Computação Evolutiva

MC 906 – Prof. Anderson Rocha

Organização dessa aula

Sumário

- Como resolver problemas
- Comp. Evolutiva (CE)
- Para que serve
- Como simular
- Abordagens de CE
- Problemas comuns
- Exemplos
- Próximos capítulos
- Lição

Resolvendo problemas...

Como resolver problemas

- Técnicas para resolver problemas classificadas em
  - Métodos fortes: problemas genéricos mas operando em um mundo específico onde impera linearidade, continuidade, diferenciabilidade. Ex.: busca iterativa, gradiente etc.

Como resolver problemas

- Técnicas para resolver problemas classificadas em
  - Métodos específicos: problemas específicos em mundos específicos. Ex.: toda técnica que conduz a uma solução na forma fechada como a resolução de sistemas lineares

Como resolver problemas

- Técnicas para resolver problemas classificadas em
  - Métodos fracos: problemas genéricos em mundos genéricos. Operam em mundos não lineares e não estacionários. Não garantem eficiência total na solução

Computação Evolutiva

Computação Evolutiva

- Teoria da Evolução Natural – C. Darwin

A vida na terra é o resultado de um processo de seleção, pelo meio ambiente, dos mais aptos e adaptados, e por isso mesmo, com mais chances de reproduzir-se.

Computação Evolutiva

- Paradigma para resolver problemas
- Não exige, para resolver um problema, um conhecimento prévio de uma maneira de encontrar a solução
- Baseada em mecanismos evolutivos da natureza: auto-organização e comportamento adaptativo

Computação Evolutiva

- Em CE abre-se mão da garantia de obtenção de uma solução ótima para se conquistar a tratabilidade via uma ferramenta de propósito geral.

Interessante,
mas para que serve?

CE – Motivação

- Podemos validar teorias e conceitos associados à biologia da evolução
- Podemos lidar com problemas com os quais não é possível ou é muito custoso obter uma descrição detalhada.

CE – Motivação

- Não é necessário reiniciar todo o processo de busca de uma solução frente a pequenas mudanças nas especificações do problema. Por que?

  - Refinamentos podem ser obtidos a partir da solução atual

Como simular este comportamento?

Idéias básicas

- A CE está baseada em algumas idéias básicas
  - População de soluções (e.g., inicialmente aleatória) no qual os indivíduos registrem os parâmetros que descrevem uma possível solução
  - Função de avaliação – julga a aptidão de cada indivíduo. Apenas atribui uma "nota".

Idéias básicas

- Operadores aplicados a uma dada geração para obtenção de uma próxima geração
  - Seleção: permite escolher os indivíduos (rep. assexuada) ou um par deles (rep. sexuada) para gerar descendência.
  - Prioridade de escolha recai sobre os mais bem avaliados

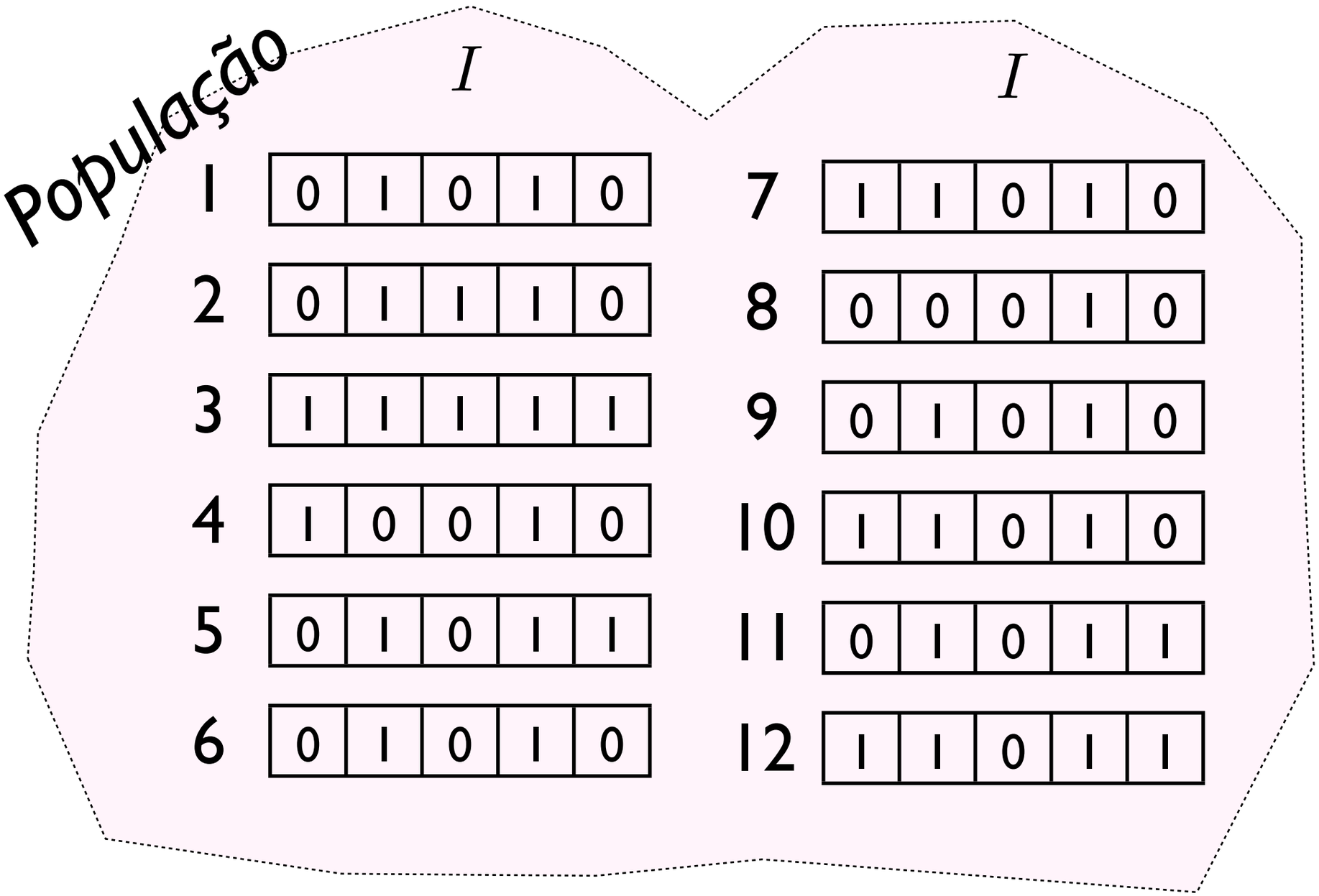
Idéias básicas

- Operadores aplicados a uma dada geração para obtenção de uma próxima geração
  - Recombinação: simula a troca de material genético entre os ancestrais, determina a carga genética dos descendentes
  - Mutação: operador que realiza mudanças aleatórias no material genético

Idéias básicas

- Adaptação: uma população inicial de soluções evolui ao longo das gerações que são simuladas no processo em direção a soluções mais adaptadas por meio dos operadores de seleção, mutação e recombinação.

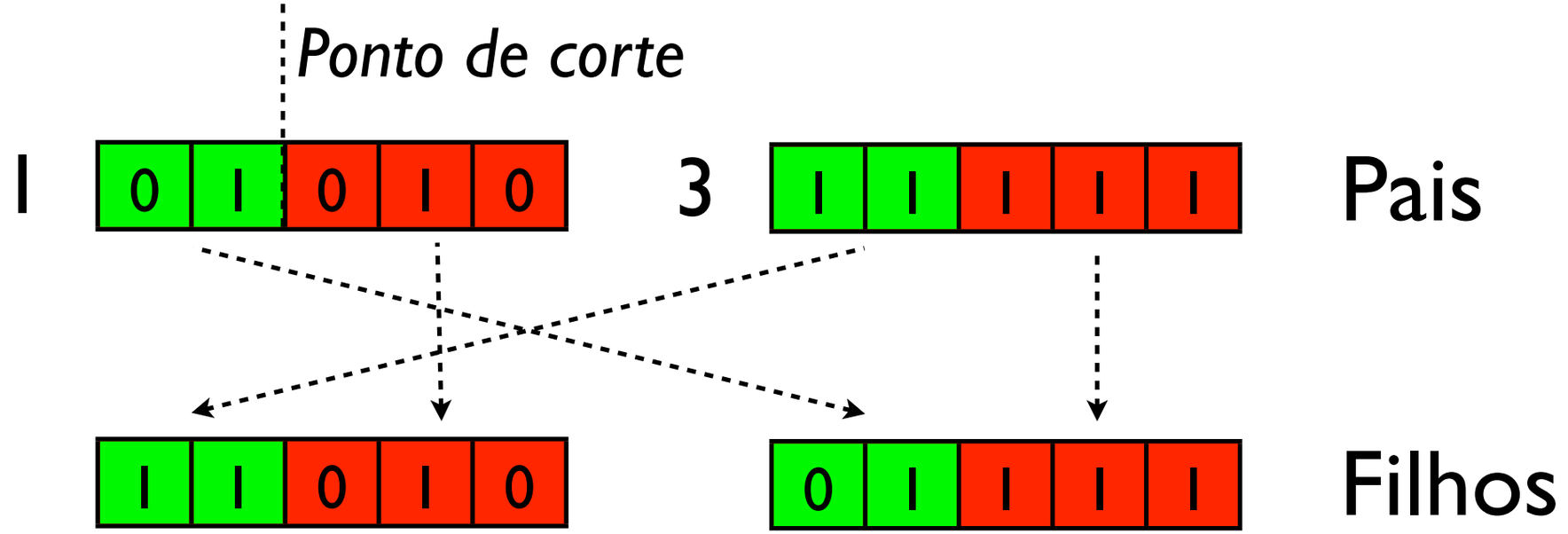
Indivíduos formam População



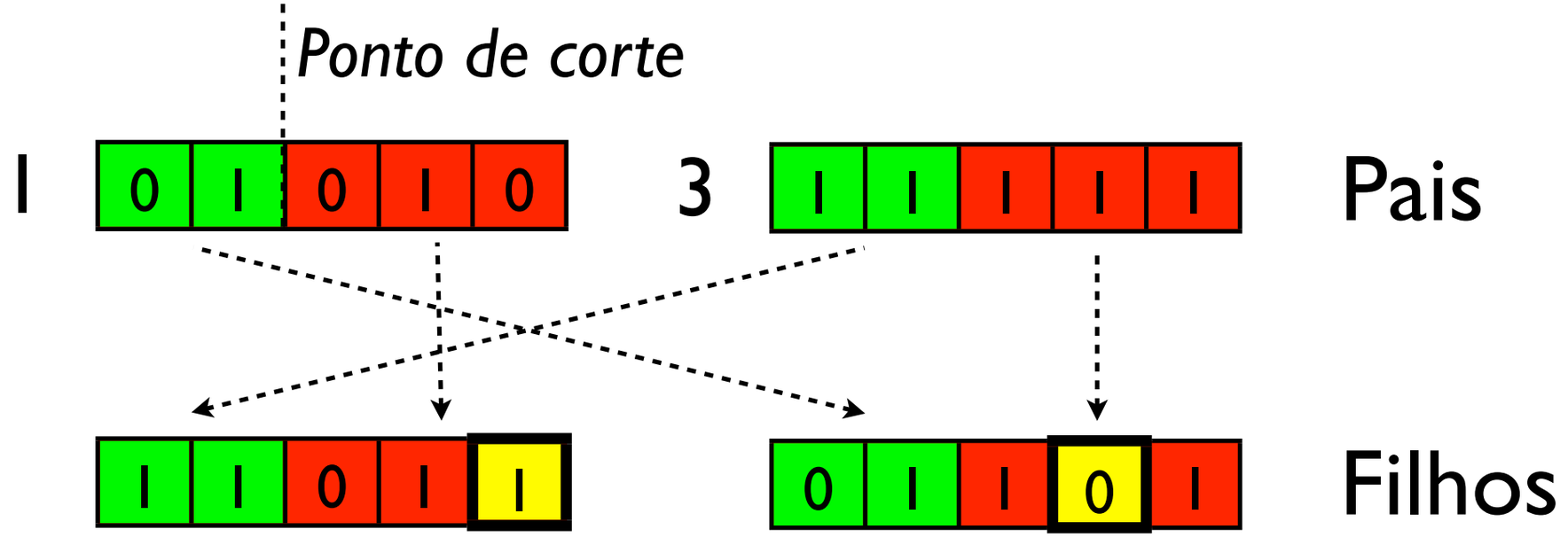
Função de avaliação – *fitness*

	$f(I)$		$f(I)$
1	0 1 0 1 0	7	1 1 0 1 0
2	0 1 1 1 0	8	0 0 0 1 0
3	1 1 1 1 1	9	0 1 0 1 0
4	1 0 0 1 0	10	1 1 0 1 0
5	0 1 0 1 1	11	0 1 0 1 1
6	0 1 0 1 0	12	1 1 0 1 1

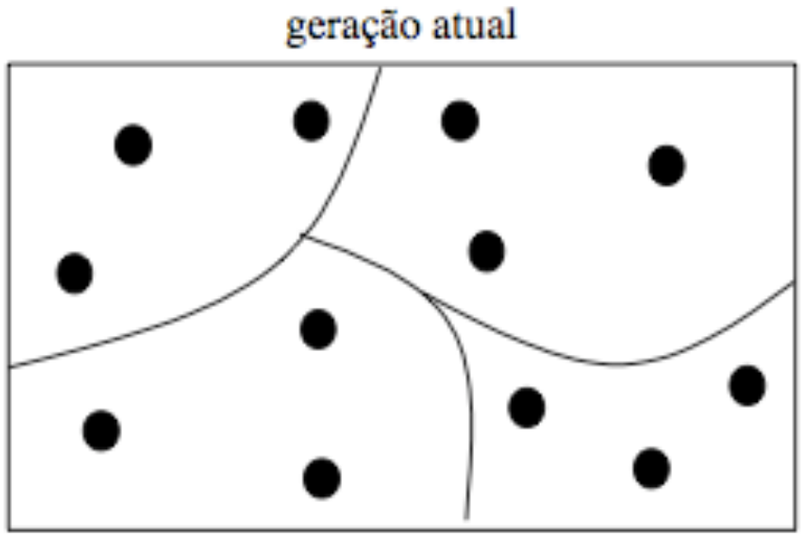
Recombinação

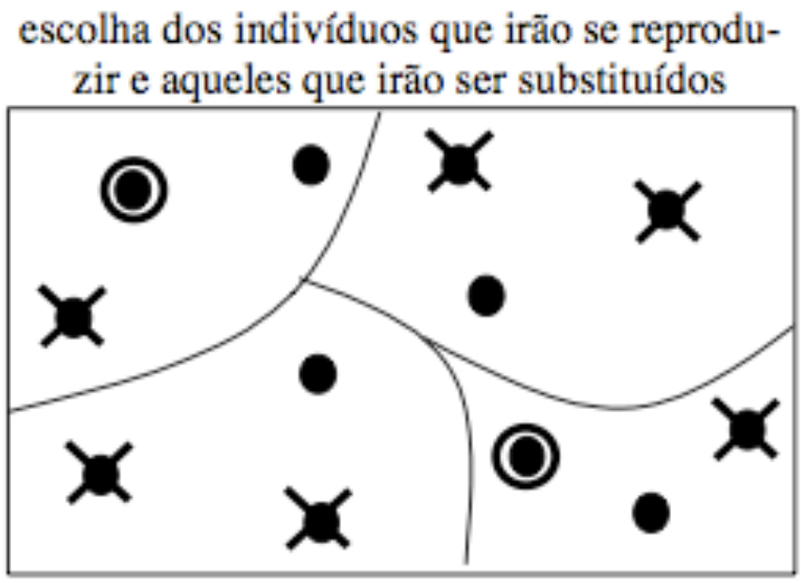


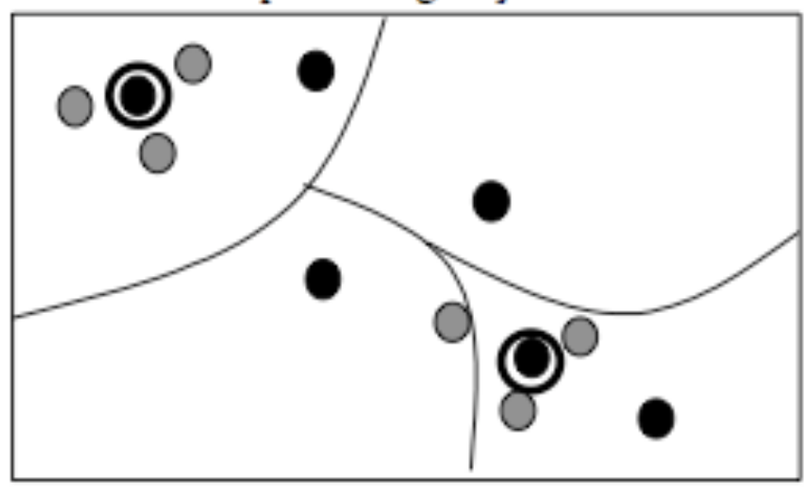
Mutação



Entendendo melhor







Diferentes abordagens de CE

Diferentes abordagens

- Diferentes abordagens de CE
  - Estratégias Evolutivas
  - Programação Evolutiva
  - Algoritmos Genéticos
  - Programação Genética

Estratégia Evolutiva

- Empregam apenas operadores de mutação
- Não necessita de muitas informações sobre o problema
- Muito utilizada em otimização (problemas multi-dimensionais, multi-modais e não lineares)

Estratégia Evolutiva

- Algoritmo básico

1. População com *m* indivíduos. Cada um tem *n* genes
2. Cada indivíduo produz *k*/*m* descendentes com pequenas mudanças (mutações)
3. Apenas os *m* melhores indivíduos dos *k* gerados permanecem vivos.

Programação Evolutiva

- Cada indivíduo gera um único descendente através de mutação
- A melhor metade da população ascendente e a melhor metade da população descendente formam a nova geração

Programação Evolutiva

- Algoritmo básico

1. População inicial escolhida aleatoriamente
2. Cada solução gera uma nova solução utilizando-se mutação
3. Calcula-se aptidão de cada solução. Os mais aptos são retidos.

Algoritmos genéticos

- Combinam variações aleatórias com seleção de indivíduos mais aptos
- AG e AE mantém uma população de soluções candidatas
- Busca multi-direcional e paralelismo
- Recombinação e Mutação

Algoritmos genéticos

- Algoritmo básico

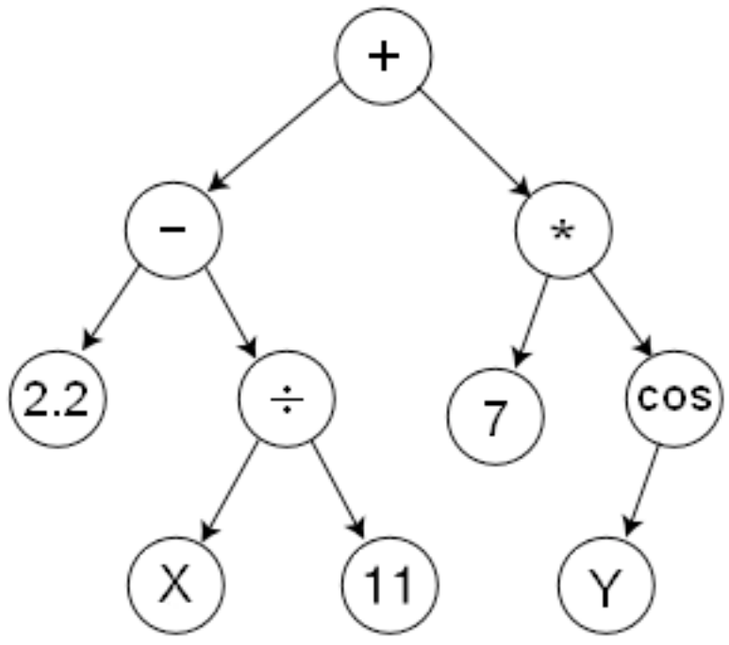
1. Inicializar população (soluções candidatas)
2. Avaliar cada cromossomo (solução)
3. Criar novos cromossomos a partir da população atual (mutação + recombinação)
4. Substituir ascendentes por descendentes
5. Se atingir critério de parada, terminar.

Programação genética

- Extensão de Algoritmos Genéticos
- Indivíduos são programas
- Representação comum: árvores

Programação genética

Indivíduo em PG

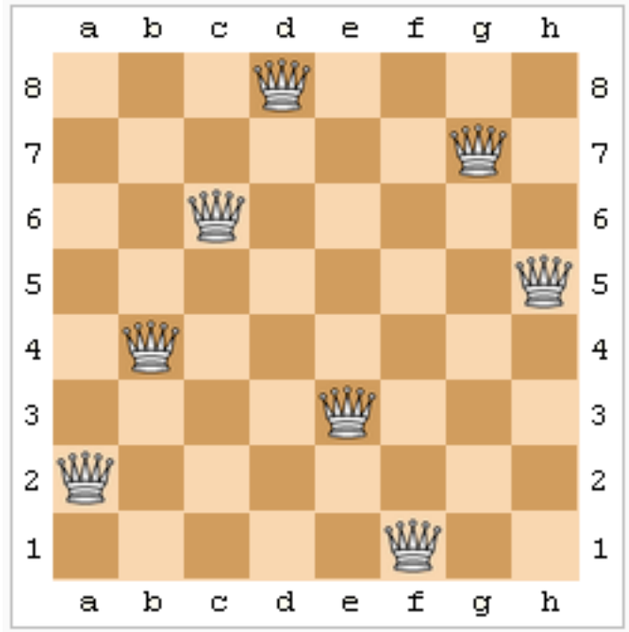


$$\left(2.2 - \left(\frac{X}{11}\right)\right) + \left(7 * \cos(Y)\right)$$

Problemas comuns

- Codificação dos indivíduos – binária, ponto flutuante etc.

- Codificação errada pode levar a problemas de convergência prematura e valores inválidos (fora do domínio)



(2,4,6,8,3,1,7,5)

Problemas comuns

- Como criar uma população inicial? Aleatório ou usar um método simples?
- Operadores genéticos
  - Quais os seus parâmetros? Mutação deve ser de valor alto ou baixo?
  - Como deve ser a recombinação?

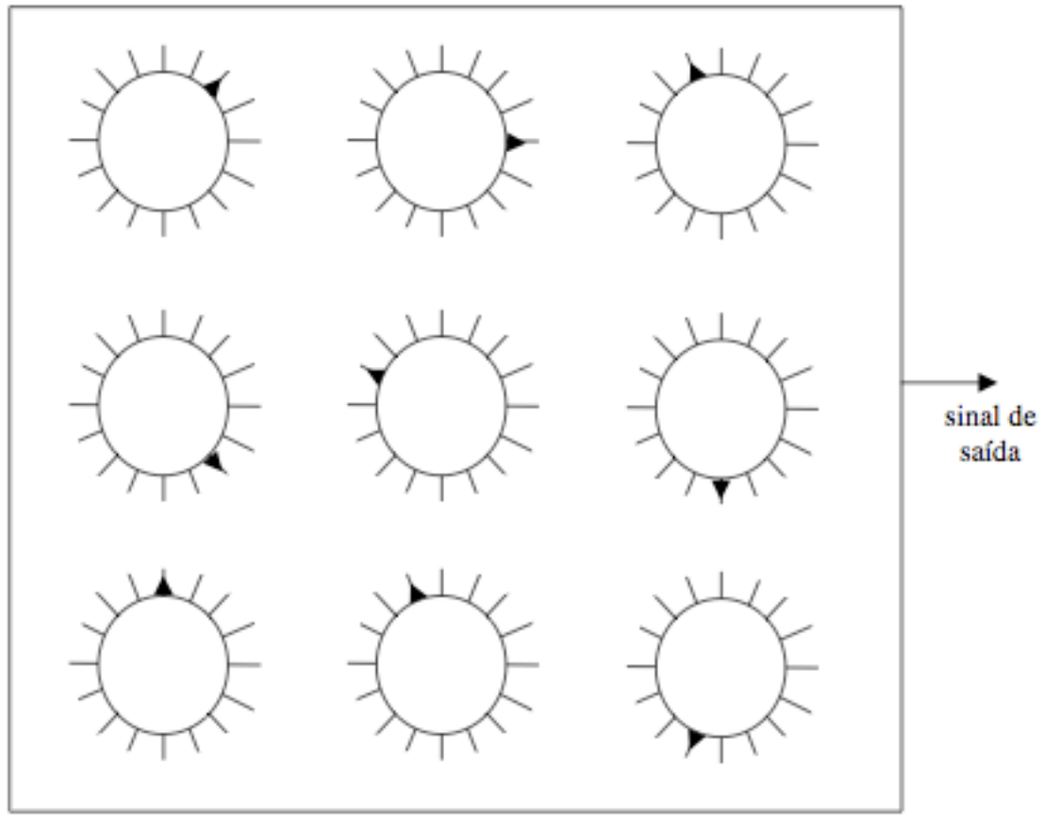
Problemas comuns

- Como selecionar indivíduos para a próxima geração?
  - Modelo roleta – probabilidade proporcional ao *fitness*. Pode perder melhor indivíduo. Solução: usar elitistismo
  - Seleção baseada em rank – ordena pelo *fitness*

Ok... mas tem algum exemplo real?

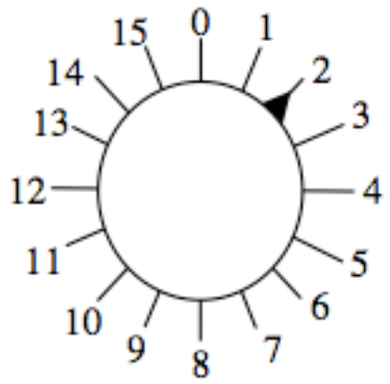
Problema 1: voltagem

- 16 posições
- 9 botões
- Encontrar combinação que maximiza sinal de saída



$16^9 \sim 68{,}7 \times 10^9$

Problema 1: Codificação

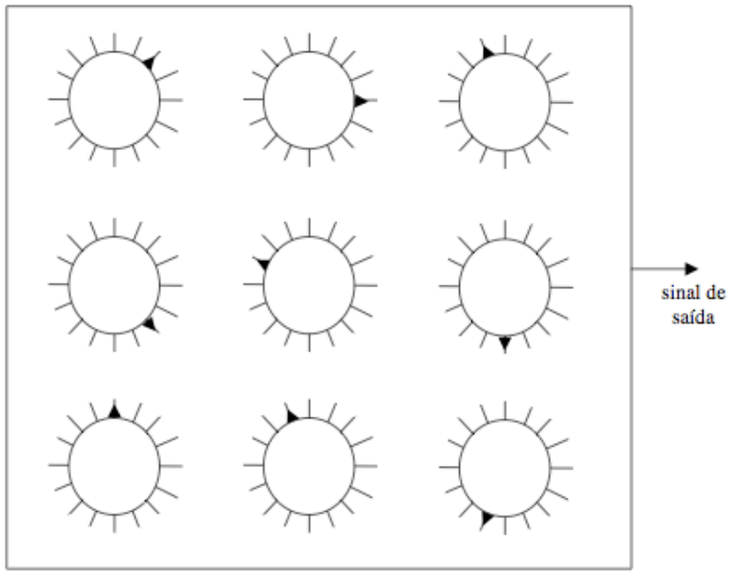


Posição	Representação	Posição	Representação
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

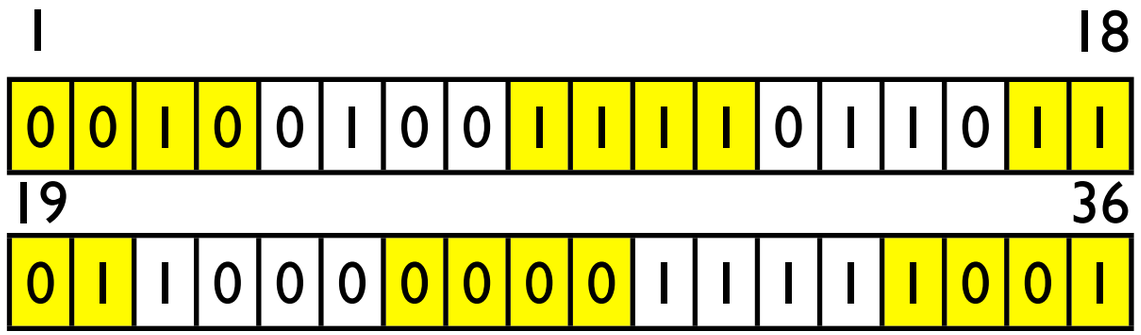
Posição atual: 0010

- 16 posições possíveis: 4 bits
- Indivíduo 4 bits * 9 botões = 36 bits

Um indivíduo



Cromossomo (solução candidata)

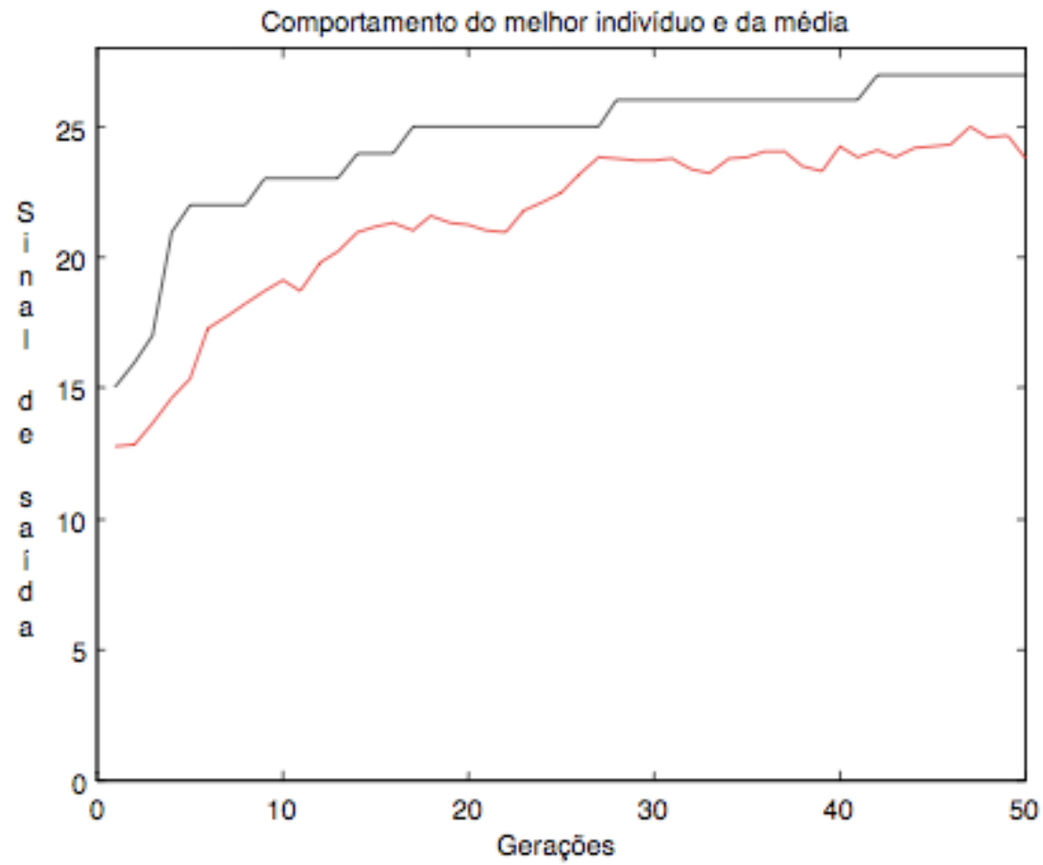


- Operadores genéticos – mutação simples e recombinação uniforme
- Que é recombinação uniforme? Para cada posição, escolhe-se com certa prob. se pai X ou Y é que contribui

Adaptação

- Função de adaptação: voltagem da saída
- Valores arbitrários
  - Prob. de bits 1 na população inicial: 50%
  - Taxa de mutação: 3%
  - Recombinação: 60%
  - Tipo de seleção na recombinação: bi-classista (50% bons, 10% ruins)

Resultado



Ponto ótimo

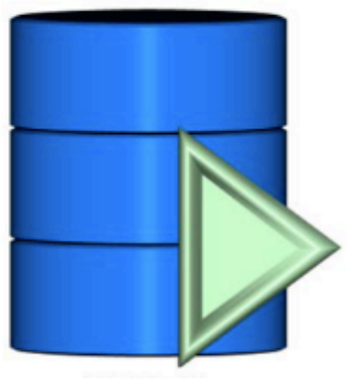
- 1500 indivíduos apenas. Quantos eram mesmo? ~10^9
- Resultados em menos de 1 segundo!

Problema 2: CBIR

Consulta

Descrição

Modelo Resultante



Banco de Dados

Modelos do Banco de Dados

Comparação

Resultado

Problema 2: CBIR

- Agora suponha que cada imagem possa ser descrita de alguma forma por cor, forma e textura

Cor: (1, 20, 50, 50, 30, 25)

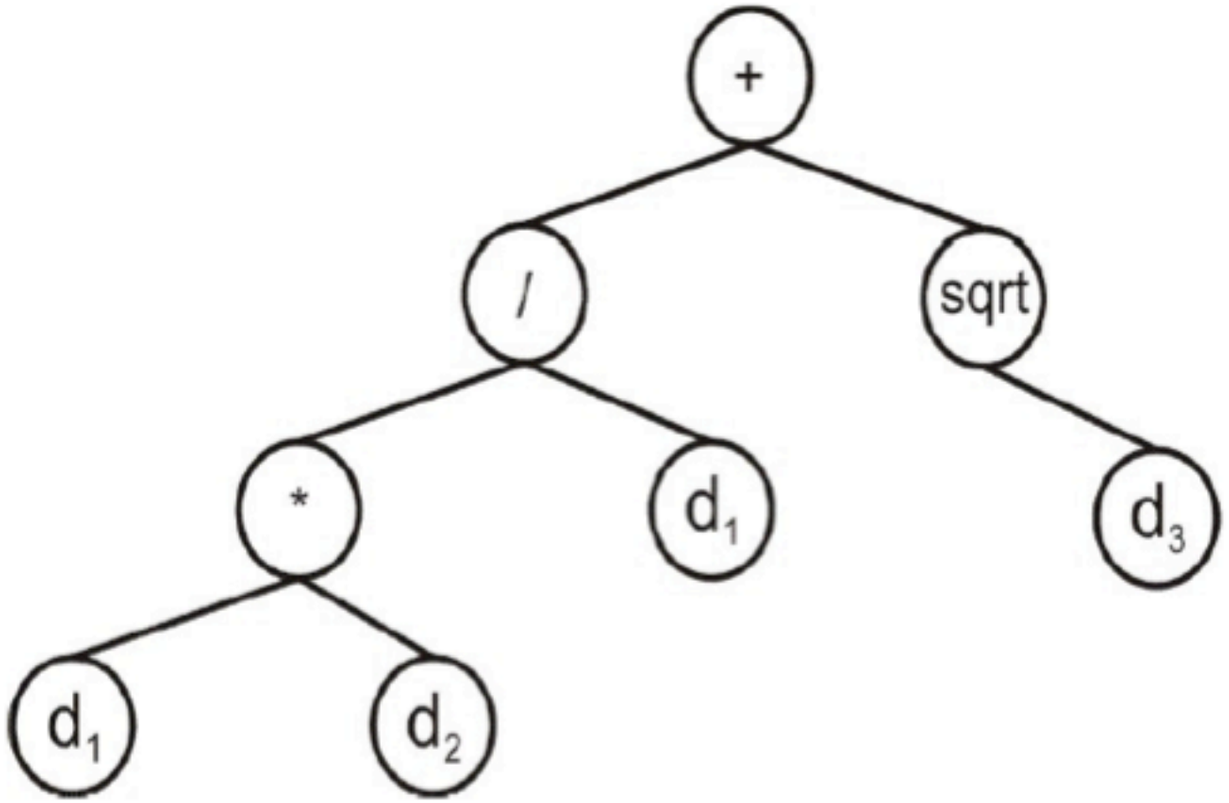
Forma: (0, 1, 1, 0, 1, 1)

Textura: (234, 50, 45, 11, 13, 14)

Como combinar?

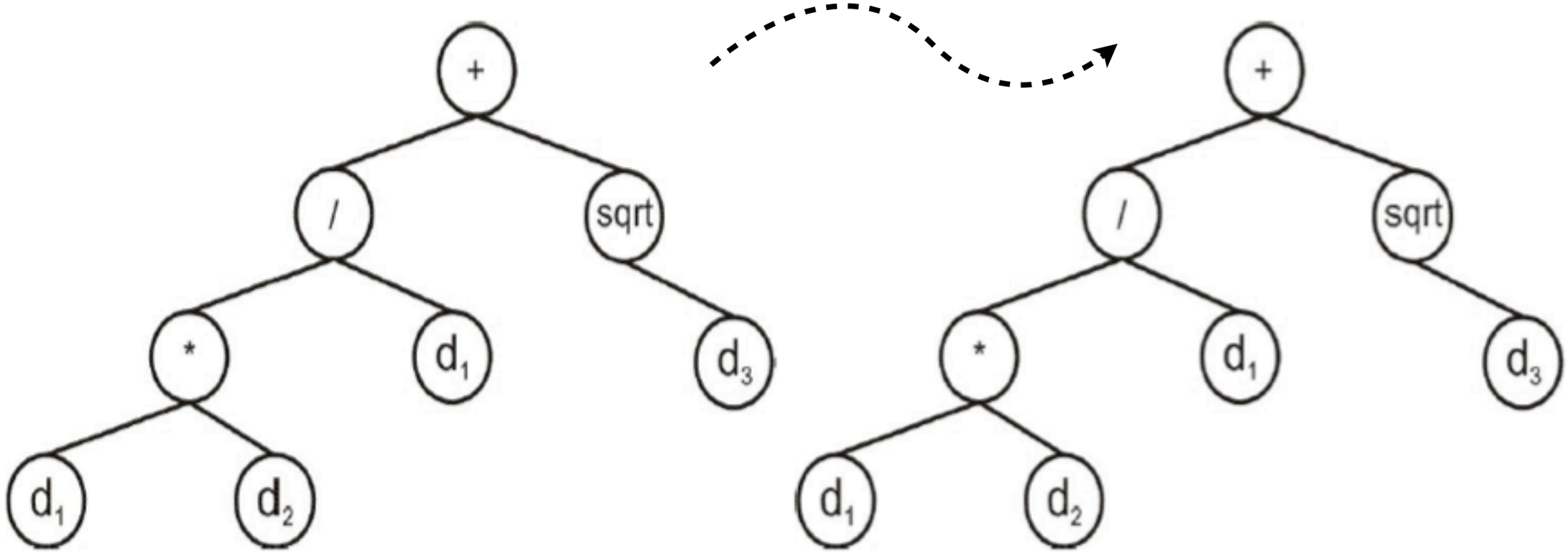
- Dado esses números que representam as imagens, como combiná-los? Soma, subtração, raiz, log, multiplicação, etc.
- Como conseguir boas combinações?
- Resposta: Programação Genética

PG e Recuperação de Imagens

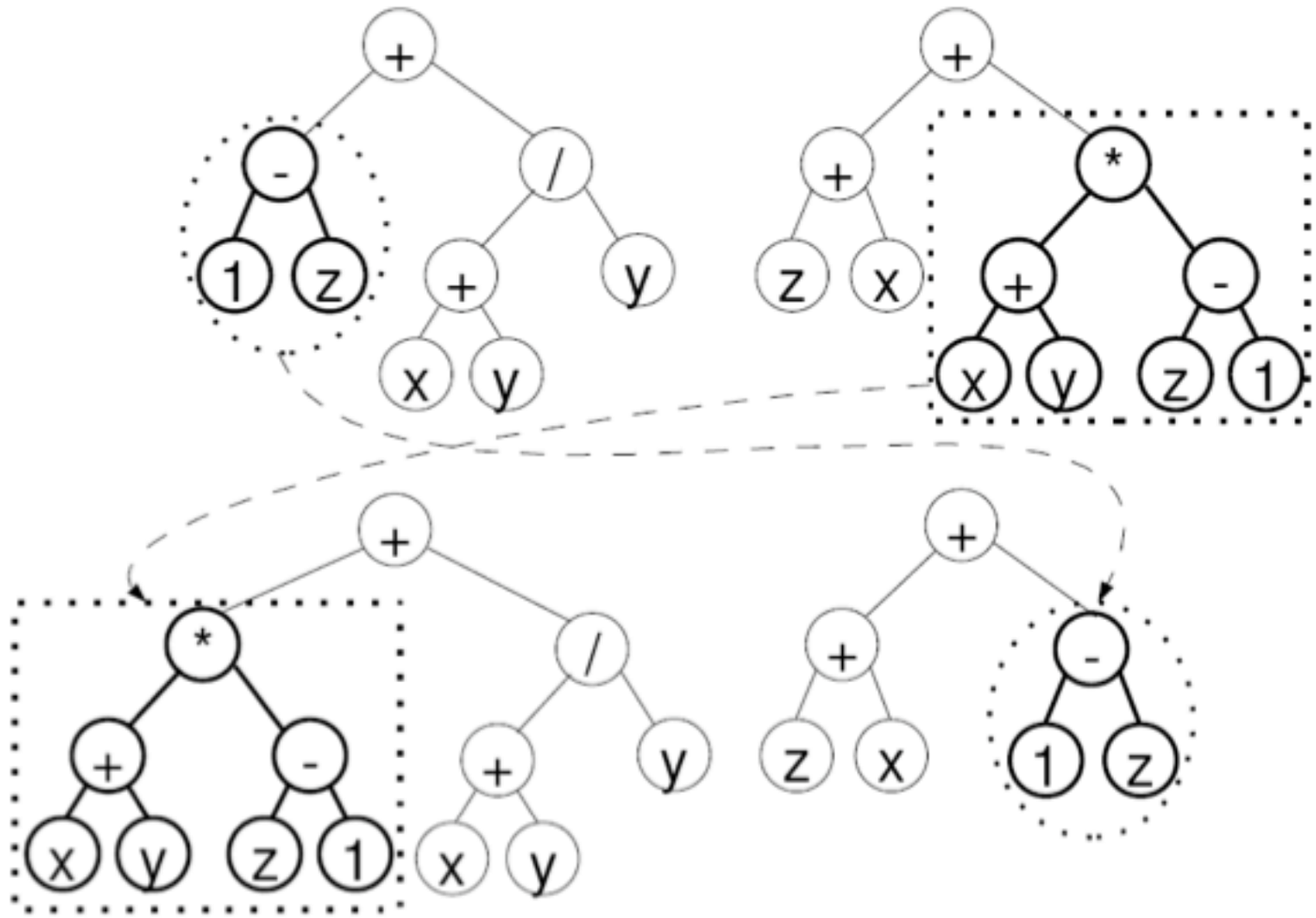


$$f(d_1, d_2, d_3) = \frac{d_1 * d_2}{d_1} + \sqrt{d_3}$$

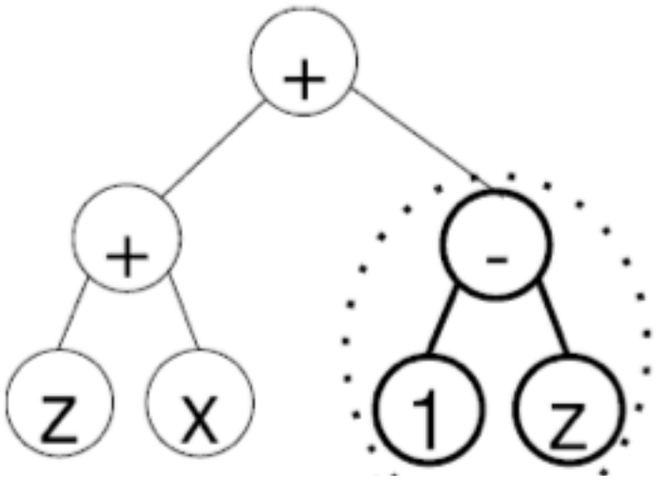
Reprodução



Recombinação

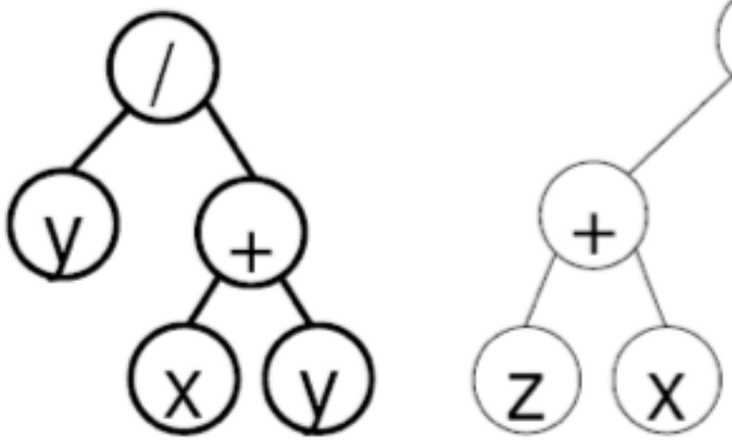


Mutação



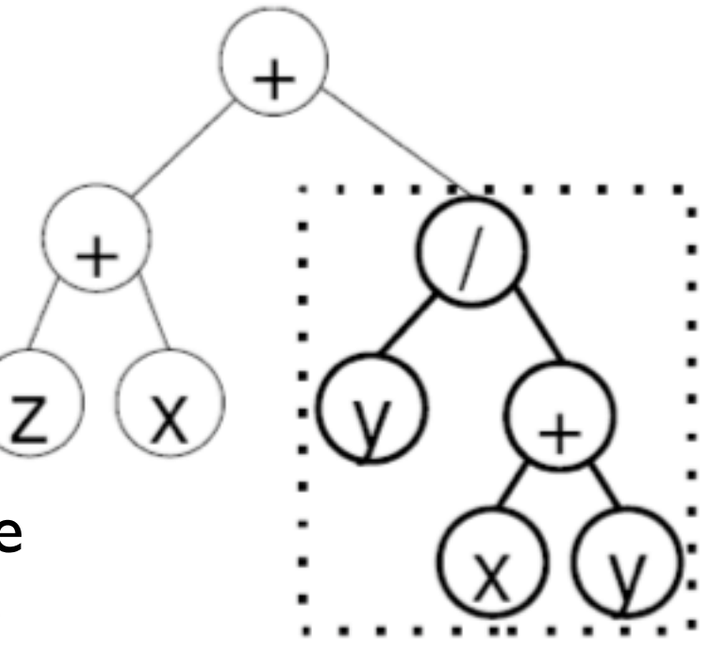
Sub-árvore selecionada

(a)



Gera sub-árvore aleatoriamente

(b)



(c)

Vamos pensar

- Como seria se usássemos algoritmos genéticos?
- GP melhor que GA?
- E no caso de recuperação de imagens?

Curiosidade

	Ex.	1	2	3	4	5	6	7	8	9
PG										
AG										
BAS										

Lição da aula de hoje

Lição de casa

1. O que aprendemos na aula de hoje?
2. Computação Evolutiva para resolver problemas difíceis
3. Baseada em leis da natureza tais como seleção natural, mutação, reprodução
4. AE não devem ser considerados prontos para o uso mas sim como um elenco de procedimentos gerais que podem ser prontamente adaptados a cada aplicação

Para casa...

Para casa...

- Considere o problema
  - Suponha que um caixeiro deva partir de sua cidade, visitar clientes em outras 99 cidades diferentes, e então retornar à sua cidade.
  - Dadas as coordenadas das 100 cidades, descubra o percurso de menor distância que passe uma única vez por todas as cidades e retorne à cidade de origem.

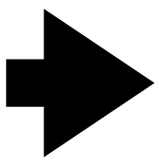
Para casa – pergunta-se

- Podemos utilizar AG para resolver esse problema? Por quê?
- Quantas combinações teríamos se tentássemos resolver por força bruta?
- Imagine uma codificação possível para o problema
- Como gerar uma solução inicial?
- Proponha uma função de adequação

Referências

Evolutionary Computation 1: Basic Algorithms and Operators. Eds. Thomas Back, David Fogel and Zbigniew Michalewicz. Capítulos 1-3, 8-10.

Evolutionary Computation 2: Advanced Algorithms and Operators. Eds. Thomas Back, David Fogel and Zbigniew Michalewicz.



Computação Evolutiva: Uma abordagem Pragmática. Fernando J. Von Zuben.

Inteligência Artificial: Ferramentas e Teorias. Guilherme Bittencourt,

Obrigado!

CE – Motivação

- Podemos validar teorias e conceitos associados à biologia da evolução
- Podemos lidar com problemas com os quais não é possível ou é muito custoso obter uma descrição detalhada.
  - Ex.: algoritmos de programação linear
  - Requerem que a função objetivo seja linear
  - Caso contrário, busca baseada no *gradiente*

CE – Motivação

- Mas técnicas de busca baseadas em *gradiente* exigem o que?
  - Função objetivo diferenciável
  - Baixo custo de diferenciação
  - O que fazer caso a diferenciação não seja possível ou muito cara?