U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Artificial Intelligence/ Inteligência Artificial

Lecture 5: Optimization and Genetic Algorithms

Luís Paulo Reis

lpreis@fe.up.pt

Director of LIACC – Artificial Intelligence and Computer Science Lab.
Associate Professor at DEI/FEUP – Informatics Engineering Department,
Faculty of Engineering of the University of Porto, Portugal
President of APPIA – Portuguese Association for Artificial Intelligence

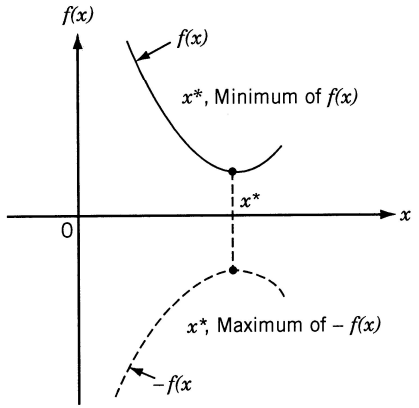
Problemas de Otimização

- **Muitos problemas do mundo real envolvem maximizar ou minimizar um valor:**
  - Como é que um fabricante de carros pode obter o máximo de peças de um pedaço de chapa?
  - Como é que uma empresa de mudanças pode encaixar/transportar o máximo maioria dos móveis num camião com um dado tamanho?
  - Como é que uma companhia telefónica pode rotear chamadas para obter o melhor uso das suas linhas e conexões?
  - Como uma universidade pode escalonar as suas aulas para fazer o melhor uso das salas de aula sem conflitos?

Problemas de Otimização

- **Problema de Otimização Matemática (minimização):**

$$\text{minimize } f_0(x)$$
$$\text{subject to } g_i(x) \leq b_i, \quad i = 1,\ldots,m$$



- $f_0 : R^n \rightarrow R$:
  - Função objetivo (calcula um valor)
- $x = (x_1,\ldots,x_n)$:
  - Variáveis controláveis (linearmente independentes)
- $g_i : R^n \rightarrow R$: $(I = 1,\ldots,m)$:
  - Restrições (podem ser de outros tipos)

Algoritmos de Melhoria Iterativa

- **Em muitos problemas de otimização, o caminho para o objetivo é irrelevante! O objetivo é ele mesmo a solução!**
- **Espaço de Estados = conjunto das configurações completas!**
- **Algoritmos Iterativos mantém um único estado (corrente) ou uma população de estados e tentam melhora-lo(s)!**
- **Algoritmos de Melhoria Iterativa:**
  - Pesquisa Subida da Colina (Hill-Climbing Search)
  - Arrefecimento Simulado (Simulated Annealing)
  - Pesquisa Tabu (Tabu Search)

  População de Soluções:
  - Algoritmos Genéticos (Genetic Algorithms)
  - Particle Swarm Optimization
  - Ant Colony Optimization
- **Estratégia: Começar como uma solução inicial/população de soluções iniciais do problema e fazer alterações de forma a melhorar a sua qualidade**

Algoritmos de Melhoria Iterativa

- **"Individual Based" (só uma solução)!**

- **Hill-Climbing Search**
  - Escolher um estado aleatoriamente do espaço de estados
  - Considerar todos os vizinhos desse estado
  - Escolher um vizinho ou o melhor vizinho
  - Se o vizinho escolhido é melhor que a solução atual ir para esse vizinho
  - Repetir o processo até não existirem vizinhos melhores
  - O estado corrente é a solução

- **Simulated Annealing**
  - Semelhante ao Hill-Climbing Search mas admite explorar vizinhos piores
  - Temperatura que vai sendo sucessivamente reduzida define a probabilidade de aceitar soluções piores

- **Tabu Search**
  - Semelhante ao Hill-Climbing Search, explora os estados vizinhos mas elimina os piores (vizinhos tabu)
  - Algoritmo determinístico

Algoritmos de Melhoria Iterativa

- **“Population Based” - População de Soluções**

- **Particle Swarm Optimization/Enxame de Particulas**
  - Vários estados de partida (enxame)
  - Explora-se a vizinhança e guarda-se, a melhor solução e o melhor estado
  - Movimentam-se os estados na direção da melhor solução encontrada até ao momento
  - A velocidade de movimentação depende das distâncias à melhor solução e melhor estado e da posição do estado

- **Ant Colony Optimization/Colónia de Formigas**
  - Vários estados (colónia formigas) de partida
  - Determina-se a probabilidade de um caminho ser melhor a partir do número de “formigas” que passa por ele

- **Algoritmos Genéticos/Genetic Algorithms**
  - Definição do estado como um cromossoma
  - Gerar soluções (cromossomas)a partir de uma população de estados inicial
  - Reprodução, Mutação e Seleção

Hill-Climbing (Subir a Colina)

Tipos de **"hill climbing"** (subir-a-colina):

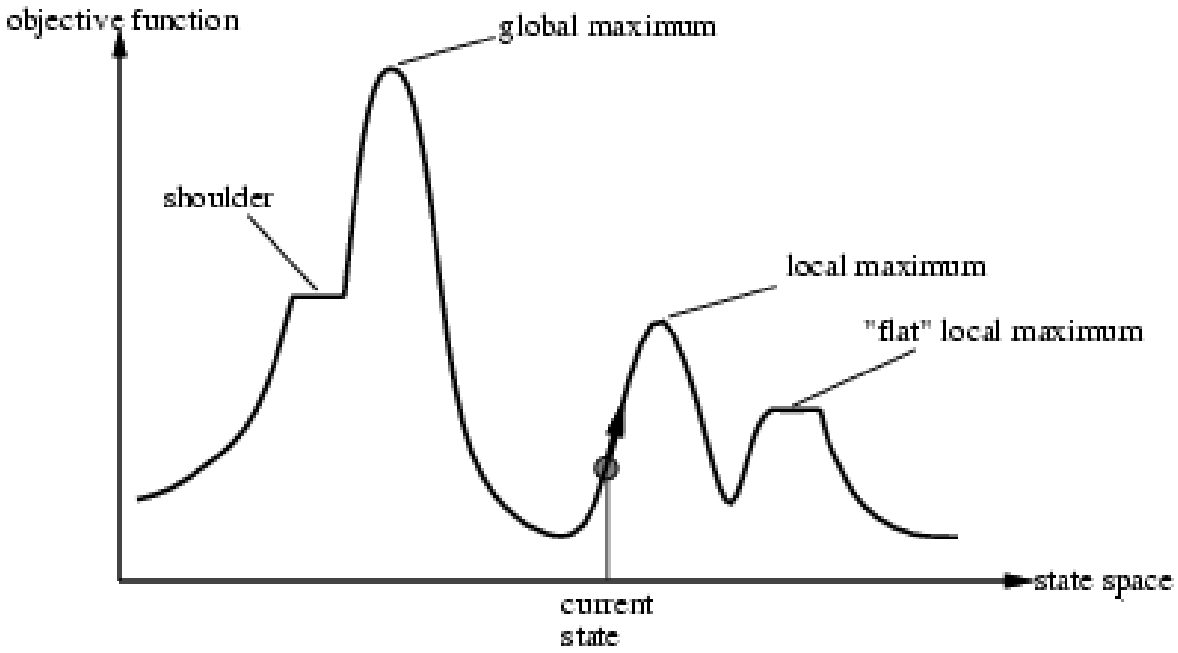
- 1) "Hill climbing" básico: Gera **um a um** os sucessores do estado atual. Encontrado um mais próximo da solução que o Estado Atual, selecciona-o e aplica-o

- 2) "Steepest ascent": Gera **todos** os sucessores possíveis ("ascenção íngreme") selecciona o mais próximo da solução

- 3) Aleatório: Gera **um** sucessores aleatoriamente de entre os possíveis e seleciona-o se for melhor que a solução atual

Pesquisa Subida da Colina (Hill-Climbing Search)

- **Problema: Dependendo do estado inicial pode ficar "preso" num mínimo local!**



Pesquisa Subida da Colina (Hill-Climbing Search)

- **Algoritmo Base do Hill-Climbing**

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
```

Arrefecimento Simulado ("Simulated Annealing")

- Estratégia: Escapar do mínimo local permitindo alguns "maus" movimentos mas gradualmente diminuindo a sua dimensão e frequência!
- Algoritmo Base do Arrefecimento Simulado:

```
current ← MAKE-NODE(INITIAL-STATE[problem])
for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← VALUE[next] – VALUE[current]
    if ΔE > 0 then current ← next
    else current ← next only with probability e^(Δ E/T)
```

Pesquisa Tabu/Tabu Search

- **Critério de Aceitação/Rejeição**
  - Simulated Annealing: Probabilistico
  - Tabu Search: Deterministico
- **Lista Tabu**
  - Lista de Movimentos Proibidos: Contendo um número fixo de soluções recentemente visitadas (FIFO)
  - Por vezes são guardados os operadores para gerar as soluções e não as próprias soluções completes na lista tabu
  - Evitar Ciclios: smaller than the size of the tabu-list
- **Algoritmo Base da Pesquisa Tabu:**

```
s ← generateInitialSolution()
best_s ← s
tabu_list ← new Queue(size)
repeat
        s' ← generateNonTabuNeighbor(s, tabu_list)
        tabu_list.push(s)
        best_s ← best(best_s, s')
        s ← s'
until stopping criterion
return best_s
```

Tabu Search – Exemplo

- $1||\sum w_j T_j$
  - Vizinho: Adjacent pairwise interchanges
  - Lista Tabu: Pares de Jobs trocados; tabu-list size = 2
  - First schedule $\boldsymbol{S_1} = 2,1,4,3 \rightarrow \sum w_j T_j = 500$

$jobs$	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
w_j	14	12	1	12

Neighbors of $\boldsymbol{S_1}$: 1,2,4,3 (480); 2,4,1,3 (436); 2,1,3,4 (652)
Best (non-tabu) neighbor of S_1 is $\boldsymbol{S_2} = \boldsymbol{2,4,1,3}$
Tabu-list: $\langle(1,4)\rangle$

Neighbors of $\boldsymbol{S_2}$: 4,2,1,3 (460); 2,1,4,3 (500); 2,4,3,1 (608)
Best (non-tabu) neighbor of S_2 is $\boldsymbol{S_3} = \boldsymbol{4,2,1,3}$
Tabu-list: $\langle(2,4),(1,4)\rangle$

Neighbors of $\boldsymbol{S_3}$: 2,4,1,3 (436); 4,1,2,3 (440); 4,2,3,1 (632)
Best (non-tabu) neighbor of S_3 is $\boldsymbol{S_4} = \boldsymbol{4,1,2,3}$
Tabu-list: $\langle(1,2),(2,4)\rangle$

Neighbors of $\boldsymbol{S_4}$: 1,4,2,3 (408); 4,2,1,3 (460); 4,1,3,2 (586)
Best (non-tabu) neighbor of S_4 is $\boldsymbol{S_5} = \boldsymbol{1,4,2,3}$
Tabu-list: $\langle(1,4),(1,2)\rangle$

...

tabu!

best so far

Algoritmos Genéticos

“O que é bom para a Natureza é bom para os Sistemas Artificiais”

“The Origin of Species on the basis of Natural Selection”

- Charles Darwin: Marco histórico do Conhecimento

• Organismos adaptados ao meio reproduzem-se

• Sucessores tem um grau de similitude com os antepassados

• Mutações aleatórias podem ter ou não sucesso

• Evolução e Mutação permitem adaptação em ambientes (moderadamente) dinâmicos

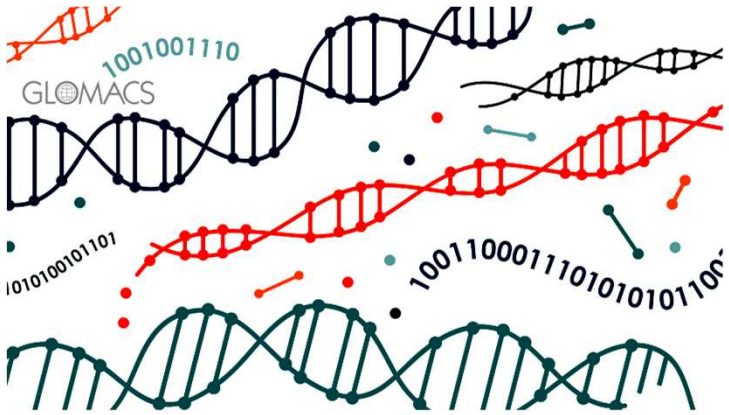
Algoritmos Genéticos

- A pesquisa em alguns espaços de pesquisa utilizando métodos de pesquisa tradicionais seria intratável
- Isto geralmente ocorre quando os estados/ soluções candidatas têm um número de sucessores muito grande
- Algoritmos genéticos são uma estratégia de pesquisa heurística aleatória
- Ideia básica: Simular a seleção natural, onde a população é composta por soluções candidatas
- O foco está em evoluir uma população da qual candidatos fortes e diversificados podem emergir através de mutação e crossover (reprodução).

Algoritmos Genéticos

- **Algoritmo Genético (ou AG) é uma variante da pesquisa de feixe estocástico na qual estados sucessores são gerados combinando dois estados pai em vez de modificar um único estado**

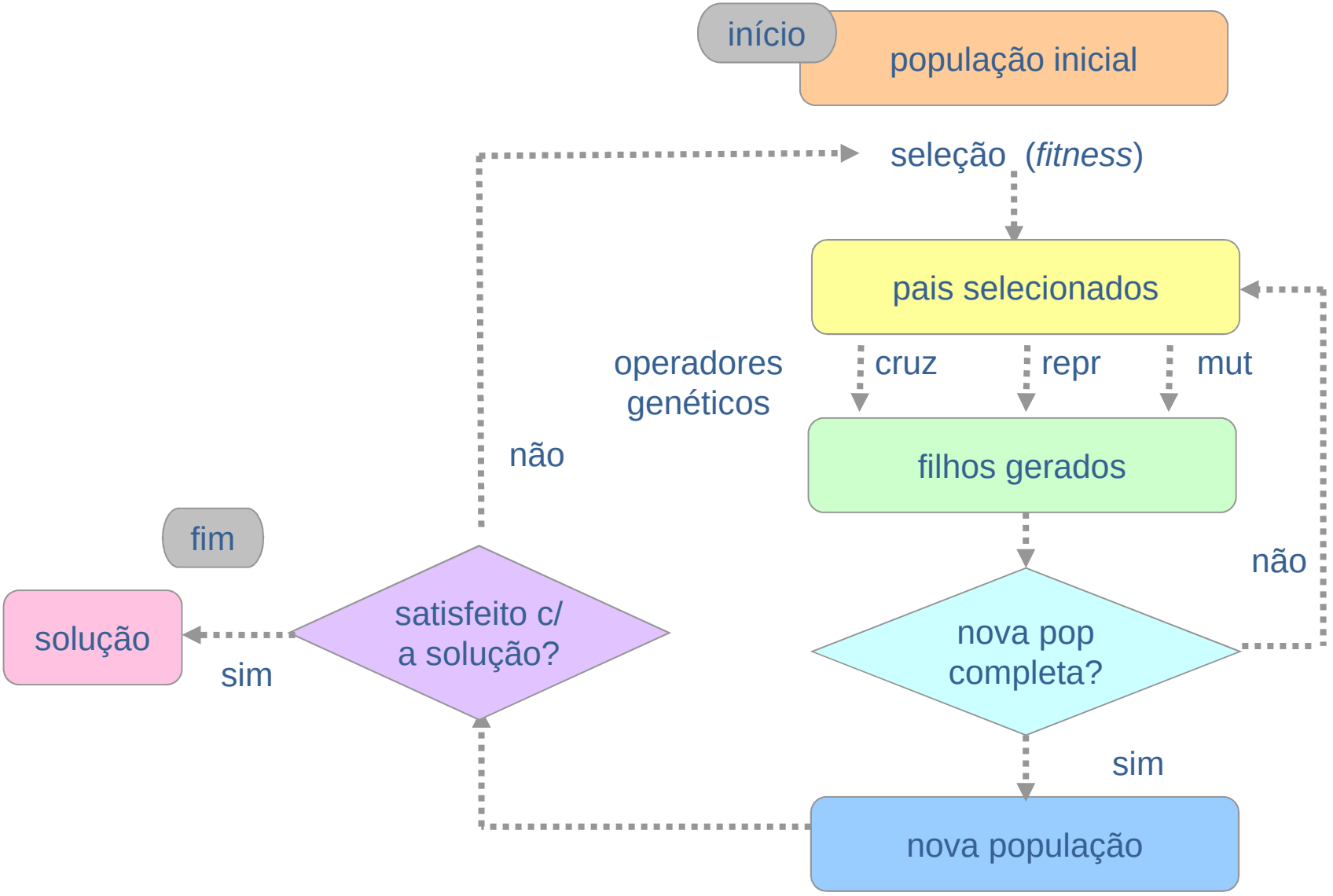
- **A analogia com a seleção natural é a mesma que na pesquisa de feixe estocástico, exceto que agora estamos a lidar com reprodução sexual em vez de reprodução assexuada**



Algoritmo Básico

- **Crie uma população inicial, aleatória ou "em branco"**
- **Enquanto o melhor candidato, encontrado até agora não é uma solução:**
  - Crie nova população usando soluções sucessoras (Reprodução)
  - Avalie a adequação de cada candidato da população
  - Devolva o melhor candidato encontrado

Algoritmos Genéticos



Componentes Básicos

- **Representação das soluções:**
  - Importante escolher bem esta representação
  - Mais trabalho aqui significa menos trabalho nas funções sucessoras
- **Função(ões) sucessora(s):**
  - Mutação
  - Crossover
- **Função de Fitness**
- **Teste de Solução**
- **Alguns parâmetros:**
  - Tamanho da População
  - Limite de Gerações
  - Método de Seleção
  - Método de Cruzamento
  - Método de Mutação
  - Elitismo

Representação de Soluções

- **Queremos codificar os candidatos de uma maneira que facilite a mutação e o crossover**
- **A representação típica de candidato é uma cadeia binária**
- **Esta cadeia pode ser considerada como o código genético de um candidato - daí o termo "algoritmo genético"!**
- **Outras representações são possíveis, mas tipicamente tornam o cruzamento e a mutação mais difíceis**

Funções Sucessoras

- **Mutação - Dado um candidato, devolver um candidato ligeiramente diferente**
- **Crossover (Cruzamento) - Dados dois candidatos, produz umnovo candidato que tenha elementos de cada um**
- **Nem sempre geramos um sucessor para cada candidato.**
- **Pelo contrário, geramos uma população sucessora baseada nos candidatos da população atual, ponderada pela sua aptidão (Fitness)**

Função Sucessora

- **Se a representação candidata é apenas uma string binária, então é fácil:**
  - Mutate(c):
    - Copy c as c'
    - For each bit b in c', flip b with probability p
    - Return c'
  - Cross (c1, c2):
    - Create a candidate c such that c[i] = c1[i]
    - if (i % 2) = 0, c[i] = c2[i]
    - Return c.
    - Alternatively, any other scheme such that c gets roughly equal information from c1 and c2.

Função de Fitness

- **A função de fitness (adequação) é análoga a uma heurística que estima quão próximo um candidato está de ser uma solução**
- **Em geral, a função de adequação deve ser consistente para um melhor desempenho.**
- **No entanto, mesmo que seja, não há garantias dado que este é um algoritmo probabilístico!**

Teste da Solução

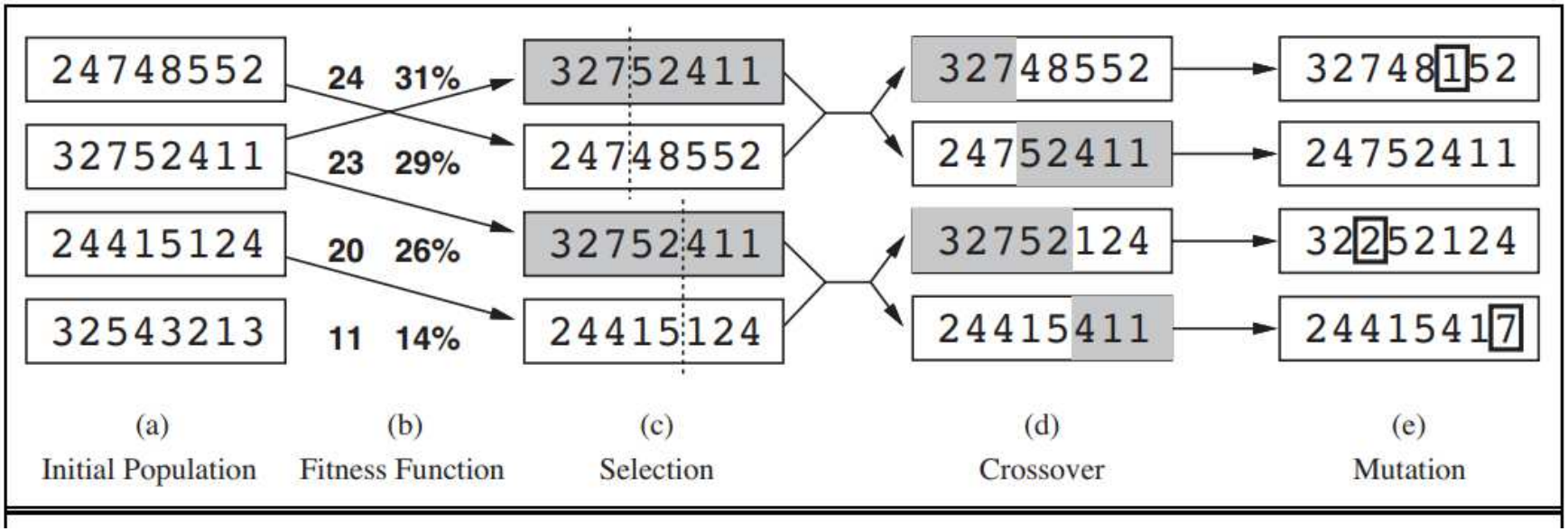
- **Dado um candidato, devolva se o candidato é uma solução**
- **Muitas vezes, apenas responde à pergunta:**
  - "o candidato satisfaz algum conjunto de restrições?"
- **Opcional! Às vezes só queremos encontrar o melhor possível num determinado número de gerações**

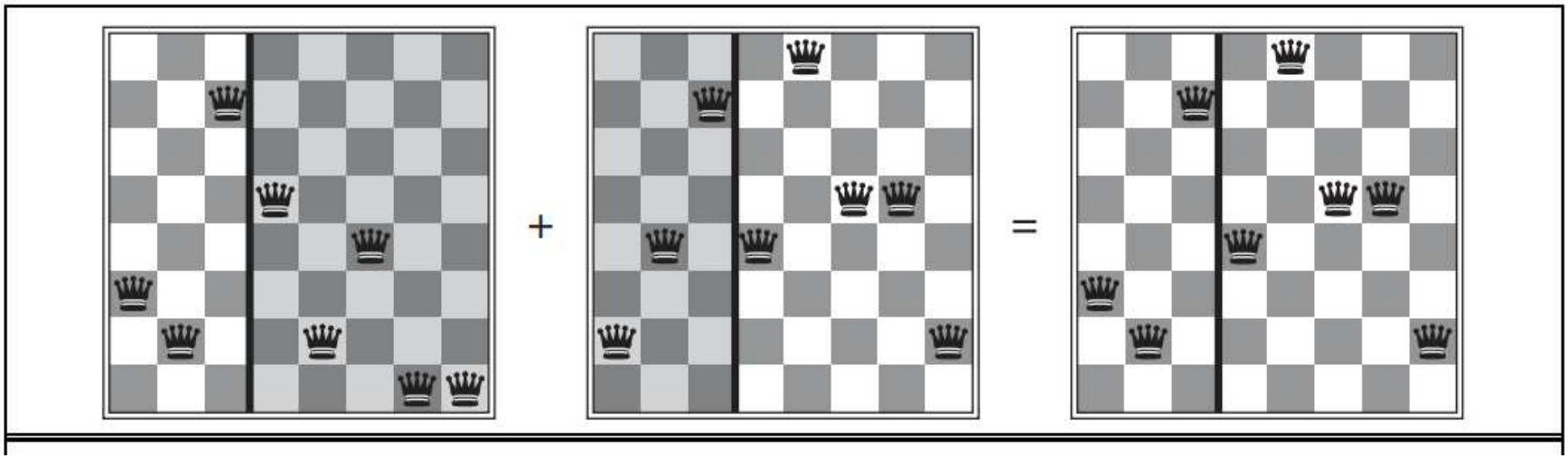
Geração da Nova População

- **Como chegamos a uma nova população?**
  - Dada uma população P, gere P ’realizando cruzamento (crossover) | P | vezes, cada vez selecionando candidatos com probabilidade proporcional à sua aptidão (fitness).
  - Obtenha P’’, alterando cada candidato em P’
  - Devolva P ’’

- **Esta é uma abordagem possível**
- **Mas é possível ser criativo!**
- **Esta abordagem não permite explicitamente que os candidatos sobrevivam a mais de uma geração - isso pode não ser o ideal**
- **Elitismo!**

Algoritmos Genéticos





Algoritmos Genéticos

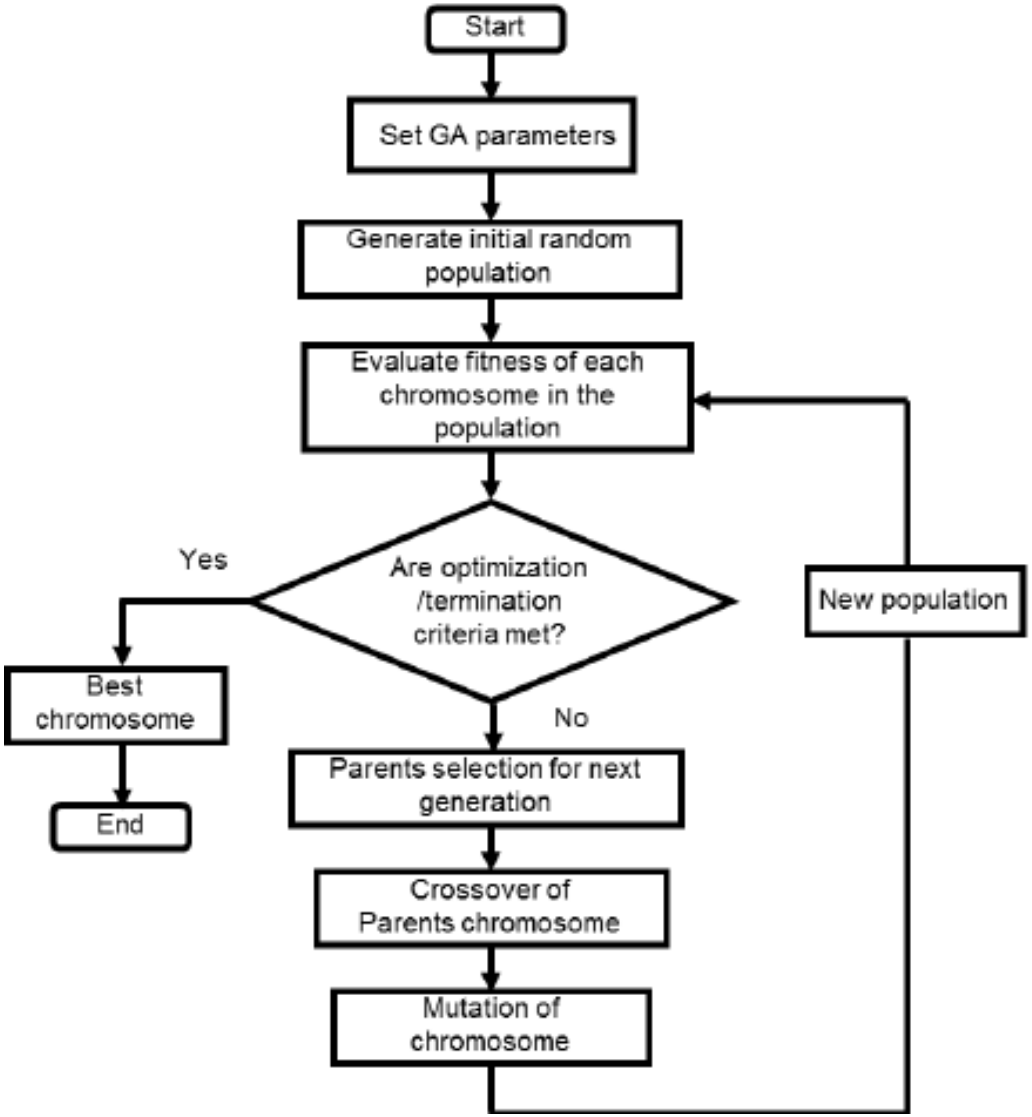
```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new_population ← empty set
    for i = 1 to SIZE(population) do
      x ← RANDOM-SELECTION(population, FITNESS-FN)
      y ← RANDOM-SELECTION(population, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) then child ← MUTATE(child)
      add child to new_population
    population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Fluxograma GAs



Terminologia Biológica

Na área dos algoritmos genéticos são utilizados termos biológicos como analogia com a biologia:

- **Cromossoma -** Codificação de uma possível solução – indivíduo
- **Gene -** Codifica uma característica particular
- **Genótipo -** Informações do DNA do indivíduo – características de base
- **Fenótipo -** Características visíveis do indivíduo depois da interação com o meio ambiente

Codificação de Indivíduos

- Material genético
- Conjunto de atributos da solução
- Cada atributo é codificado como uma sequência de bits
- Individuo codificado como a concatenação das sequências de bits
- Codificação binária, códigos

População

Conjunto de indivíduos que estão sendo analisados como possíveis soluções:

- Relacionado a dimensão do espaço de pesquisa
- Populações pequenas têm grandes chances de perder a diversidade necessária (exploração de todo o espaço de soluções)
- Populações grandes perderão grande parte de sua eficiência pela demora em avaliar a função de fitness

Reprodução

Reprodução sexual, genes são trocados entre dois pais

crossover

Os filhos são sujeitos a modificações, nas quais os bits elementares são mudados

mutação

Função de Fitness

- Mede a adaptação do indivíduo – qualidade da solução dada por este indivíduo
- Representativa do problema: diferencia uma solução boa de uma má
- Heurística de busca no espaço de estado
- Cuidados com o custo computacional

Seleção

O operador escolhe quais indivíduos participarão na criação da próxima geração

Torneio* | *Roleta* | *...

Requisitos para usar AG

- Representações das possíveis soluções do problema no formato de um código genético
- População inicial que contenha diversidade suficiente para permitir ao algoritmo combinar características e produzir novas soluções
- Existência de um método para medir a qualidade de uma solução potencial
- Um procedimento de combinação de soluções para gerar novos indivíduos na população
- Um critério de escolha das soluções que permanecerão na população ou que serão retirados desta
- Procedimento para introduzir periodicamente alterações em algumas soluções da população.

Manter a diversidade da população aumenta a possibilidade de se produzir soluções inovadoras!

Métodos de Seleção

Torneio

Selecionar conjuntos de indivíduos (com cardinalidade a definir) e utilizar o melhor deles

A cardinalidade tem impacto na diversidade dos pais selecionados

Roleta

Seleccionar conjuntos de indivíduos de acordo com uma função de distribuição.

A função de distribuição é baseada na função de fitness

Métodos de Seleção

Selecionar a melhor geração da população atual para progenitores

S	F(S)
1	14
2	17
3	17
4	11
5	8
6	23
7	22
8	20

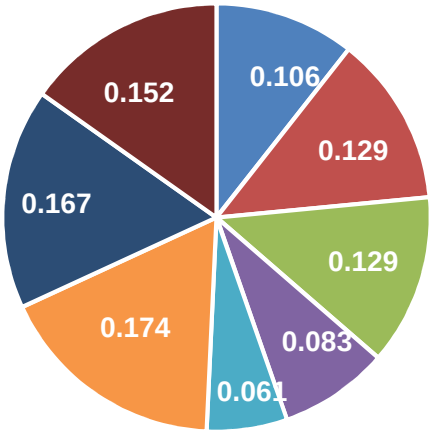
np=3

S2

S8

Método do torneio

- Definir número de participantes (np)
- Seleção aleatória dos participantes



S	Prb(S)
1	0,106
2	0,129
3	0,129
4	0,083
5	0,061
6	0,174
7	0,167
8	0,152

Método da roleta

- Atribuir valor relativo da solução
- Seleção dos participantes de acordo com a distribuição de probabilidades

Reprodução e Métodos de Combinação

Reprodução

Preserva características úteis

Introduz variedade e novidades

Estratégias:

- Pais únicos: clonar + mutação
- Pais múltiplos: recombinação + mutação

Métodos de combinação

Cruzamento: cria novos indivíduos misturando características de dois indivíduos pais (crossover)

Copia de segmentos entre os pais

Crossovers multi-ponto, dois pontos, um ponto, uniforme e inversão

Cruzamento e Mutação

Cruzamento

Progenitor 1: 1010101011 0101010111

Progenitor 2: 0000100101 0101110010

Cruzamento num ponto:

1010101011 0101110010

0000100101 0101010111

Cruzamento uniforme: os filhos são formados a partir dos bits dos pais (sorteado)

Mutação

Esta operação inverte aleatoriamente alguma característica do indivíduo

Cria novas características que não existiam

Mantém diversidade na população

Terminação

- **Processo geracional é repetido até que uma condição de finalização seja atingida.**
- **Condições comuns de terminação são:**
  - Solução ótima foi alcançada
  - Uma solução é encontrada que satisfaz critérios mínimos
  - Número fixo de gerações atingidas
  - Orçamento alocado (tempo de computação / dinheiro) atingido
  - A adequação da solução de classificação mais elevada está a alcançar ou já atingiu um patamar tal que iterações sucessivas não produzem mais resultados melhores
  - Inspeção manual
  - Combinações das ideias acima

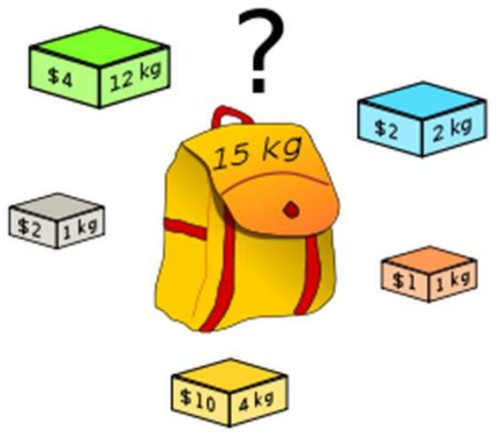
Utilização de AGs

- Funções multimodais
- Funções discretas ou continuas
- Funções altamente dimensionais, incluindo combinatórias
- Dependência não linear dos parâmetros
- Usados frequentemente para obter solução em problemas NP
- Não usar Ags quando outro método como hill-climbing, etc., funciona bem ou pelo menos antes de experimentar esse tipo de método!

AG's são apropriados para problemas complexos, mas algumas melhorias devem ser realizadas!

Exemplo – Knapsack

- O problema da mochila (Knapsack problem) é um problema de otimização combinatória
- Pretende-se preencher uma mochila com objetos de diferentes pesos e valores
- O objetivo é que se preencha a mochila com o maior valor possível, não ultrapassando o peso máximo
- O problema da mochila é um dos 21 problemas NP-completos de Richard Karp, expostos em 1972
- A formulação do problema é extremamente simples, porém sua solução é complexa



Exemplo - Knapsack

Exemplo:

N = 8

Capacidade (C) = 50

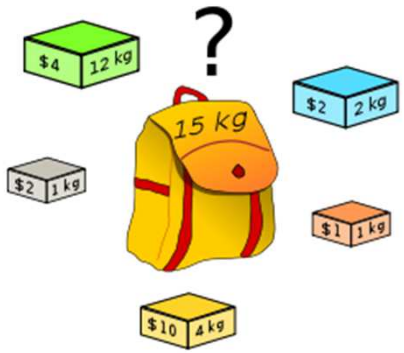
	1	2	3	4	5	6	7	8
Valor	4	3	6	7	2	9	7	6
Peso	12	16	8	21	16	11	6	12

Objectivo

Maximizar $\sum_i O_i \times v_i$

Restrição $\sum_i O_i \times p_i \leq 50$

?



Representação das soluções: Oi

1	1	0	0	0	0	1	0

Soluções Inválidas

Soluções Inválidas: Como resolver?

- Reparar
- Alterar a representação
- Penalizar

Uma função de penalização define em que quantidade a solução X viola a restrição R

Mede a distância a que a solução está de uma região de aceitação

Transforma um problema com restrições num problema sem restrições

Exemplo - Knapsack

	1	2	3	4	5	6	7	8
Valor	4	3	6	7	2	9	7	6
Peso	12	16	8	21	16	11	6	12

Avaliação

$$f(s) = \begin{cases} \sum_i O_i \times v_i & se\ solução\ válida \\ 0 & se\ solução\ não\ válida \end{cases}$$

$$f(s) = \sum_i O_i \times v_i - p(s)$$

Onde $$p(s) = \begin{cases} \alpha\left(\sum_i O_i \times v_i\right) - C & se\ solução\ não\ válida \\ 0 & se\ solução\ válida \end{cases}$$

A penalização de uma solução válida é zero e é proporcional ao grau de violação das restrições

Exemplo - Knapsack

	1	2	3	4	5	6	7	8
Valor	4	3	6	7	2	9	7	6
Peso	12	16	8	21	16	11	6	12

Criar e avaliar a população inicial

Solução								Valor	Peso	p(s)	f(s)
1	1	0	0	0	0	1	0	14	34	0	14
0	0	0	1	1	0	1	1	22	55	5	17
0	1	0	1	0	0	1	0	17	43	0	17
1	0	0	0	0	0	1	0	11	18	0	11
0	1	1	1	1	0	1	0	25	67	17	8
1	0	0	1	0	0	1	1	24	51	1	23
0	1	0	1	0	1	1	0	26	54	4	22
1	1	0	0	0	0	1	1	20	46	0	20

Exemplo - Recombinação

Recombinar soluções

Combinar o material genético de dois progenitores para gerar novas soluções

Objetivo: Combinar características interessantes de duas soluções

Recombinação por ponto de corte

- Alinhar os dois progenitores
- Selecionar um ponto de corte aleatório
- Combinar secções complementares para obter descendentes

Exemplo - Recombinação

População inicial

Solução (progenitores)								Valor	Peso	p(s)	f(s)
1	1	0	0	0	0	1	0	14	34	0	14
0	0	0	1	1	0	1	1	22	55	5	17
0	1	0	1	0	0	1	0	17	43	0	17
1	0	0	0	0	0	1	0	11	18	0	11
0	1	1	1	1	0	1	0	25	67	17	8
1	0	0	1	0	0	1	1	24	51	1	23
0	1	0	1	0	1	1	0	26	54	4	22
1	1	0	0	0	0	1	1	20	46	0	20

Recombinação

Solução (filhos)								Valor	Peso	p(s)	f(s)
0	0	0	1	1	0	1	0	16	43	0	16
1	0	0	0	0	0	1	1	17	30	0	17
0	1	0	1	0	0	1	1	23	55	5	18
1	1	0	0	0	1	1	0	23	45	0	23

Exemplo - Mutação

Mutação

Alteração do valor de um gene (mutação binária)

Cria variabilidade no conjunto das soluções

Introduz alterações no material genético

Objetivo: Introduzir diversidade na população

População filhos

Solução (filhos)								Valor	Peso	p(s)	f(s)
0	0	0	1	1	0	1	0	16	43	0	16
1	0	0	0	0	0	1	1	17	30	0	17
0	1	0	1	0	0	1	1	23	55	5	18
1	1	0	0	0	1	1	0	23	45	0	23

Mutação

Solução (filhos)								Valor	Peso	p(s)	f(s)
0	0	0	1	1	0	1	0	16	43	0	16
1	0	0	0	0	1	1	1	26	41	0	26
0	1	0	1	0	0	1	1	23	55	5	18
1	1	0	0	0	1	1	0	23	45	0	23

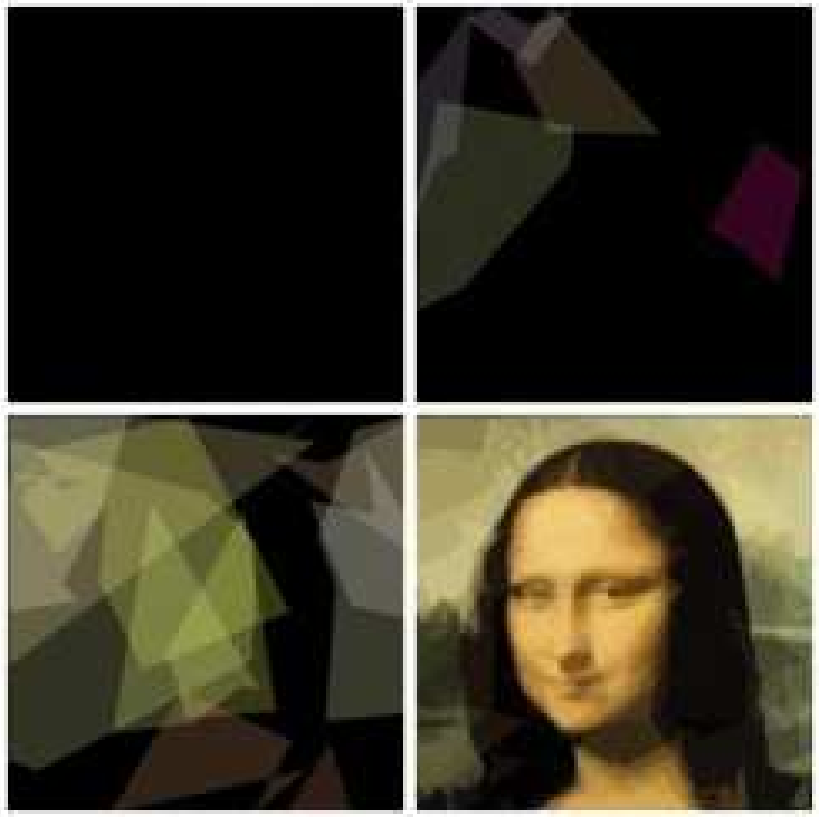
AGs - Prós e Contras

- **Prós**
  - Mais rápido (e menores requisitos de memória) do que pesquisa num espaço de pesquisa muito grande de modo sistemático
  - Fácil, pois se a função de representação e a adequação de candidatos estiver correta, uma solução pode ser encontrada sem qualquer trabalho analítico explícito
- **Contras**
  - Aleatório - não é ideal nem é um algoritmo completo
  - Pode ficar preso nos máximos/mínimos locais, embora o crossover e a mutação possam ajudar a atenuar isso
  - Pode ser difícil descobrir qual a melhor forma de representar um candidato como uma cadeia de bits (ou outra)
  - Mais pesado do que Hill-Climbing, Arrefecimento Simulado ou outro algoritmo ("individual based")

Exemplo – Evolving Mona Lisa

- **Image compression – evolving the Mona Lisa**

Generation 1



Generation 301

Generation 2716

Generation 904314

http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/

Exemplo – Evolving Mona Lisa

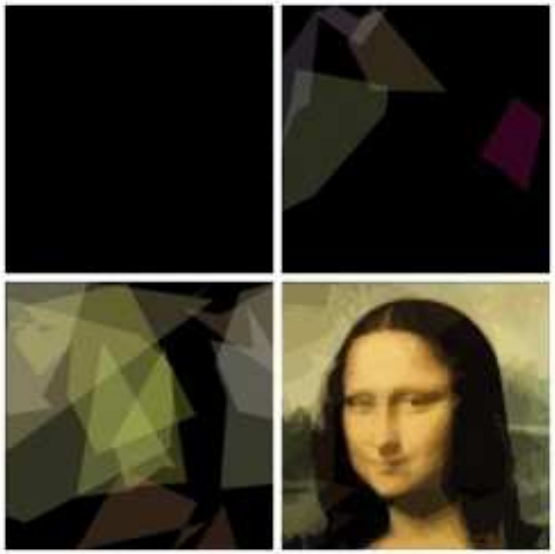
- **Program that keeps a string of DNA for polygon rendering.**

- **The procedure of the program is quite simple:**
  - 0) Setup a random DNA string (application start)
  - 1) Copy the current DNA sequence and mutate it slightly
  - 2) Use the new DNA to render polygons onto a canvas
  - 3) Compare the canvas to the source image
  - 4) If the new painting looks more like the source image than the previous painting did, then overwrite the current DNA with the new DNA
  - 5) Repeat from 1

- **Now to the interesting part :-)**

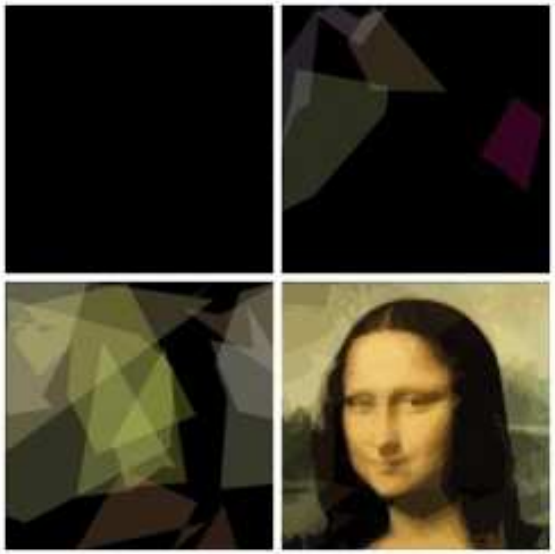
- **Could you paint a replica of the Mona Lisa using only 50 semi transparent polygons?**

- **Took 904314 generations, though began to be recognizable as the Mona Lisa after 10,000 or so.**
-
- **Images are generation 1, 301, 2716, and 904314.**



Exemplo – Evolving Mona Lisa

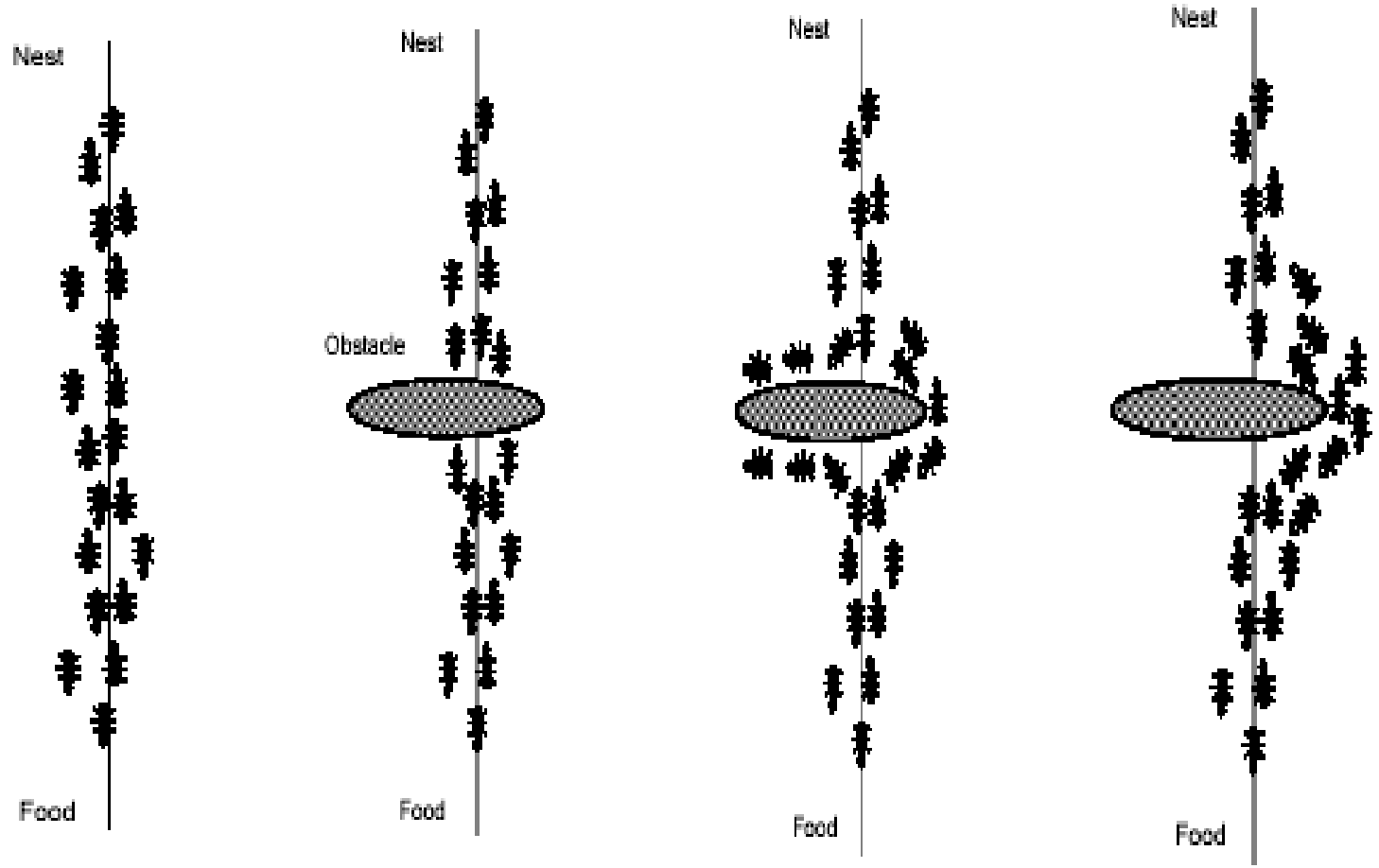
- Usa apenas 50 polígonos de 6 vértices cada
- Assumindo que cada polígono tem 4 bytes para cores (RGBA) e 2 bytes para cada um dos 6 vértices, essa imagem requer apenas 800 bytes.
- No entanto, o tempo de compressão é proibitivo e o armazenamento é mais barato que o tempo de processamento!



Ant Colony Optimization

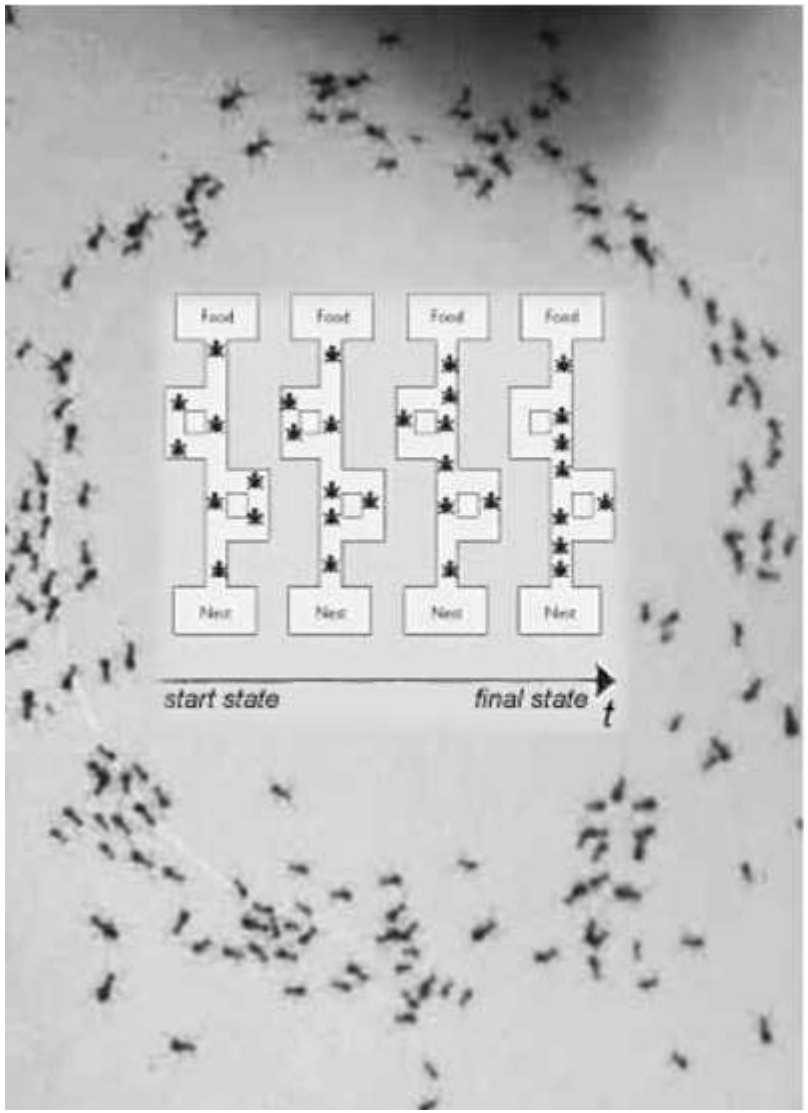
- O algoritmo da otimização da colônia de formigas (ACO, do inglês ant colony optimization algorithm), foi introduzido por Marco Dorigo na sua tese de doutoramento
- É uma heurística baseada em probabilidades, criada para solução de problemas computacionais que envolvem pesquisa de caminhos em grafos
- Este algoritmo foi inspirado na observação do comportamento das formigas ao saírem de sua colónia para encontrar comida

Comportamento das Formigas



Ant Algorithms – (P.Koumoutsakos – based on notes L. Gamberdella (www.idsia.ch)

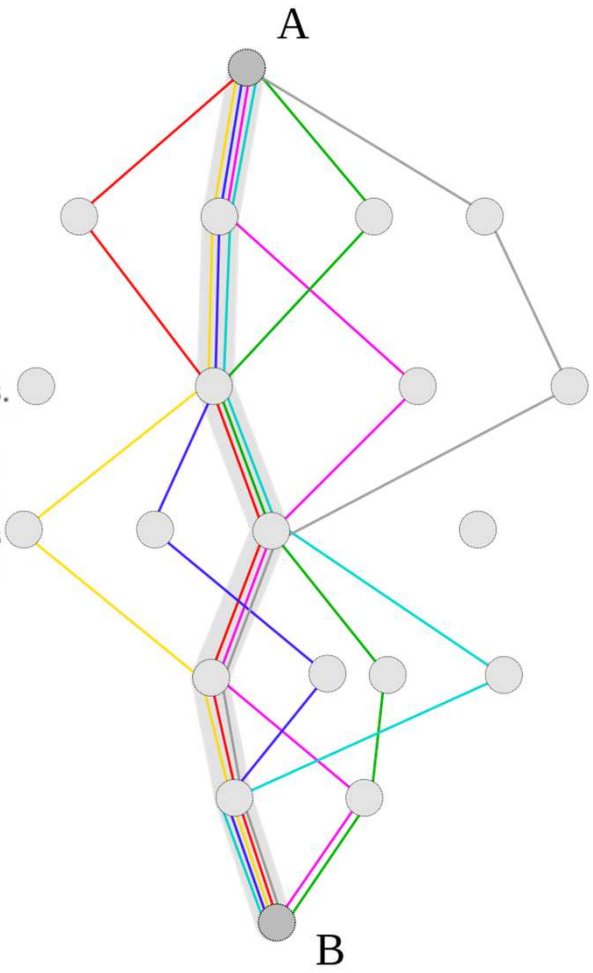
Ant Colony Optimization



The main quality of the colonies of insects, ants or bees lies in the fact that they are part of a self-organized group in which the keyword is simplicity.

Every day, ants solve complex problems due to a sum of simple interactions, which are carried out by individuals.

The ant is, for example, able to use the quickest way from the anthill to its food simply by following the way marked with pheromones.



Exemplo: Ant Colony para TSP

```
begin
    InitializationStep()
    while termination conditions not met do
        ConstructAntSolutions()
        ApplyLocalSearch()
        UpdatePheromones()
end
```

Loop

Randomly position m artificial ants on n cities

For city=1 to n

For ant=1 to m

{Each ant builds a solution by adding one city after the other}

Select probabilistically the next city according to exploration and exploitation mechanism

Apply the local trail updating rule

End for

End for

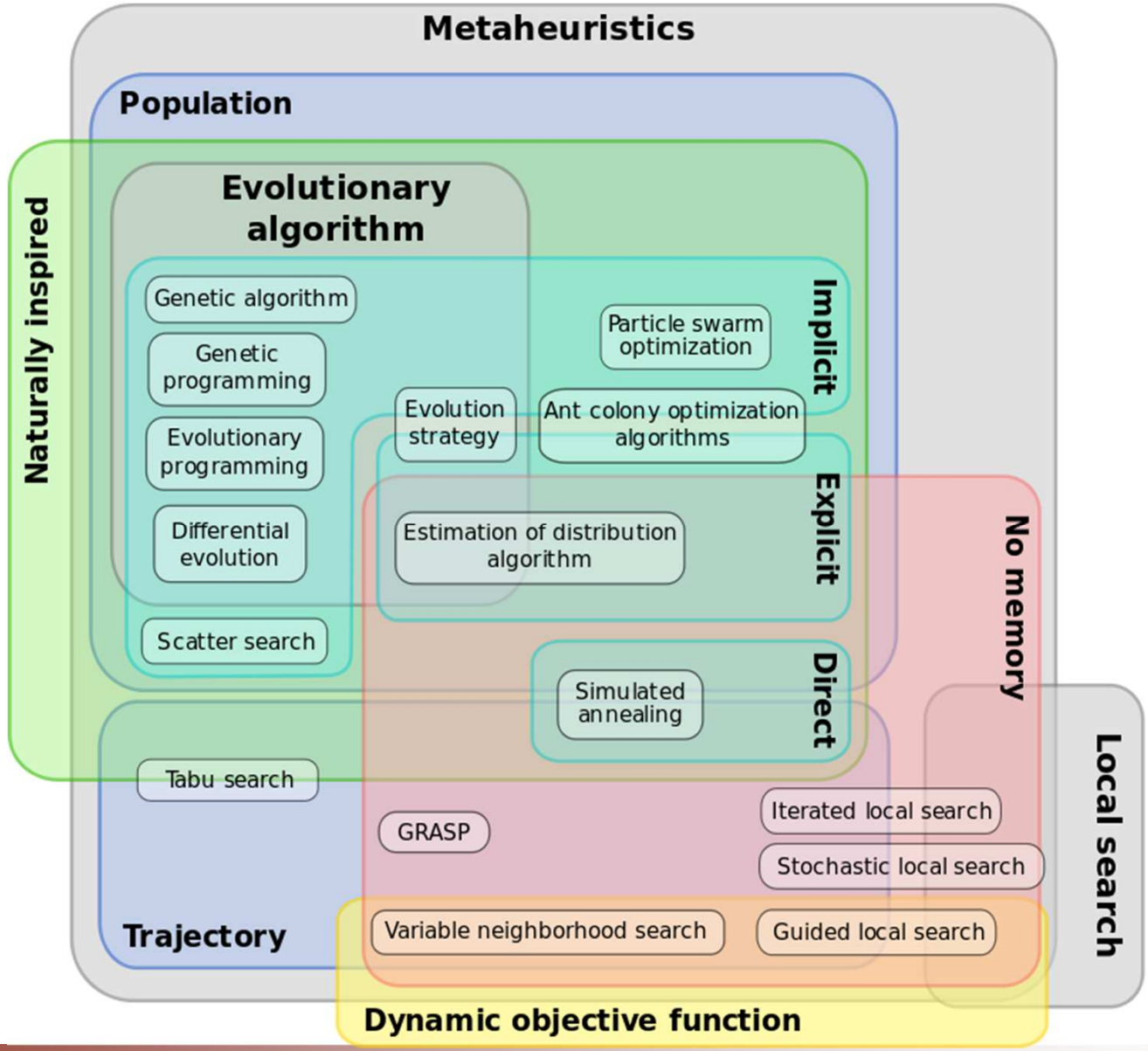
Apply the global trail updating rule using **the best ant**

Until End_condition

Metaheuristics

- In computer science and mathematical optimization, a **metaheuristic** is a **higher-level procedure or heuristic** designed **to find, generate, or select a heuristic** (partial search algorithm) that may provide a sufficiently **good solution to an optimization problem**, especially with incomplete or imperfect information or limited computation capacity
- Metaheuristics sample a **set of solutions** which is **too large** to be completely sampled.
- Metaheuristics make few assumptions about the optimization problem being solved, and so they may be usable for a **variety of problems**
- Metaheuristics **do not guarantee** that a globally **optimal solution** can be found on some class of problems
- Many metaheuristics implement some form of stochastic optimization, so that the solution found is dependent on the set of random variables generated
- In combinatorial optimization, by searching over a large set of feasible solutions, **metaheuristics can often find good solutions with less computational effort** than optimization algorithms or simple heuristics
- They are very useful approaches for **optimization problems**

Metaheuristics Global Analysis



Metaheuristics Properties

These are properties that characterize most metaheuristics:

- Metaheuristics are strategies that **guide the search process**
- The goal is to efficiently explore the search space in order to find near-optimal solutions
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes
- Metaheuristic algorithms are approximate and usually non-deterministic
- Metaheuristics are not problem-specific

Classification - Local search vs. global search

- One type of search strategy is an improvement on simple local search algorithms
- A well known local search algorithm is the hill climbing method which is used to find local optimums. However, hill climbing does not guarantee finding global optimum solutions
- Many metaheuristic ideas were proposed to improve local search heuristic in order to find better solutions
- Such metaheuristics include simulated annealing, tabu search, iterated local search, variable neighborhood search, and GRASP
- These metaheuristics can both be classified as local search-based or global search metaheuristics
- Other global search metaheuristic that are not local search-based are usually population-based metaheuristics
- Such metaheuristics include ant colony optimization, evolutionary computation, particle swarm optimization, and genetic algorithms

Classification - Single-solution vs. population-based

- Another classification dimension is single solution vs population-based searches.
- Single solution approaches focus on modifying and improving a single candidate solution;
- Single solution metaheuristics include simulated annealing, iterated local search, variable neighborhood search, and guided local search.

- Population-based approaches maintain and improve multiple candidate solutions, often using population characteristics to guide the search
- Population based metaheuristics include evolutionary computation, genetic algorithms and particle swarm optimization.
- Another category of metaheuristics related with Population-based is Swarm intelligence which is a collective behavior of decentralized, self-organized agents in a population or swarm. Ant colony, particle swarm, social cognitive optimization are examples of this category.

U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Artificial Intelligence/ Inteligência Artificial

Lecture 5: Optimization and Genetic Algorithms

Luís Paulo Reis

lpreis@fe.up.pt

Director of LIACC – Artificial Intelligence and Computer Science Lab.
Associate Professor at DEI/FEUP – Informatics Engineering Department,
Faculty of Engineering of the University of Porto, Portugal
President of APPIA – Portuguese Association for Artificial Intelligence