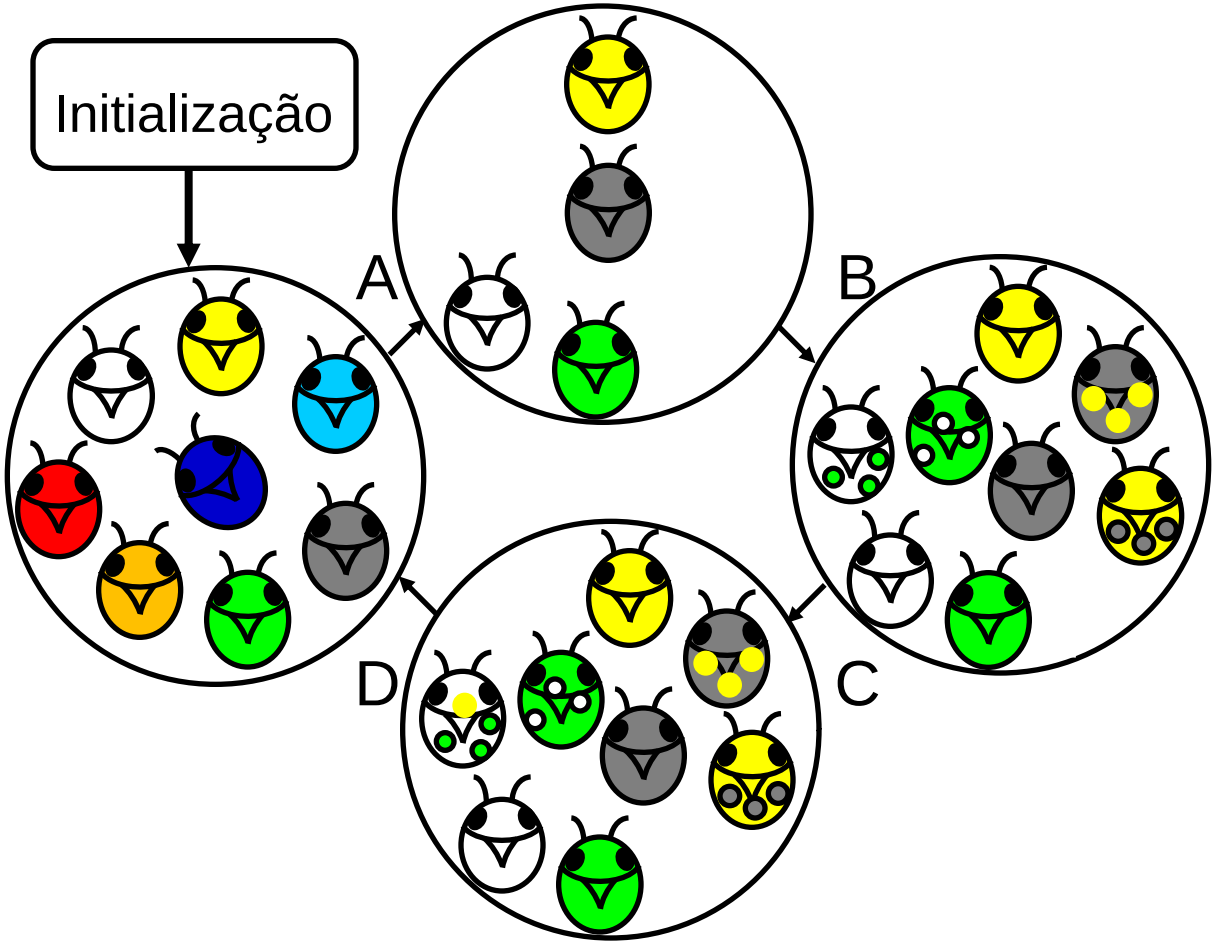
Computação Bioinspirada

Aula 07

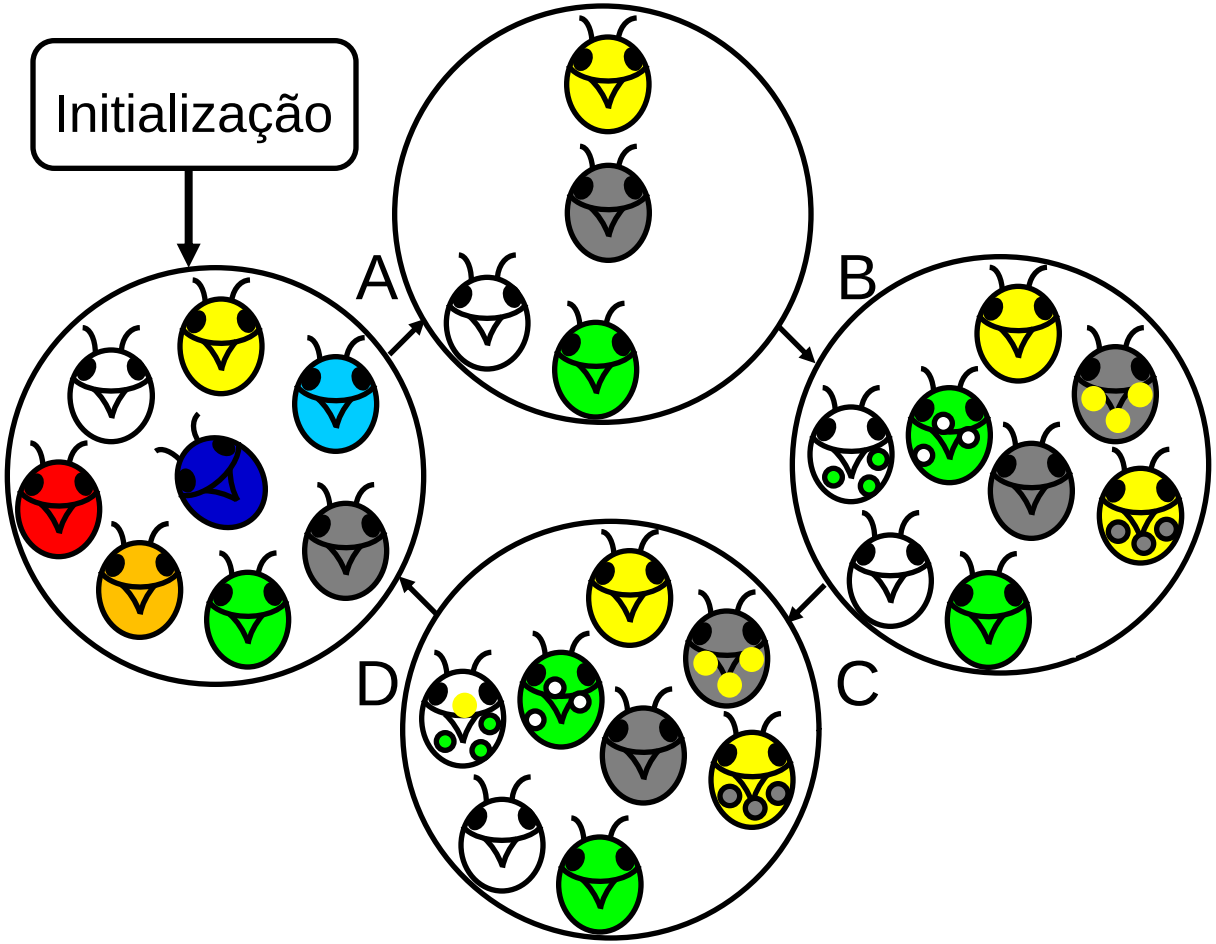
azevedolab.net

© 2017 Dr. Walter F. de Azevedo Jr.

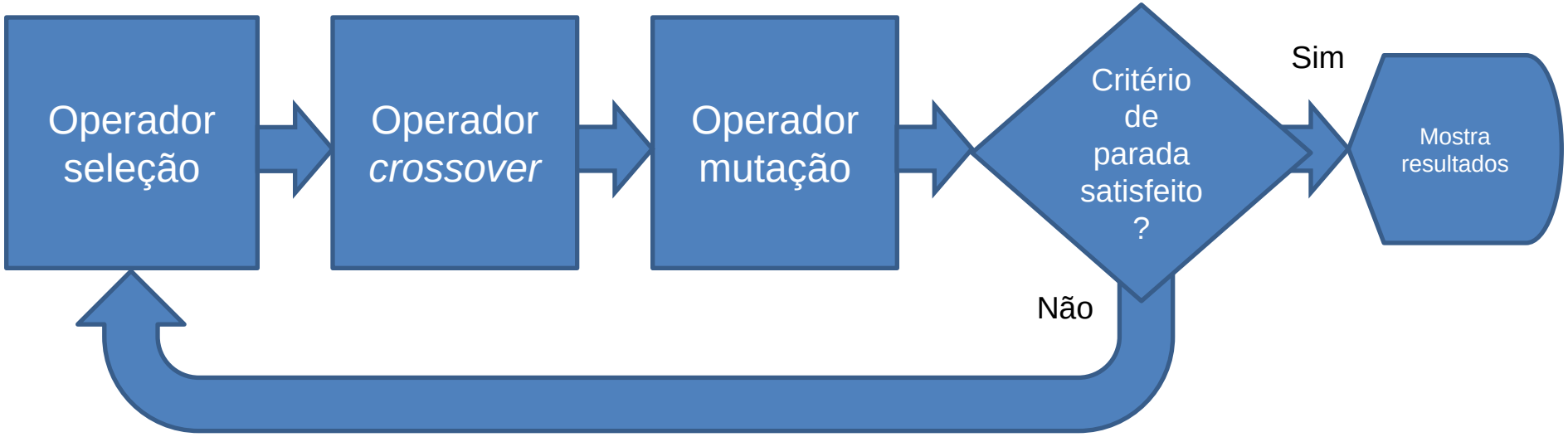
Os últimos algoritmos evolucionários que implementamos estavam baseados na ideia original do algoritmo genético, como ilustrado no diagrama abaixo. No algoritmo genético temos a geração de uma população aleatória (inicialização).



Em seguida temos o operador seleção (A). Na sequência temos o operador *crossover* (B). A população resultante do *crossover* é submetida ao operador mutação (C). A população resultante é a população inicial para um novo ciclo (iteração).



Há outros algoritmos que usam as ideias de evolução, mas são baseados em métodos computacionais distintos. Todos usam como inspiração a evolução, mas sua implementação computacional varia de algoritmo para algoritmo. Um dos algoritmos evolucionários de maior sucesso é o **algoritmo de evolução diferencial**. Na evolução diferencial temos os operadores clássicos dos algoritmos genéticos: seleção, *crossover* e mutação, como mostrados abaixo. Como uma implementação distinta do *crossover*.



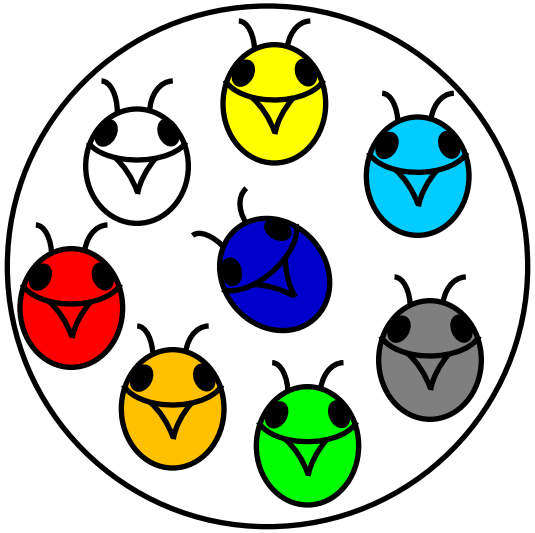
O algoritmo de evolução diferencial executa a parte inicial de geração aleatória de uma população com N cromossomos, em seguida avalia a função ajuste de cada cromossomo. A novidade está na formação cromossomos filhos. O filhos são gerados da seguinte forma, para cada cromossomo pai na população, chamado aqui cromossomo j, são escolhidos aleatoriamente três outros cromossomos pais distintos, cromossomos k, l e m. Calcula-se um novo cromossomo, chamado aqui de n, da seguinte forma:

$$cromossomo(n) = cromossomo(m) + peso.[cromossomo(k) - cromossomo(l)]$$

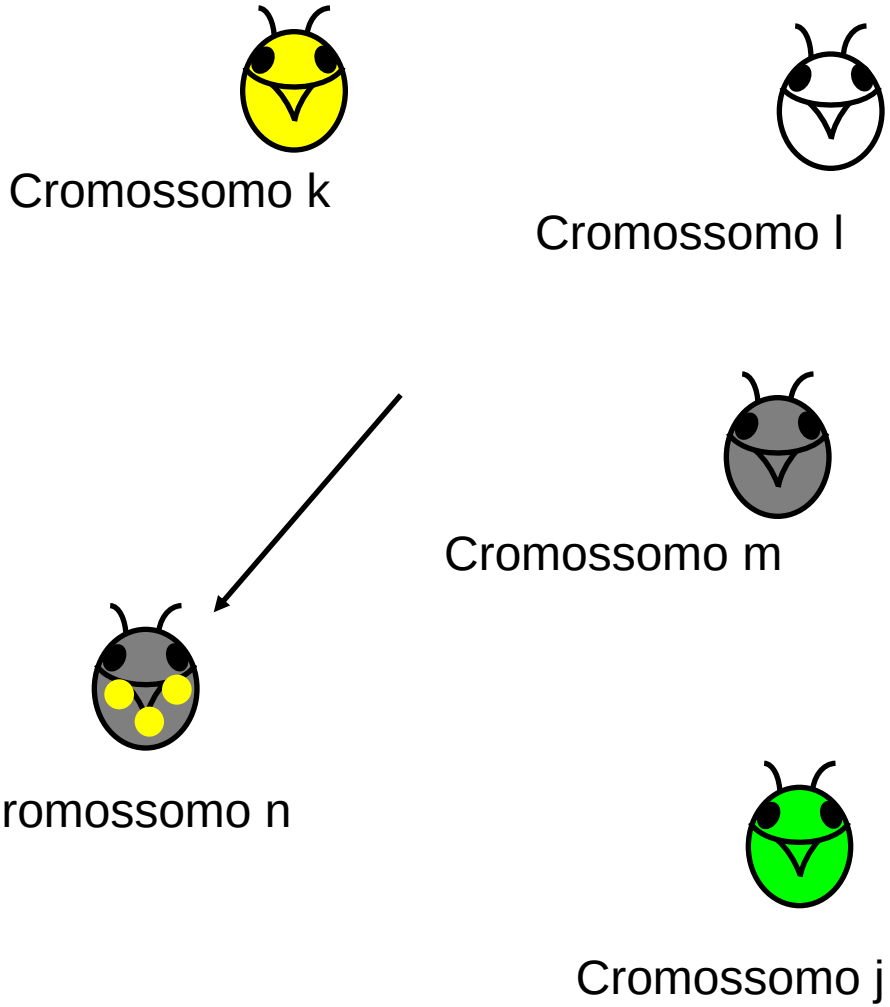
o peso varia entre 0 e 2.

O cromossomo novo (n) será incorporado à população, se ao gerarmos um número aleatório entre 0 e 1, este for menor que a probabilidade de *crossover*, caso não seja, o cromossomo filho não é considerado. Um último teste é realizado no cromossomo filho n, se este apresenta função escore maior que o cromossomo pai j, caso seja, o cromossomo j é deletado e substituído pelo cromossomo n. As etapas seguintes são idênticas ao algoritmo genético original. Estamos considerando aqui que a função ajuste de maior valor é a que representa um indivíduo mais bem adaptado. Veja, este critério depende do tipo de problema a ser resolvido pelo algoritmo evolucionário.

Na implementação do algoritmo de evolução diferencial, como nos algoritmos genéticos, o primeiro passo é gerar indivíduos de forma aleatória, como no exemplo das joaninhas.



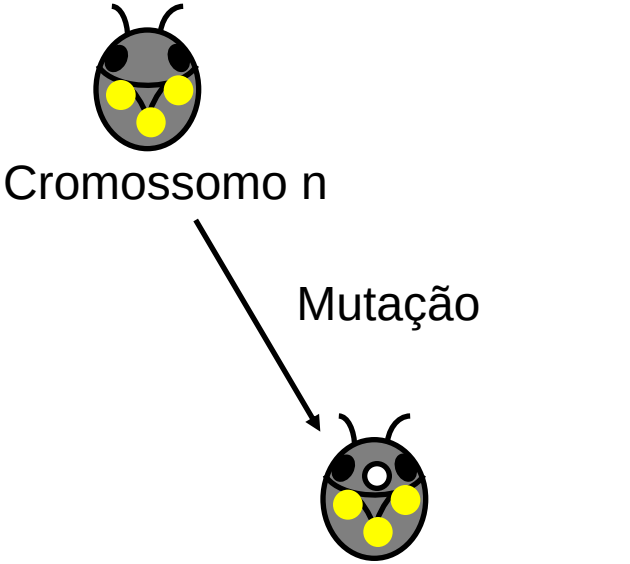
No passo seguinte, calculamos a função ajuste, que indica a adaptação de cada indivíduo. Depois aplicamos o operador *crossover*, descrito anteriormente, onde três joaninhas pais (joaninhas cinza, branca e amarela) geram uma joaninha filha, para cada cromossomo pai (joaninha verde), como mostrado ao lado.

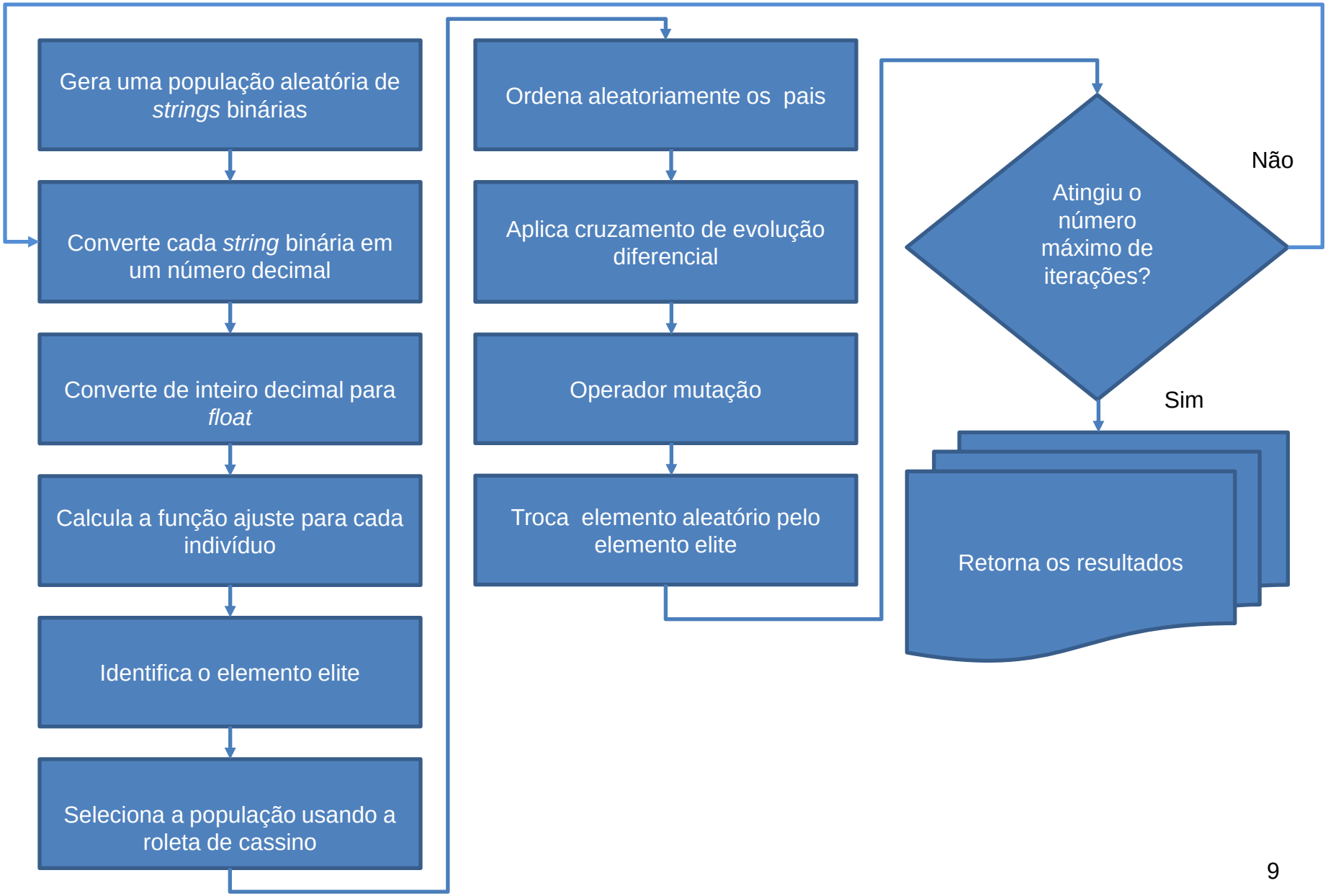


cromossomo(n) = cromossomo(m) + peso.[cromossomo(k) - cromossomo(l)]

Considerando que o cromossomo n passou no teste da probabilidade de *crossover*, e que tem função ajuste maior que a do pai (cromossomo j), este ocupa o lugar do pai, ou seja, o cromossomo j é deletado e substituído pelo cromossomo n. Por último, aplicamos a mutação, onde selecionamos aleatoriamente cromossomos filhos. O critério para escolher se um cromossomo filho sofrerá mutação, é a probabilidade de mutação (Pm). Geramos um número aleatório entre 0 e 1, se este número for menor ou igual à probabilidade de mutação, a mutação ocorre, caso contrário não. Testamos a condição de mutação para todos os cromossomos filhos. Terminamos um ciclo da evolução, ou iteração. O processo se repete um número finito de vezes.

Cromossomo j é deletado e substituído pelo n




Gera uma população aleatória de *strings* binárias
Converte cada *string* binária em um número decimal
Converte de inteiro decimal para *float*
Calcula a função ajuste para cada indivíduo
Identifica o elemento elite
Seleciona a população usando a roleta de cassino
Ordena aleatoriamente os pais
Aplica cruzamento de evolução diferencial
Operador mutação
Troca elemento aleatório pelo elemento elite
Atingiu o número máximo de iterações?
Não
Sim
Retorna os resultados

Projeto 3 - Algoritmo de Evolução Diferencial para Funções de Duas Variáveis
Programa: *de01.py*

Data de entrega: 28/07/2017

Modifique o código do programa *lga03.py* (projeto 2), de forma que o programa ache o máximo ou o mínimo de uma função de duas variáveis (*f(x,y)*) a partir do algoritmo de evolução diferencial. O programa deve gerar o gráfico da função de duas variáveis.
O novo programa não terá a classe *Touche* para desenhar as tartarugas.
Deve ser entregue o código (*de01.py*), bem como um **arquivo PDF** na forma de relatório com os testes para identificação do mínimo para todas as funções de duas variáveis indicadas no site www.python4stem.net. Usem o código *sphere.py* como modelo para programarem classes para cada uma das funções testes.
O relatório deve conter uma tabela com as seguintes informações para cada função teste.

Função	Equação	Mínimo Global	Intervalo	Parâmetros usados no *de.in*	Mínimo Global Obtido
Sphere	$f(x,y) = x^2+y^2$	f(0,0) = 0	-5.12<x,y<5.12		
Ackley					
Bukin Function N. 6					

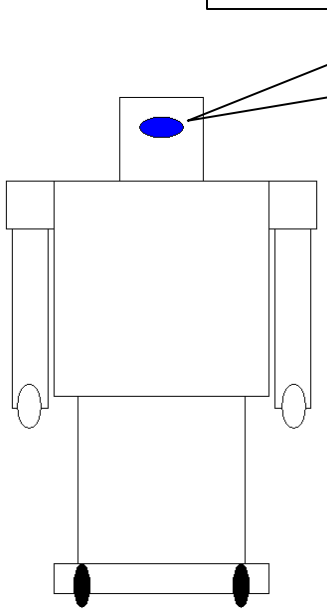
Devem ser usados os seguintes parâmetros como entrada (arquivo *de.in*):

```
# Set up initial values for DE
weight_in = 1.5          # Weight to be used in the new crossover operator
pop_size = 500           # Population size
max_iter = 100           # Number of iterations
p_cross = 0.8            # Probability of crossover
len_str1 = 36            # Length of strings 1
len_str2 = 36            # Length of strings 2
p_mut = 0.01             # Probability of mutation_Coley_algo_03
seed = 12345             # Seed for generation of random number
min_f = -5.12            # Minimum float
max_f = 5.12             # Maximum float
type_of_opt = "min"      # Type of optmimization (to find maximum or minimum)
```

Os valores atribuídos aos parâmetros acima são apenas ilustrativos, vocês devem testar o melhor conjunto de parâmetros para cada função teste.

Algumas funções teste têm domínios distintos para o eixo *x* e *y*, o programa *de01.py* considera domínios idênticos para os eixos *x* e *y*. Assim, considere *min_f* e *max_f* o mais abrangente possível, o suficiente para incluir ambos domínios.

This text was produced in a HP Pavillon dm4 notebook with 6GB of memory, a 500 GB hard disk, and an Intel® Core® i7 M620 CPU @ 2.67 GHz running Windows 7 Professional. Text and layout were generated using PowerPoint 2007. The image on the first slide was taken from http://briandcolwell.com/2017/02/artificial-intelligence-the-big-list-of-things-you-should-know/.html on March 26th 2017. This text uses Arial font.



-BRESSERT, Eli. **SciPy and NumPy**. Sebastopol: O'Reilly Media, Inc., 2013. 57 p. Link

-DAWSON, Michael. **Python Programming, for the Absolute Beginner.** 3ed. Boston: Course Technology, 2010. 455 p.

-COLEY, David A. **An Introduction to Genetic Algorithms for Scientists and Engineers.** Singapore: World Scientific Publishing Co. Pte. Ltd., 1999. 227 p.

-EIBEN, A. E., SMITH, J. E**. Introduction to Evolutionary Computing. Natural Computing Series**. 2nd ed. Berlin: Springer-Verlag, 2015. 287 p. Link

-HACKELING G. **Mastering Machine Learning with scikit-learn**. Birmingham: Packt Publishing Ltd., 2014. 221 p. Link

-HETLAND, Magnus Lie. **Python Algorithms. Mastering Basic Algorithms in the Python Language.** 2ed. Nova York: Springer Science+Business Media LLC, 2010. 294 p. Link

-IDRIS, Ivan. **NumPy 1.5. An action-packed guide dor the easy-to-use, high performance, Python based free open source NumPy mathematical library using real-world examples. Beginner's Guide.** Birmingham: Packt Publishing Ltd., 2011. 212 p. Link

-LUTZ, Mark. **Programming Python.** 4ed. Sebastopol: O'Reilly Media, Inc., 2010. 1584 p. Link

-TOSI, Sandro. **Matplotlib for Python Developers.** Birmingham: Packt Publishing Ltd., 2009. 293 p. Link

Última atualização: 28 de maio de 2017.