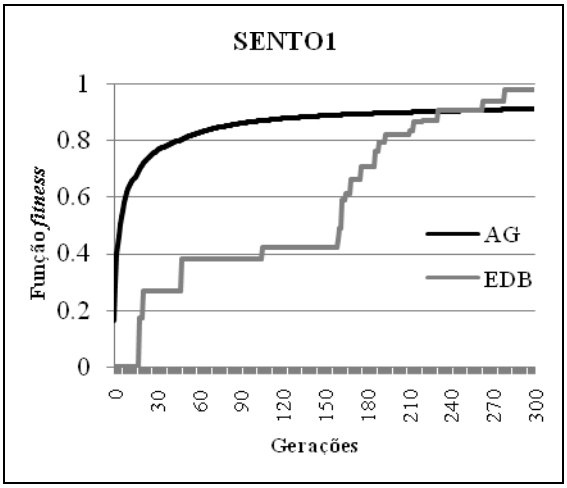

Gráfico 6 – Função *fitness* (SENTO1)

Ao analisar os Gráficos 1-6 pode-se perceber como cada algoritmo se comporta durante a evolução da melhor solução encontrada. Nos Gráficos 1 e 3 nota-se que ambos algoritmos atingiram o valor máximo de *fitness* (1,00) e o valor ótimo conhecido (Tabela 3).

Para todas as instâncias, exceto para WEING1, os valores da função de *fitness* do AG convergem muito rapidamente quando comparados com os valores do EDB. Esta característica sugere que o AG converge para um ponto de máximo local e fica estagnado sem possibilidade de melhoria. Os Gráficos 4, 5 e 6 evidenciam ainda mais a diferença de evolução entre EDB e AG, para problemas com espaços de busca mais complexos. Nestes casos, a função de *fitness* do EDB evolui mais suavemente, sugerindo a manutenção da diversidade ao longo das iterações. Esta é uma característica essencial para a obtenção de soluções boas para instâncias de grande complexidade como as instâncias PB6 e SENTO1 (Tabela 2). Observa-se, ainda, nos Gráficos 5 e 6 que a curva de evolução do EDB tem um grande potencial de melhora. Assim, se o número máximo de gerações fosse aumentado, certamente o EDB poderia obter resultados ainda melhores para estas instâncias.

O tempo em processamento de cada instância também é um resultado muito relevante para os experimentos. Ele demonstra o esforço computacional que cada método necessita para processar seu algoritmo. A Tabela 6 compara o tempo de processamento que o AG e o EDB levaram para executar uma vez cada instância.

Tabela 6 – Tempo de processamento para uma execução do AG e do EDB

Instância	AG	EDB
WEING1	2,360s	0,120s
WEING7	7,136s	0,320s
WEISH1	4,036s	0,148s
WEISH14	7,656s	0,328s
PB6	25,022s	0,528s
SENTO1	37,618s	0,780s

Analisando os dados da Tabela 6 pode-se concluir que, em média, o EDB foi 30 vezes mais rápido que o AG, demonstrando-se um algoritmo rápido e eficaz. Para as 100 execuções do algoritmo para cada instância, o tempo em processamento total do AG ultrapassou 2,3 horas, enquanto o EDB totalizou menos de 4 minutos.

8. Conclusão

O PMM e suas aplicações na indústria são motivações para pesquisadores buscarem novos e alternativos métodos para resolver problemas NP-completos. Os experimentos feitos com o novo algoritmo EDB apresentam resultados muito satisfatórios, com uma vantagem significativa quando comparado com os resultados alcançados com o AG. O algoritmo de Evolução Diferencial adaptada para problemas binários e com o recurso de mutação mostrou-se eficaz, robusto e promissor. O baixo esforço computacional é uma característica forte deste método pois, em média, foi 30 vezes mais rápido do que um AG.

Por ser ainda um método novo, trabalhos futuros vão focar no teste do EDB com outras instâncias do PMM, bem como analisar diferentes ajustes de parâmetros e aplicar em outros problemas binários. De maneira geral, os resultados relatados neste artigo sugerem um grande potencial do EDB, encorajando a continuidade da pesquisa.

Referências

Martello, S. e Toth., P. (1990) Knapsack problems – Algorithms and Computer Implementation. John Wiley & Sons, New York, USA.

Beasley, J.E. e Chu, P.C. (1998) Genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics 4:63−86.

Khuri, S., Bäck, T. e Heitkoetter, J. (1994) The zero/one multiple knapsack problem and genetic algorithm. Proc. ACM Symposium on Applied Computing, Phoenix, USA, pp. 188−193.

Hristakeva, M. e Shrestha, D. (2011) Solving the 0-1 Knapsack Problem with Genetic Algorithms. Simpson College, Iowa, USA.

Taskiran, G.K. (2010) An improved Genetic Algorithm for Knapsack Problems. Wright State University, Ohio, USA.

Bortfeldt, A. e Winter, T. (2008) A Genetic Algorithm for the Two-Dimensional Knapsack Problem with Rectangular Pieces. Fern Universität, Hagen, Germany.

Holland, J.H. (1975) Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.

Eberhart, R.C. e Shi, Y. (1998) Comparison between genetic algorithms and particle swarm optimization. Proc. 7th Annual Conference on Evolutionary Programming, pp. 611−616

Price, K. V., Storm, R. M. and Lampinem, J. A. (2005), Differential Evolution. A Practical Approach to Global Optimization.

Goldberg, D. E. (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. EUA: Addison-Wesley.

Weingartner, H. M. and Ness, D. N. (1967) Methods for the solution of the multi-dimensional 0/1 knapsack problem. Operations Research, 15:83-103.

Shi, W. (1979) A branch and bound method for the multiconstraint zero one knapsack problem. J. Opl. Res. Soc., 30:369-378.

Freville, A. and Plateau, G. (1990) Hard 0-1 multiknapsack test problems for size reduction methods. Investigación Operativa, 1:251-270.

Senyu, S. and Toyada, Y. (1967) An approach to linear programming with 0-1 variables. Management Science, 15:B196-B207.

Beasley, J. E. (2005) Operations Research Library (ORLib). [http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/mknap2.txt]