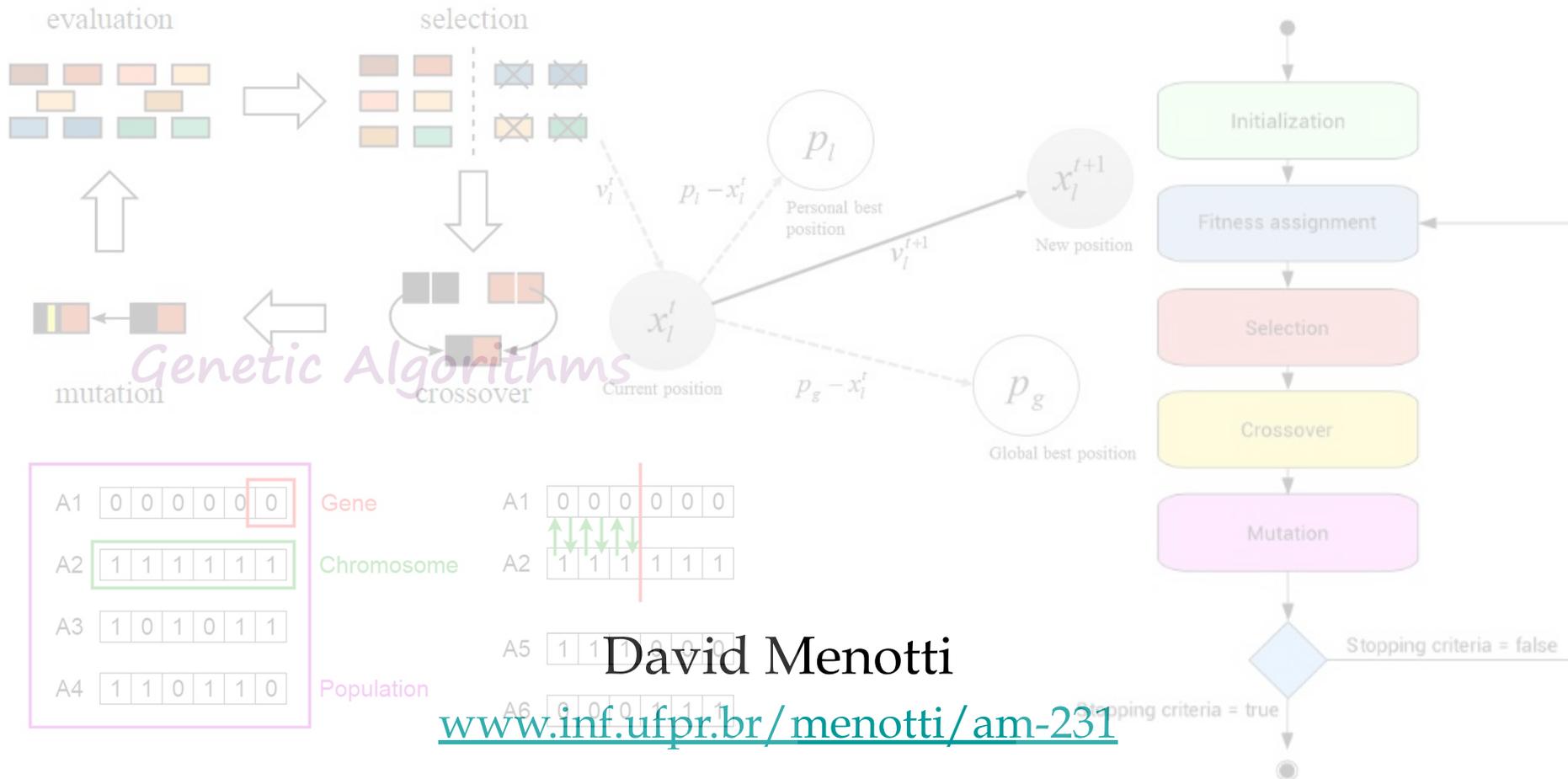


# Computação Evolutiva

## Algoritmos Genéticos & PSO



David Menotti

# Hoje

- Computação Evolucionária
  - Algoritmos Genéticos
  - *Particle Swarm Optimization*  
(Inteligência de Enxames)

# Computação Evolutiva

# Objetivos

- Introduzir os principais conceitos dos algoritmos genéticos
  - SOGA (Single-Objective GA)
  - MOGA (Multi-Objective GA)
- Entender como e por que eles funcionam.
- Vislumbrar possíveis aplicações de otimização usando AG.

# Introdução

- Primeiros trabalhos datam da década de 50.
  - A. Frazer (1957)
- John Holland (Pai dos AGs)
  - *Adaptation in Natural and Artificial Systems* (1975)
- David Goldberg (aluno de Holland)
  - *Genetic Algorithms in Search, Optimization, and Machine Learning* (1989)
- Lawrence Davis
  - Handbook of Genetic Algorithms

# Principais Conceitos

- Baseado na evolução natural.
- População de indivíduos, também chamados de cromossomos.
  - Soluções potenciais.
- Um gene do cromossomo geralmente corresponde à uma variável do problema.
  - Representações binária e real.

# Algoritmo Básico

Início

$t \leftarrow 0$

Inicializar População( $t$ )

Enquanto (condição de terminação não for satisfeita)

$t \leftarrow t + 1$

Seleciona População( $t$ ) de População( $t - 1$ )

Cruzamento População( $t$ )

Mutação População( $t$ )

Avaliação População( $t$ )

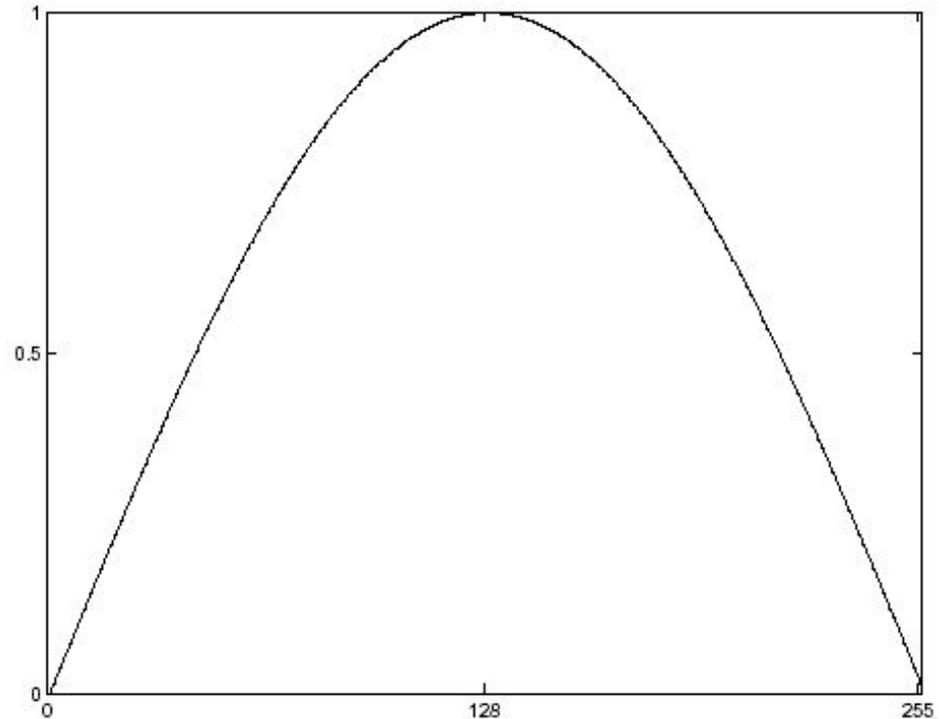
Fim

Fim

# Um Exemplo

- Encontrar o valor de  $x$  que maximize a função:

$$f(x) = \sin(\pi x / 256)$$
$$0 \leq x \leq 255$$



# Representando o Problema

- Esse problema contém uma única variável ( $x$ ), a qual pode assumir valores entre 0 e 255.
- Utilizando uma codificação binária, a variável  $x$  pode ser codificada em uma string de 8 bits:
  - 00000000 -> 0.
  - 11111111 -> 255.

# Determinando a População Inicial

- Nesse exemplo usaremos uma população de 8 indivíduos, inicializados aleatoriamente.

<u>Indivíduos</u>
10111101
11011000
01100011
11101100
10101110
01001010
00100011
00110101

# Fitness da População Inicial

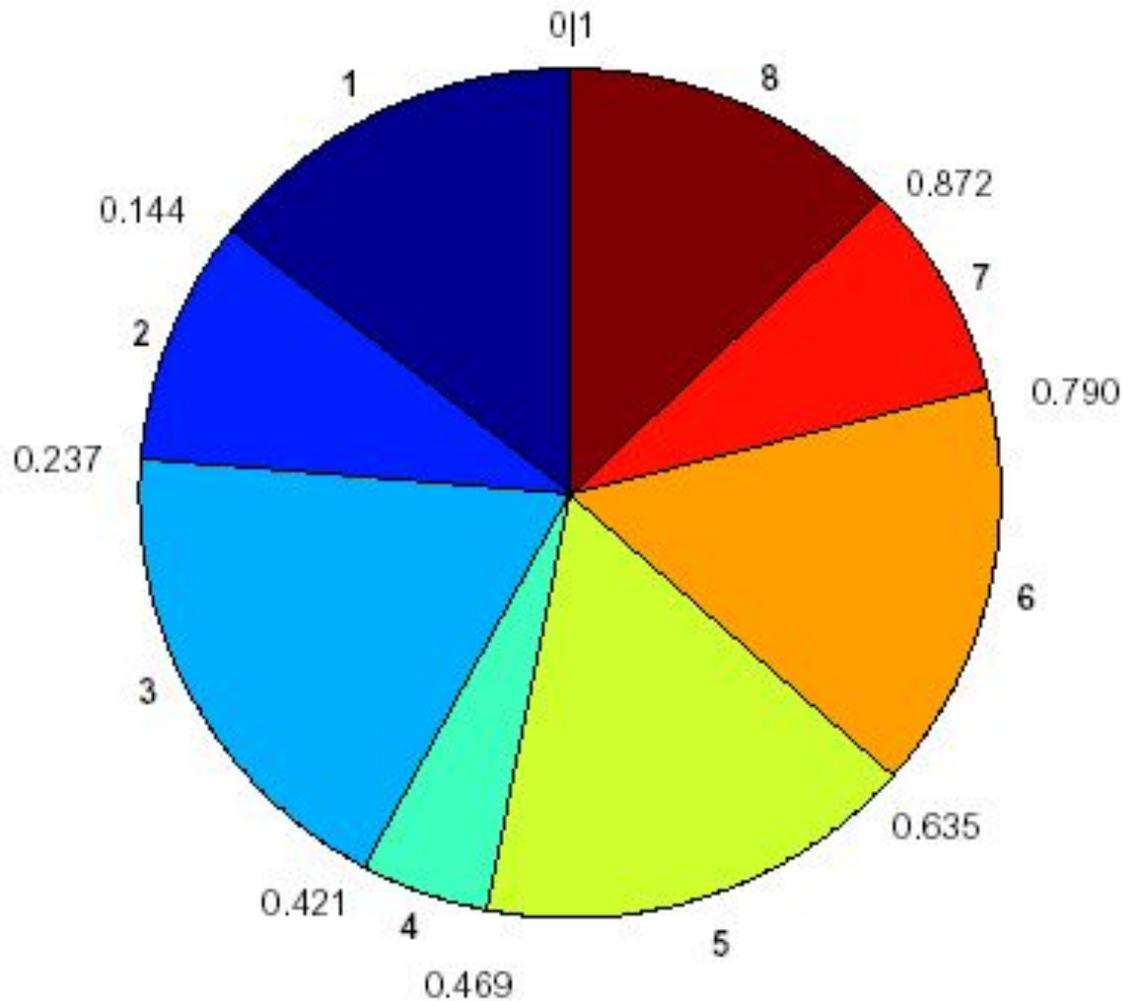
Indivíduos	$x$	$f(x)$	$f_{norm}$	$f_{acm}$
10111101	189	0.733	0.144	0.144
11011000	216	0.471	0.093	0.237
01100011	99	0.937	0.184	0.421
11101100	236	0.243	0.048	0.469
10101110	174	0.845	0.166	0.635
01001010	75	0.788	0.155	0.790
00100011	35	0.416	0.082	0.872
00110101	53	0.650	0.128	1.000

$$f(x) = \sin(\pi x / 256)$$

# Reprodução

- Após o cálculo da fitness acontece a reprodução.
  - Gerar uma nova população com o mesmo número de indivíduos.
- Processo estocástico que leva em consideração a fitness normalizada.
  - Indivíduos ruins também tem chance de reproduzir (probabilidade baixa).

# Roleta



As porções maiores tem mais chances de serem selecionadas, porem as menores também tem chances (reduzidas, é claro).

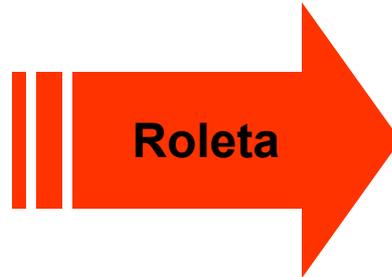
# Reprodução

- Rodamos a roleta oito vezes.
  - Se o número cair entre 0 e 0.144, o indivíduo selecionado é o 1, e assim por diante.
- Número gerados aleatoriamente:
  - 0.293, 0.971, 0.160, 0.169, 0.664, 0.568, 0.371, 0.109
- Indivíduos selecionados:
  - 3, 8, 2, 5, 6, 5, 3, 1

# População Reproduzida.

População Inicial

Indivíduos
10111101
11011000
01100011
11101100
10101110
01001010
00100011
00110101



Indivíduos Reproduzidos

Índice	Indivíduos
3	01100011
8	00110101
2	11011000
5	10101110
6	01001010
5	10101110
3	01100011
1	10111101

**Nesse caso**, os piores indivíduos (7 e 4) não foram selecionadas para compor a nova população.

# Cruzamento

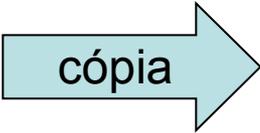
- Após a reprodução da população, a mesma sofre duas operações genéticas:
  - Cruzamento e Mutação.
- Cruzamento troca porções de strings de dois indivíduos pais.
  - O filho terá alguns genes do pai e alguns da mãe.
- O cruzamento é baseado em uma probabilidade que indica quantos indivíduos sofreram cruzamento.

$$P_c = 0.75 \quad \longrightarrow \quad (8 \times 0.75 = 6)$$

# Cruzamento

- Sendo assim, 3 pares de indivíduos serão selecionados aleatoriamente e o restante será simplesmente copiado para a nova população.
- Por uma questão de simplicidade, selecionamos os 6 primeiros indivíduos para o cruzamento.

# Cruzamento

Pontos de Cruzamento	Indivíduos após o cruzamento	Fitness
<pre> 0 1 1   0 0 0   1 1 0 0 1   1 0 1   1 1           </pre>	<pre> 0 1 1 1 0 1 1 1 0 0 1 0 0 0 1 1           </pre>	<p>0.993 (119)</p> <p>0.41 (35)</p>
<pre> 1   1 0 1   1 0 0 0 1   0 1 0   1 1 1 0           </pre>	<pre> 1 0 1 0 1 0 0 0 1 1 0 1 1 1 1 0           </pre>	<p>0.88 (168)</p> <p>0.40 (222)</p>
<pre> 0 1   0 0 1 0 1   0 1 0   1 0 1 1 1   0           </pre>	<pre> 0 1 1 0 1 1 1 0 1 0 1 0 1 0 1 0           </pre>	<p>0.992 (138)</p> <p>0.97 (110)</p>
<pre> 0 1 1 0 0 0 1 1 1 0 1 1 1 1 0 1           </pre>	<p> <b>cópia</b></p> <pre> 0 1 1 0 0 0 1 1 1 0 1 1 1 1 0 1           </pre>	<p>0.93 (99)</p> <p>0.73 (189)</p>
	75%	

# Mutação

- Evitar a convergência prematura do algoritmo.
  - Taxas de mutação entre 0.1 e 1% são geralmente utilizadas.
  - Altas taxas de mutação faz com que o AG explore diferentes áreas do espaço.
  - Geralmente inicia-se com taxas de mutação mais elevadas e diminuí-se conforme o algoritmo converge.

# Mutação.

Indivíduos após o  
cruzamento

Fitness

0 1 1 1 0 1 1 1  
0 0 1 0 0 0 1 1  
1 0 1 0 1 0 0 0  
1 1 0 1 1 1 1 0  
0 1 1 0 1 1 1 0  
1 0 1 0 1 0 1 0  
0 1 1 0 0 0 1 1  
1 0 1 1 1 1 0 1

0.993 (119)  
0.41 (35)  
0.88 (168)  
0.40 (222)  
0.992 (138)  
0.97 (110)  
**0.34 (227)**  
0.73 (189)

1 1 1 0 0 0 1 1

Bit Selecionado  
Aleatoriamente  
 $P_m = 1/64$



0

# Nova População

População Inicial	Fitness	População Intermediária	Fitness
0 1 1 0 0 0 1 1	0.73	0 1 1 1 0 1 1 1	0.993
0 0 1 1 0 1 1 1	0.47	0 0 1 0 0 0 1 1	0.41
1 1 0 1 1 0 0 0	0.93	1 0 1 0 1 0 0 0	0.88
1 0 1 0 1 1 1 0	0.24	1 1 0 1 1 1 1 0	0.40
0 1 0 0 1 0 1 0	0.84	0 1 1 0 1 1 1 0	0.992
1 0 1 0 1 1 1 0	0.78	1 0 1 0 1 0 1 0	0.97
0 1 1 0 0 0 1 1	0.41	1 1 1 0 0 0 1 1	0.34
1 0 1 1 1 1 0 1	0.65	1 0 1 1 1 1 0 1	0.73

A nova população (que dá início a segunda geração do algoritmo) deve ser do tamanho da população inicial, ou seja, 8 indivíduos.

# Nova População

- Estratégias mais comuns para selecionar a nova população:
  - Roleta
    - Processo estocástico, onde o melhor indivíduo pode ser perdido
  - Somente a população intermediária
  - Ranking
    - Garante o melhor indivíduo na próxima população
    - **Estratégia Elitista**

# Nova População

- Utilizando a estratégia de ranking, a nova população seria:

0 1 1 0 0 0 1 1 0.73	0 1 1 1 0 1 1 1 0.993	0 1 1 1 0 1 1 1
0 0 1 1 0 1 1 1 0.47	0 0 1 0 0 0 1 1 0.41	0 1 1 0 1 1 1 0
1 1 0 1 1 0 0 0 0.93	1 0 1 0 1 0 0 0 0.88	1 0 1 0 1 0 1 0
1 0 1 0 1 1 1 0 0.24	1 1 0 1 1 1 1 0 0.40	1 1 0 1 1 0 0 0
0 1 0 0 1 0 1 0 0.84	0 1 1 0 1 1 1 0 0.992	1 0 1 0 1 0 0 0
1 0 1 0 1 1 1 0 0.78	1 0 1 0 1 0 1 0 0.97	0 1 0 0 1 0 1 0
0 1 1 0 0 0 1 1 0.41	1 1 1 0 0 0 1 1 0.34	1 0 1 0 1 1 1 0
1 0 1 1 1 1 0 1 0.65	1 0 1 1 1 1 0 1 0.73	1 0 1 1 1 1 0 1

# Exercício

- Encontrar o valor máximo de  $x^2$  em  $\{0, 1, \dots, 31\}$ .
- Utilizar.
  - Representação binária, ex:  $01101 = 13$ .
  - População de 4 indivíduos.
  - Crossover de 1 ponto (Roleta Russa).
  - Probabilidade de Mutação =  $1/20$ .
  - Probabilidade Cruzamento =  $0.8\%$ .
  - Ranking para a nova população.
- Fazer duas gerações.

# Exercício II

- Resolva o mesmo exercício mas utilizando uma população de 8 indivíduos.
  - Quais impactos você pode observar?

# Algumas Considerações

- Representação de Variáveis
- População: Tamanho e Inicialização
- Operador de Cruzamento
- Operador de Mutação
- Seleção

# Representação de Variáveis

- AGs foram concebidos inicialmente para resolver problemas em representação binária
  - Exemplo anterior (Arranjos binários de tamanho fixo).
- Motivação
  - *Schemata Theory*
  - Maximizar o paralelismo implícito dos AGs
  - Fácil implementação e visualização dos problemas

# Paralelismo Implícito

- Cada indivíduo da população existe como um ente isolado e é avaliado de forma independente.
- Implementação paralela
  - Cada processo avalia um indivíduo da população.
    - *Cluster computing*: Solução barata e eficaz.

# Representação de Variáveis

- No nosso exemplo, a representação que maximiza é 10000000 ( $x = 128$ ).
- A representação de 127 é 01111111
- Como podemos notar, para uma pequena variação do valor de  $x$ , todos os bits da string devem ser modificados.
  - Para uma pequena mudança no valor real, uma grande mudança no valor binário.

# Representação de Variáveis

- Esse tipo de situação não é o ideal.
  - Torna a busca mais lenta.
- Suponha que os limites da variável  $x$  sejam [2.500-2.600]
  - Três casas de precisão.
  - String binária de 12 posições
- Agora considere que o problema possui 100 variáveis.

# Representação de Variáveis

- Isso nos levaria a uma string binária de 1200 posições.
- Impactos:
  - Quanto maior o tamanho da string, maior deve ser a população
  - Consequentemente, a complexidade computacional aumenta.
- Melhor empregar uma codificação real
  - Diferentes operadores de cruzamento e mutação.

# População

- Tamanho da população tem relação direta com o espaço de busca.
  - Quanto maior a população, maior será a busca realizada pelo algoritmo.
- População pode variar de 20 a 200
  - Depende da complexidade do problema em questão.

# Inicialização

- Geralmente a população é inicializada de maneira estocástica.
- Em alguns casos é interessante inserir alguns indivíduos conhecidos.
- Faz com que o algoritmo procure em regiões promissoras
  - Melhor tempo de convergência.

# Operador de Cruzamento

- Cria novos indivíduos através da combinação de dois ou mais indivíduos.
  - Idéia
- >>> Troca de informações entre informações candidatas <<<<
  - O melhor de dois indivíduos **pode** ser combinado.
- Operador mais utilizado é o de 1 ponto.
  - Similar ao que vimos anteriormente mas somente com um ponto de corte.

# Operador de Cruzamento

- Cruzamento Uniforme
  - Para cada bit dos filhos, é decidido com uma probabilidade  $p$ , qual pai vai contribuir para aquela posição.
  - Troca bits e não segmentos de bits.
- Qual utilizar??
  - Geralmente o operador depende do problema

# Operador de Cruzamento

- Os operadores vistos até então também podem ser aplicados em codificações com ponto flutuante.
- Entretanto, alguns operadores têm sido especialmente usados para esse problemas:
  - Cruzamento aritmético
  - Combinação linear de dois cromossomos:

# Operador de Cruzamento

- Dois indivíduos selecionados  $x_1$  e  $x_2$
- Os filhos resultantes serão:

$$x'_1 = ax_1 + (1 - a)x_2$$

$$x'_2 = (1 - a)x_1 + ax_2$$

Onde  $a$  é um número aleatório no intervalo  $[0,1]$

- Interessante para problemas de otimização com restrições onde a região factível é convexa.
- Isso porque se  $x_1$  e  $x_2$  pertencem a região factível, os filhos também pertencerão.

# Operador de Mutação

- Idéia
  - >>> Criar variabilidade na população mas sem destruir o progresso já obtido na busca <<<
- Geralmente aplica-se taxas de mutação maiores no início da busca
  - Busca global (*Exploration*)
- A medida que o algoritmo evolui, a taxa é decrescida
  - Busca local (*Exploitation*)

# Operador de Mutação

- Qual é o impacto de utilizar altas taxas de mutação durante todo o processo?
- O AG ainda é eficiente?

# Operador de Mutação

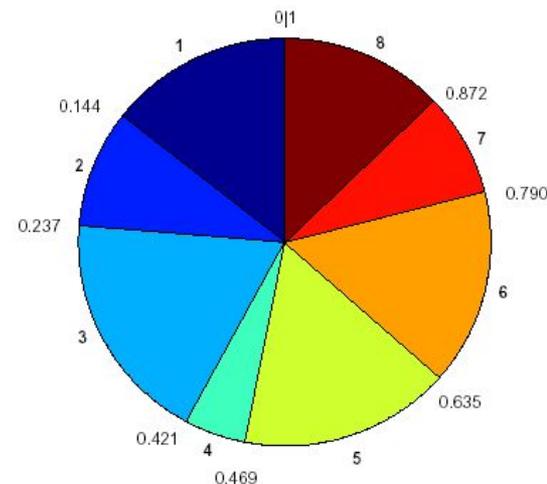
- Codificação com ponto flutuante:
  - Mutação uniforme
    - Seleciona um gene do cromossomo aleatoriamente e atribui um número aleatório com distribuição de probabilidade uniforme amostrado no intervalo [UB-LB].
    - >> Somente um gene é modificado <<

# Operador de Mutação

- Codificação com ponto flutuante:
  - Mutação Gaussiana
    - Todos os genes do cromossomo são modificados na forma  $x' = x + N(0, \sigma)$  onde  $N(0, \sigma)$  é o vetor de variáveis aleatórias Gaussianas independentes com média zero e desvio padrão  $\sigma$

# Seleção

- Método mais empregado
  - Roleta: Quanto maior a fitness, maior a probabilidade dele passar para a próxima geração.
  - Entretanto, pode acontecer que o melhor indivíduo não seja selecionado.
    - Processo estocástico.
    - Similar a natureza
    - Nem sempre o mais forte sobrevive



# Seleção

- Seleção Elitista
  - Garante que o melhor indivíduo vai estar presente na próxima geração.
- Ranking
  - Ordena os indivíduos pela *fitness* e seleciona os melhores para a próxima solução
  - Garante o elitismo
    - >>> **Menor diversidade** <<<

# Porque Funciona

- Como vimos até agora, o funcionamento dos AGs envolvem basicamente cópia e troca de porções de strings, e alterações de bits.
- Teorema que explica o funcionamento
  - *Schema Theorem*
  - Permite a se referir de uma forma compacta às similaridades dos cromossomos.

# Porque Funciona

- Esquema (*schema*; plural *schemata*)
  - Template para descrever os cromossomos
  - Ex: \*1 serve para 01 11. Ou seja, 01 e 11 são instâncias de \*1
  - Melhores esquemas tendem a perpetuarem-se através das gerações.
  - Os esquemas que servem como base para a construção de futuras gerações são chamados de *building blocks*

# AGs Multi-Objetivos

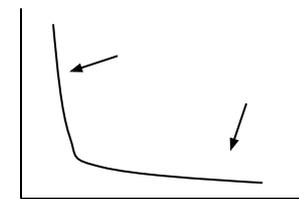
- Em muitos problemas do mundo real, várias funções de objetivos devem ser atualizadas ao mesmo tempo.
  - Ex: custo e conforto.
- A função de fitness  $F(x)$  não é dada somente por uma função  $f$  mas sim por várias  $f_i$ .
- Geralmente um processo de **escala** deve ser utilizado, pois geralmente as funções não apresentam resultados na mesma escala.
  - Custo em reais
  - Conforto por algum índice específico.

# AGs Multi-Objetivos

- Além disso, precisamos combinar os objetivos em um único valor de fitness  $F(x)$

$$F(x) = \sum_{i=1}^N \omega_i f_i(x)$$

- Onde  $\omega_i$  é o peso atribuído para cada objetivo.
- Problema: **Convergência prematura em função dos pesos escolhidos**



Convergência em função dos pesos

# Dominância

- Em um problema multi-objetivos as soluções podem ser expressas em termos de pontos dominantes ou superiores.
- Em um problema de minimização
  - $x_1$  é parcialmente menor que  $x_2$  quando nenhum valor de  $x_2$  for menor que  $x_1$  e pelo menos um valor de  $x_2$  é maior que  $x_1$
  - Se  $x_1$  é parcialmente menor  $x_2$ , então  $x_1$  **domina**  $x_2$

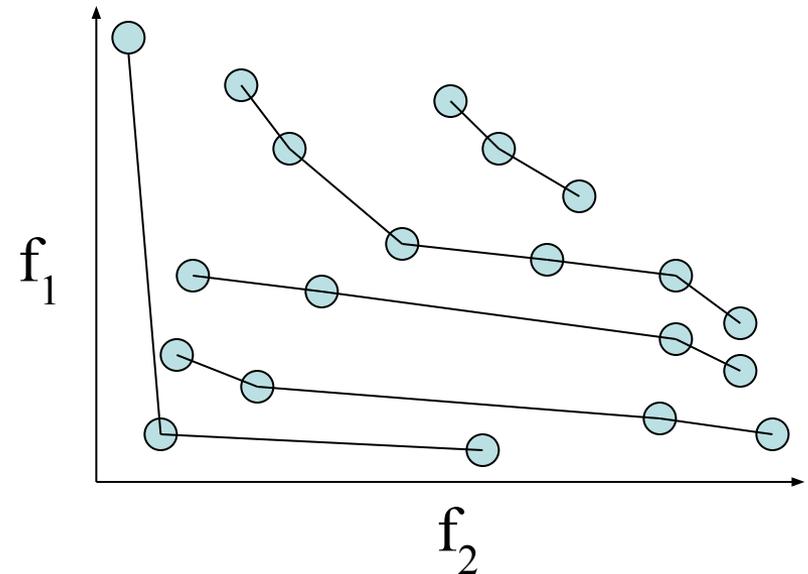
# Dominância

- A idéia é utilizar esse conceito para determinar a probabilidade de reprodução dos indivíduos.
- Ou seja, ao invés de utilizarmos diretamente o valor da fitness, utilizamos o conceito de dominância.
  - Os dominantes têm mais chances de se reproduzir.
- As soluções ótimas são as soluções não dominadas, também conhecidas como **Pareto-ótimas**.

$$\begin{aligned} \forall i \in \{1, 2\} : f_i(x^{(1)}) \leq f_i(x^{(2)}) \quad \wedge \\ \exists j \in \{1, 2\} : f_j(x^{(1)}) < f_j(x^{(2)}) \end{aligned}$$

# Ranking by Fronts

- Atribuir *rank* 1 para os indivíduos não dominados,
- Removê-los da população.
- Encontrar novos indivíduos não dominados,
- Atribuir *rank* 2,
- E assim por diante...

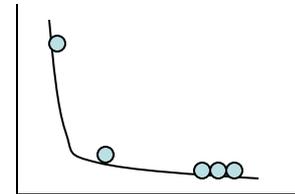


# *Ranking by Fronts*

- Essa estratégia fornece aos indivíduos do mesmo ranking um mesmo valor de fitness.
- Porém não garante que o Pareto seja uniformemente distribuído.
- Quando existem várias soluções ótimas, a população tende a convergir somente para uma delas.
  - Isso deve-se a erros estocásticos no processo de seleção (*Genetic Drift*)

# Mantendo a Diversidade (Sharing)

- Como evitar esse tipo de problema?
  - Compartilhar o valor de alguns indivíduos.
- Quais?
  - Aqueles pertencentes ao nichos mais populosos.
- Desta maneira, aqueles pertencentes a nichos menos populosos, também terão chances.



# Diversidade

- A intenção do compartilhamento é criar diversidade evitando assim a convergência prematura para algum ponto do espaço de busca.

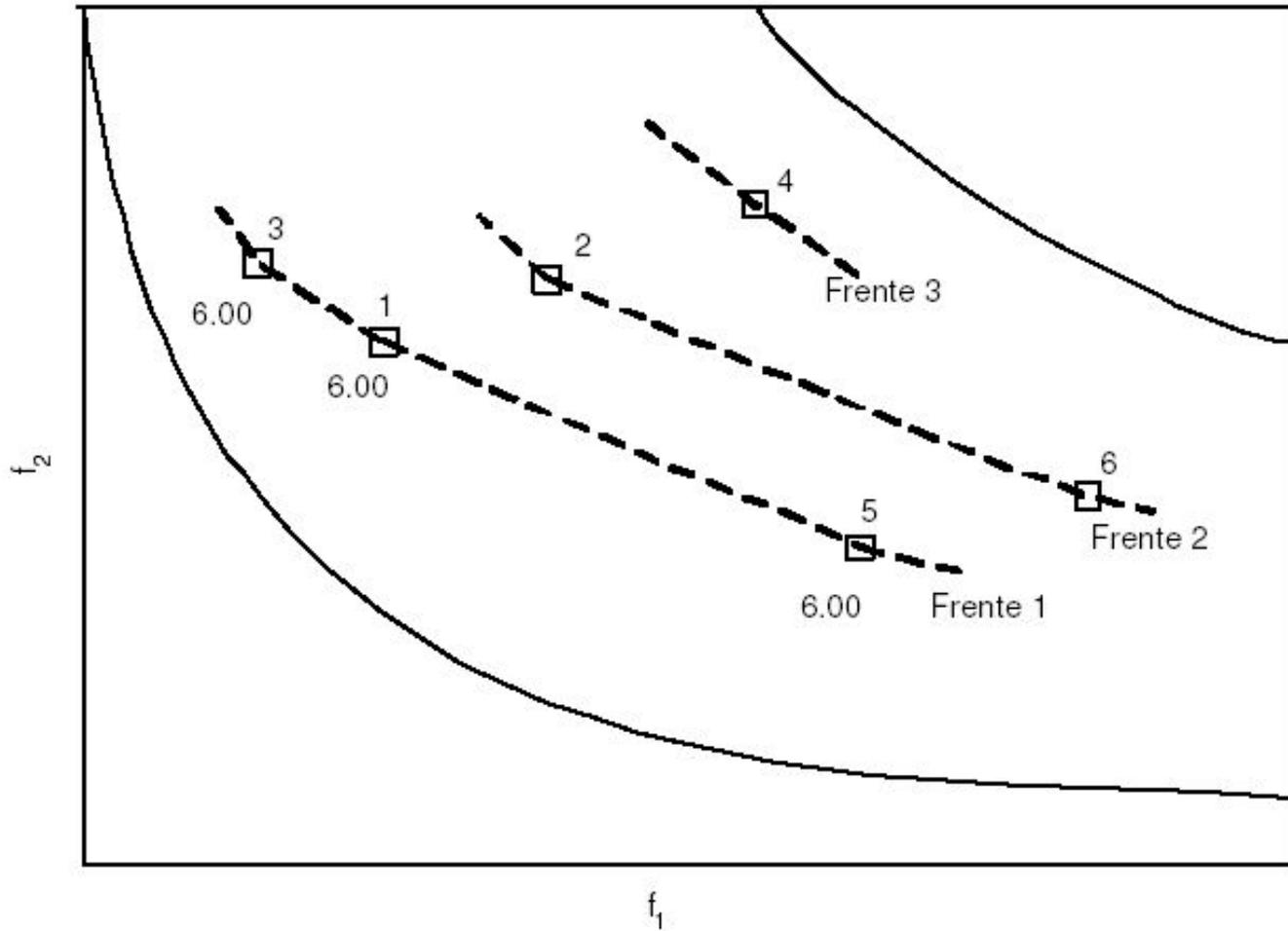
# Non-Dominated Sorting GA (NSGA)

- Um dos algoritmos propostos para otimização multi-objetivos
  - [Srinivas&Deb 95].
  - Utiliza o conceito de *ranking* apresentado anteriormente.
  - Difere do GA clássico somente na maneira em que os indivíduos são selecionados
    - Operador de Seleção.

# NSGA

- A seleção é realizada com base na não-dominância
  - Todos os indivíduos da primeira frente (rank 1) recebem a mesma fitness (dummy fitness)
  - Isso garante a mesma chance de reprodução para todos os indivíduos.

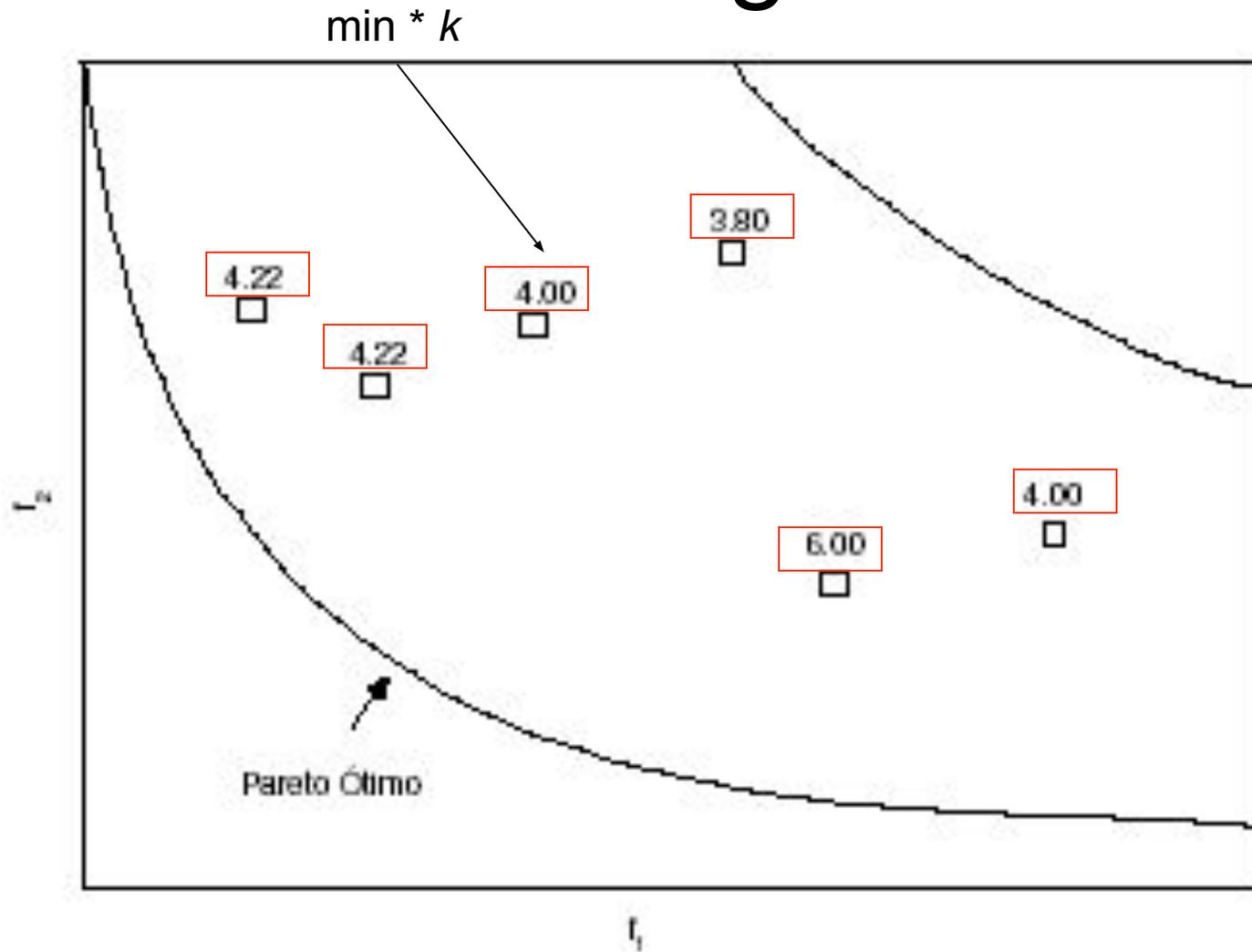
# Ordenação



# Diversidade (Sharing)

- Diversidade
  - É mantida através do compartilhamento da fitness daqueles indivíduos mais próximos
    - Que fazem parte do mesmo nicho.
  - Após o compartilhamento, os indivíduos são ignorados e o restante da população é processado da mesma maneira.

# Sharing



*dummy fitness*

# NSGA

- A população é reproduzida de acordo com o valor das *dummy fitness*, através de qualquer método de seleção visto anteriormente
  - Roleta russa, ranking, etc...
- >>> **Converte n objetivos em um simples valor de fitness** <<<
- Resolve os problemas de escala e sensibilidade em direção dos pesos.

# Exercício

- Faça o “ranking by fronts” dos seguintes pontos e identifique quais os pontos que devem ter suas *fitness* compartilhadas.
  - Critério de compartilhamento:
    - $De(X1, X2) > 0.3$

ISC	Vendas	ISC	Vendas
7	1000	3,1	4000
6	3800	2,1	5000
5	1100	1,2	6000
4,5	1900	1,1	6500
4	5500	1,2	6800
3	2000		

# Aplicações

- Os AGs são utilizados para resolver um grande gama de problemas.
  - Problemas difíceis
    - Espaços de busca mal-compreendidos.
    - Funções desconhecidas e descontínuas.
- Foco principal em problemas de otimização.

# Exemplo Prático

- Data Mining: Minerar uma grande base de dados para identificar o perfil dos consumidores.
  - Os indivíduos podem representar regras de previsão.
  - A *fitness* mede a quantidade de regras associadas com os indivíduos.
- Exemplo:
  - Representar uma regras para prever quando um cliente comprará um produto oferecido a ele.
    - (idade < 18) e (produto = videogame)

# Exemplo Prático

- Os fatores que podem ser medidos:
  - Número de tuplas selecionadas na base de dados.
  - Generalidade: número de tuplas coberta pela regra
  - Múltiplos objetivos poderiam ser utilizados utilizando NSGA por exemplo.

# Exemplo Prático

- Nesse contexto, o cruzamento poderia funcionar da seguinte maneira:

Antes do cruzamento:

```
(idade < 18)(produto = videogame)      |(sexo=F)  
(idade < 35)(produto = liquidificador)|(sexo=M)
```

Após o cruzamento:

```
(idade < 18)(produto = videogame)      (sexo=M)  
(idade < 35)(produto = liquidificador)(sexo=F)
```

# Exemplo Prático

- A mutação, mudando um gene por exemplo:

Antes da mutação:

```
(idade < 18) (produto = liquidificador) (sexo=M)
```

Após a mutação:

```
(idade < 18) (produto = videogame)      (sexo=M)
```

# Exemplo Prático

- Calculando a fitness

`(idade > 65) (produto = videogame) (sexo=F) (compra=SIM)`



`(idade < 18) (produto = videogame) (sexo=M) (compra=SIM)`



# Exercício

- Como esse exemplo poderia ser codificado utilizando uma string binária?

# Exercício

- Considere o exercício com a população de 8 indivíduos de dimensão cinco.
  - Suponha que o objetivo agora seja encontrar o máximo valor de  $f(x)$ , mas ao mesmo tempo minimizar o número de dígitos 1 no cromossomo.
  - Utilize o AG multi-objetivo.
  - Compartilhe a dummy-fitness quando
    - $De(x_1, x_2) < 5$
    - Divida a dummy fitness pelo numero de elementos no nicho.

# Laboratório

- <http://homepage.sunrise.ch/homepage/pglaus/gentore.htm>
  - Fractal
- <http://www-cse.uta.edu/~cook/ai1/lectures/applets/gatasp/TSP.html>
  - Caixeiro viajante - applet java
- <http://math.hws.edu/xJava/GA/>
  - Applet de um mundo artificial
- [http://userweb.elec.gla.ac.uk/y/yunli/ga\\_demo/](http://userweb.elec.gla.ac.uk/y/yunli/ga_demo/)
  - Simulador de AG.
- Programa executável disponível na página do curso.

# Particle Swarm Optimization

# Objetivos

- Introduzir os principais conceitos da inteligência de enxame.
- Apresentar PSO
- Diferenças entre PSO e a computação evolutiva.

# Introdução

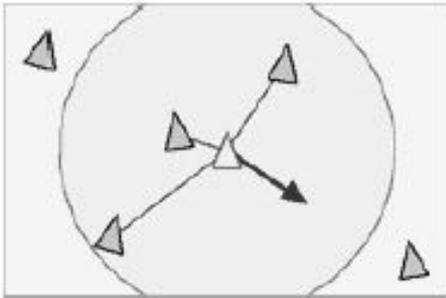
- No início dos anos 90, alguns pesquisadores começaram a fazer analogias entre o comportamento dos enxames de criaturas e problemas de otimização
  - ACO – Ant Colony Optimization
  - PSO – Particle Swarm Optimization
    - Inteligência de Enxame

# Inteligência de Enxame

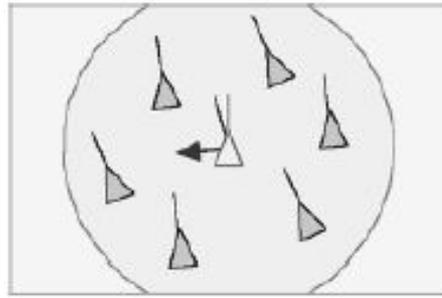
- Enxame:
  - Indivíduos possuem estruturas simples
  - Comportamento coletivo pode ser complexo
  - Relacionamento entre o comportamento do indivíduo e o comportamento do enxame através de interações (cooperação) entre os indivíduos.

# Inteligência de Enxame

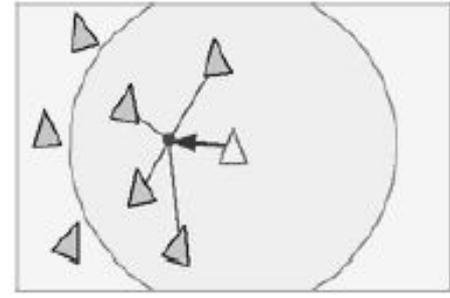
- Cardumes, enxames e revoadas são guiados por três forças:



Separação: Não batem uns nos outros.



Alinhamento: Tentam manter a mesma velocidade dos seus vizinhos.



Direcionamento: seguem a direção do centro da sua vizinhança.

Princípios utilizados em softwares de animações (cinema)

# Princípios Sócio-cognitivos

- Avaliação:
  - A tendência de avaliar um estímulo (positivo ou negativo – atrativo ou repulsivo) é a característica comportamental mais encontrada em organismos vivos.
  - Aprendizagem é impossível se não existe a capacidade de avaliar.

# Princípios Sócio-cognitivos

- Comparação (Teoria de Festinger, 54)
  - Descreve como as pessoas utilizam os outros como padrão de comparação.
  - Em quase tudo que pensamos e fazemos, nós nos julgamos através da comparação com os outros.
  - Na inteligência de exame, os indivíduos tentam seguir (imitar) os melhores.

# Princípios Sócio-cognitivos

- Imitar:
  - Poucos animais são capazes de realizar uma imitação (seres humanos e alguns pássaros)
    - Imitação: Não se trata de imitar o comportamento somente (“*monkey see, monkey do*”), mas entender o seu propósito.
    - Ex: Um macaco pode ver um outro com um determinado objeto e usar esse objeto para um outro propósito.

# PSO

- Tem várias similaridades com as técnicas evolutivas discutidas anteriormente.
- O sistema é inicializado com uma população de soluções aleatórias e busca uma solução ótima através das gerações.
- Diferentemente dos AGs, PSO não conta com operadores evolutivos, tais como cruzamento e mutação.

# PSO

- Comparado aos AGs,
  - Mais fácil e simples de implementar.
  - Exige menos parâmetros ajustáveis.
  - Representação contínua
    - Adaptações para representações binárias.

# Exemplo

- Considere o seguinte cenário:
  - Um grupo de pássaros procurando comida em uma determinada área, a qual tem um único pedaço de comida.
  - Os pássaros não sabem onde está a comida, mas sabem o qual distante a comida está a cada iteração.
  - Qual seria a melhor estratégia para procurar comida.
    - Seguir aquele que está mais próximo da comida.

# Exemplo

- Nesse contexto, PSO aprende a partir do cenário.
- Cada pássaro (partícula) é uma solução potencial.

# Partículas

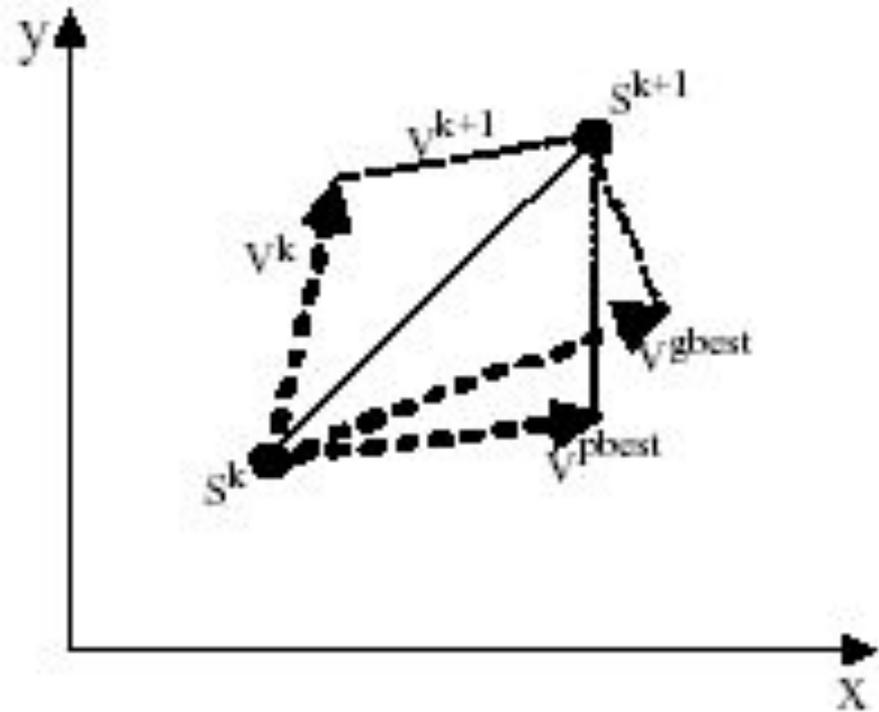
- Por uma questão de simplicidade, vamos considerar um espaço 2D.
- A posição de cada partícula é representada por  $(x,y)$ .
- As velocidades nos eixos  $x$  e  $y$  são representadas por  $v_x$  e  $v_y$ , respectivamente.
- A modificação da partícula é realizada com base nas variáveis de velocidade.

# Movimentação no Espaço

- A cada geração, cada partícula é atualizada com base em dois valores.
  - A melhor posição que ela encontrou (*pbest*)
  - A melhor posição de todas as outras partículas (*gbest*).
- Cada partícula tenta modificar a sua posição levando em consideração as seguintes informações:

# Movimentação no Espaço

- Sua posição corrente
- As velocidades correntes
- A distância entre a posição corrente e *pbest*
- A distância entre a posição corrente e *gbest*



# Movimentação no Espaço

- Essa movimentação se dá através da seguinte equação :

Nova  
velocidade

$c_1$  e  $c_2$  – Fatores de aprendizagem

$$v_i^{k+1} = wv_i^k + c_1 \text{rand}() \times (pbest_i - s_i^k) + c_2 \text{rand}() \times (gbest - s_i^k)$$

Velocidade

Posição corrente

$w$  – função de peso  
 $w$  grande – exploração global  
 $w$  pequeno – exploração local

Melhor posição  
desta partícula.

Melhor posição  
de todas as  
partículas.

# Movimentação no Espaço

$$s_i^{k+1} = s_i^k + v_i^{k+1}$$

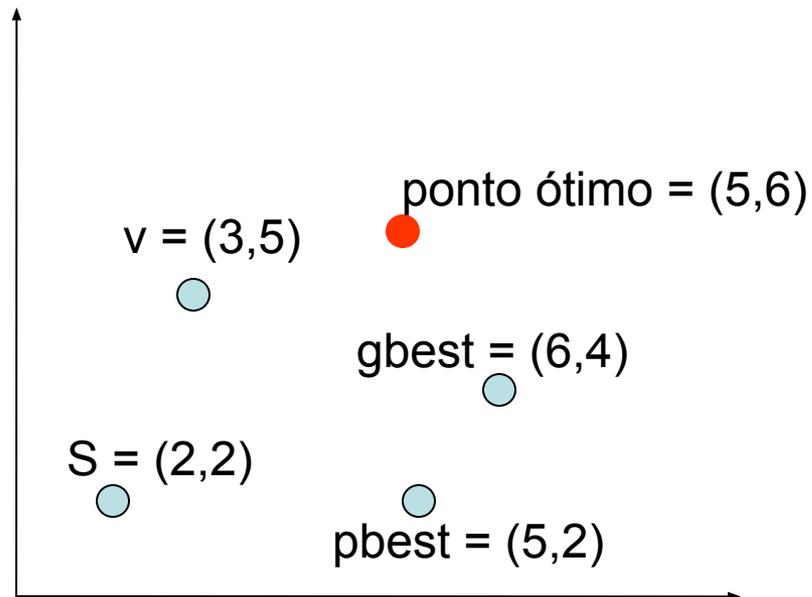
Nova posição da  
partícula

Posição  
anterior

Velocidade  
calculada com a  
equação anterior

# Exercício

- Calcule a nova posição da partícula
  - Teste para  $w = 0.9$  e  $0.1$



```
Para cada partícula
  Inicialize a partícula
Fim
```

```
Faça
  Para cada partícula
    Calcule o valor de fitness
    Se o valor de fitness  $\geq$  pbest
      pbest = valor da fitness
  Fim
```

```
gbest = melhor partícula
```

```
Para cada partícula
  Calcule a velocidade da partícula
  Atualize a posição da partícula
Fim
```

Enquanto não atingir o critério de parada  
ou número máximo de iterações.

# Pontos em Comum com AGs

- Ambos começam com populações aleatórias e avaliam a fitness dos indivíduos.
- Buscam o ponto ótimo de maneira estocástica.
- Ambas não garantem o sucesso, embora produzam bons resultados na maioria dos casos.

# Diferenças

- A maneira pela qual se compartilha informação é diferente.
  - AG – cruzamento
  - PSO – somente gbest fornece informação.
- Experimentos tem mostrado que PSO converge mais rápido que AGs

# Controlando Parâmetros

- Número de partículas
  - Geralmente entre 20 e 40. Muitas vezes 10 é suficiente
  - Dimensão
    - Depende do problema assim como nas técnicas evolutivas.
  - Domínio
    - Valores máximo e mínimo para as partículas
  - Fatores de aprendizagem
    - C1 e C2 – Geralmente igual a 2. Entretanto, outros valores entre [0,4] podem ser utilizados.



# Referências