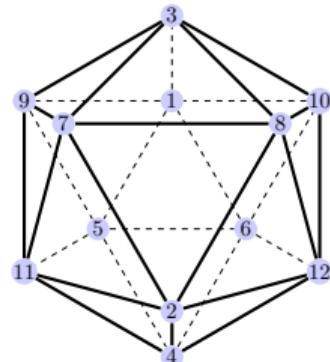


# 10 - Algoritmos Populacionais & Genéticos

Alexandre Checoli Choueiri

08/11/2025



## ① Algoritmos populacionais

- 1.1 Introdução
- 1.2 Pseudocódigo
- 1.3 Exemplos

## ② Algoritmos evolucionários

## ③ Algoritmos genéticos

## ④ AG e Estruturas de Dados

## ⑤ Exemplo - TSP

# Algoritmos populacionais

# Algoritmos populacionais

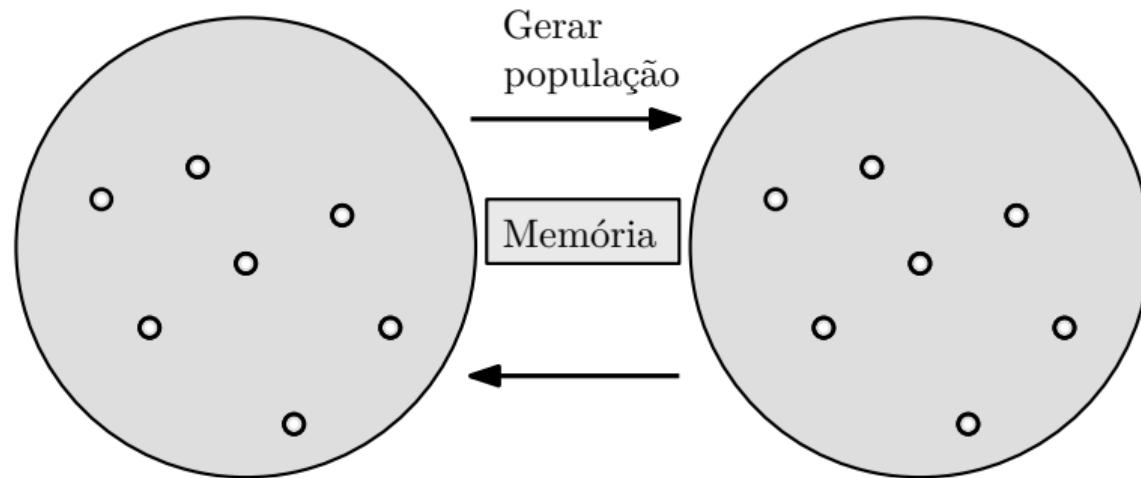
## Introdução

Como o nome já diz, **algoritmos populacionais** lidam com muitas soluções ao mesmo tempo (**população**), em que a cada iteração os elementos vão sendo melhorados.

Inicialmente uma população de soluções é gerada. Em uma segunda etapa novos elementos são criados e substituídos na população. O processo é repetido até que um critério de parada seja atingido.

# Algoritmos populacionais

## Introdução



Pode ou não existir um mecanismo de memória que armazena quais novos indivíduos foram adicionados à população em iterações anteriores.

# Algoritmos populacionais

## Pseudocódigo

Abaixo é mostrado um pseudocódigo de algoritmos populacionais.

---

### Algorithm 1 Template de algoritmo populacional

---

```
 $P = P_0$                                 ▷ Criação da pop. inicial  
 $t = 0$   
while Critério de parada não atingido do  
     $P'_t = \text{Gera-Pop}(P_t)$                 ▷ Gera novos elementos  
     $P_{t+1} = \text{Seleciona-Pop}(P_t \cup P'_t)$   
     $t = t + 1$   
end while  
return Melhor individuo da População.
```

---

# Algoritmos populacionais

## Exemplos

Como já mencionamos no estudo do algoritmo [Simulated Annealing](#), muitas metaheurísticas são bio-inspiradas, ou seja, buscam na natureza a idéia geral para o seu *design*. Com as metaheurísticas populacionais essa bio-inspiração é **mais acentuada**. Alguns exemplos de fenômenos naturais que inspiraram metaheurísticas:

1. O movimento de pulo dos sapos
2. A refração da luz
3. Músicos improvisando jazz (harmony search)
4. Evolução das espécies
5. Colônia de formigas
6. Movimento de morcegos, pássaros, moscas
7. Movimento dos planetas
8. Colonização de impérios...

# Algoritmos populacionais

## Exemplos

### Atenção

Existe uma "onda" na comunidade de otimização que tende para a criação de muitas "novas" metaheurísticas bio-inspiradas. Muitas delas não passam de casos específicos de outros métodos já desenvolvidos, porém com uma roupagem de novidade. Essa tendência é extremamente prejudicial para a área, portanto **tome cuidado com esses novos e inovadores métodos.**

Para uma **ótima** reflexão sobre o problema, ler os artigos:

1. 2012 - *Metaheuristics - the metaphor exposed* - Kenneth Sorensen
2. 2022 - *Exposing the greywolf moth flame whale firefly bat and antlion algorithms six misleading optimization techniques inspired by bestial metaphors* - Camacho

# Algoritmos populacionais

## Exemplos

### Atenção

Existe uma "onda" na comunidade de otimização que tende para a criação de muitas "novas" metaheurísticas bio-inspiradas. Muitas delas não passam de casos específicos de outros métodos já desenvolvidos, porém com uma roupagem de novidade. Essa tendência é extremamente prejudicial para a área, portanto **tome cuidado com esses novos e inovadores métodos.**

E para entender a dimensão do problema...**não ler** os artigos:

1. *2017 - Sperm motility algorithm a novel metaheuristic approach for global optimisation - raouf*

# Algoritmos populacionais

## Exemplos

### Atenção

Existe uma "onda" na comunidade de otimização que tende para a criação de muitas "novas" metaheurísticas bio-inspiradas. Muitas delas não passam de casos específicos de outros métodos já desenvolvidos, porém com uma roupagem de novidade. Essa tendência é extremamente prejudicial para a área, portanto **tome cuidado com esses novos e inovadores métodos.**

E para entender a dimensão do problema...**não ler** os artigos:

1. *2017 - Sperm motility algorithm a novel metaheuristic approach for global optimisation - raouf*
2. *2020 - Tiki-taka algorithm a novel metaheuristic inspired by football playing style - Mohd*

# Algoritmos populacionais

## Exemplos

### Atenção

Existe uma "onda" na comunidade de otimização que tende para a criação de muitas "novas" metaheurísticas bio-inspiradas. Muitas delas não passam de casos específicos de outros métodos já desenvolvidos, porém com uma roupagem de novidade. Essa tendência é extremamente prejudicial para a área, portanto **tome cuidado com esses novos e inovadores métodos.**

E para entender a dimensão do problema...**não ler** os artigos:

1. *2017 - Sperm motility algorithm a novel metaheuristic approach for global optimisation - raouf*
2. *2020 - Tiki-taka algorithm a novel metaheuristic inspired by football playing style - Mohd*
3. *2023 - Perfectionism Search Algorithm (PSA) An Efficient Meta-Heuristic - Amin*

# Algoritmos populacionais

## Exemplos

Os duas inspirações que estão destacadas na lista abaixo podem ser consideradas como verdadeiras inovações, que de fato geraram algoritmos novos e com conceitos que agregaram para o **avanço da pesquisa** na área.

1. O movimento de pulo dos sapos
2. A refração da luz
3. Músicos improvisando jazz (harmony search)
4. Evolução das espécies
5. Colônia de formigas
6. Movimento de morcegos, pássaros, moscas
7. Movimento dos planetas
8. Colonização de impérios...

# Algoritmos populacionais

## Exemplos

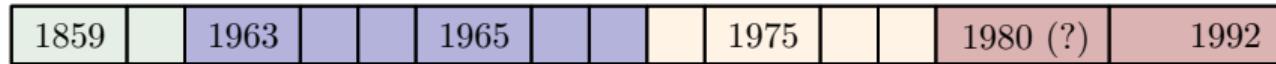
Nós estudaremos os algoritmos inspirados na **evolução das espécies**.

1. O movimento de pulo dos sapos
2. A refração da luz
3. Músicos improvisando jazz (harmony search)
4. **Evolução das espécies**
5. Colônia de formigas
6. Movimento de morcegos, pássaros, moscas
7. Movimento dos planetas
8. Colonização de impérios...

# Algoritmos evolucionários

# Algoritmos evolucionários

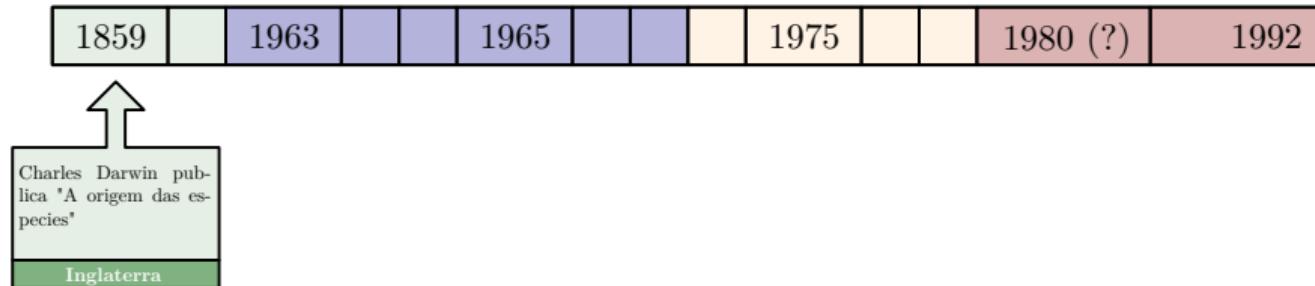
## Linha do tempo



Vejamos uma linha do tempo dos algoritmos inspirados na evolução das espécies, hoje chamados de **algoritmos evolucionários**.

# Algoritmos evolucionários

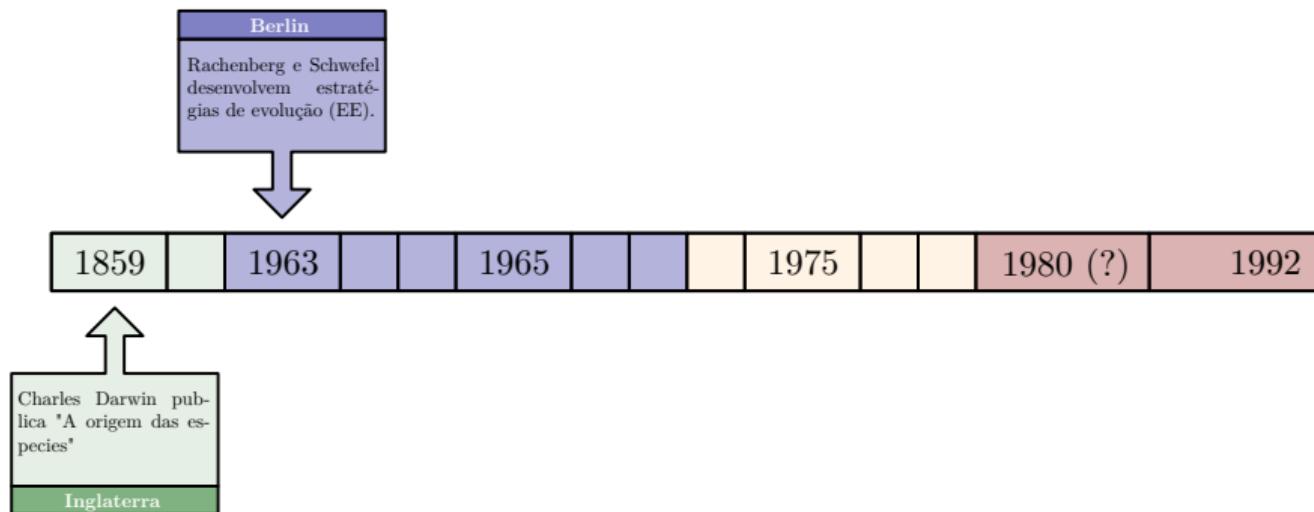
## Linha do tempo



A grande inspiração para todos os algoritmos foi a revolucionária teoria de Darwin sobre como as espécies evoluem ao longo do tempo.

# Algoritmos evolucionários

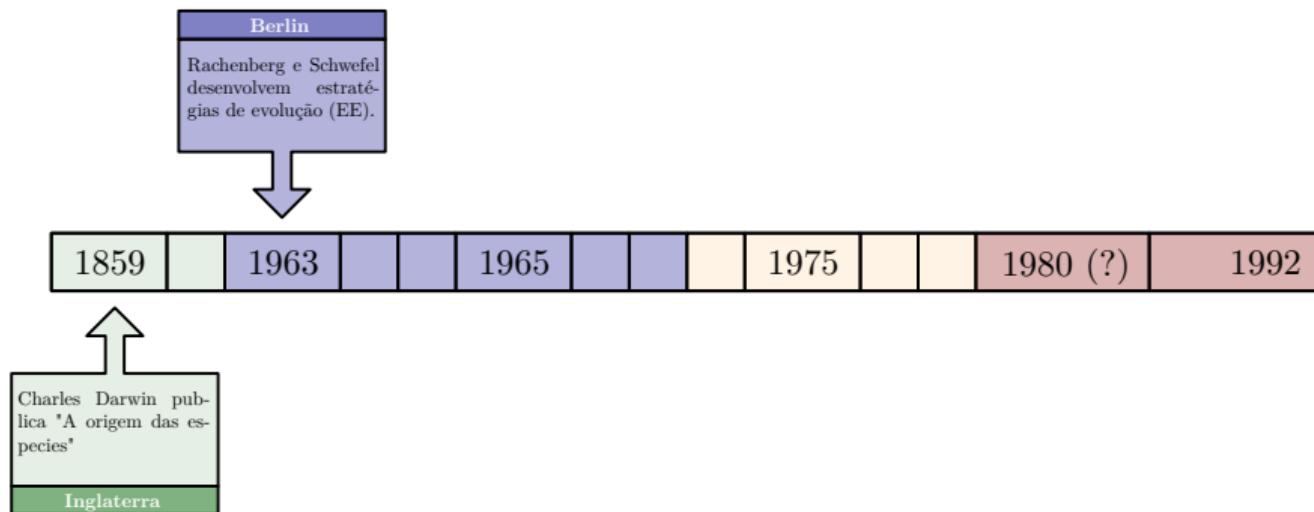
## Linha do tempo



Em Berlin, Rachenberg e Schwefel desenvolvem um modelo de algoritmos chamados **estratégias de evolução (EE)**.

# Algoritmos evolucionários

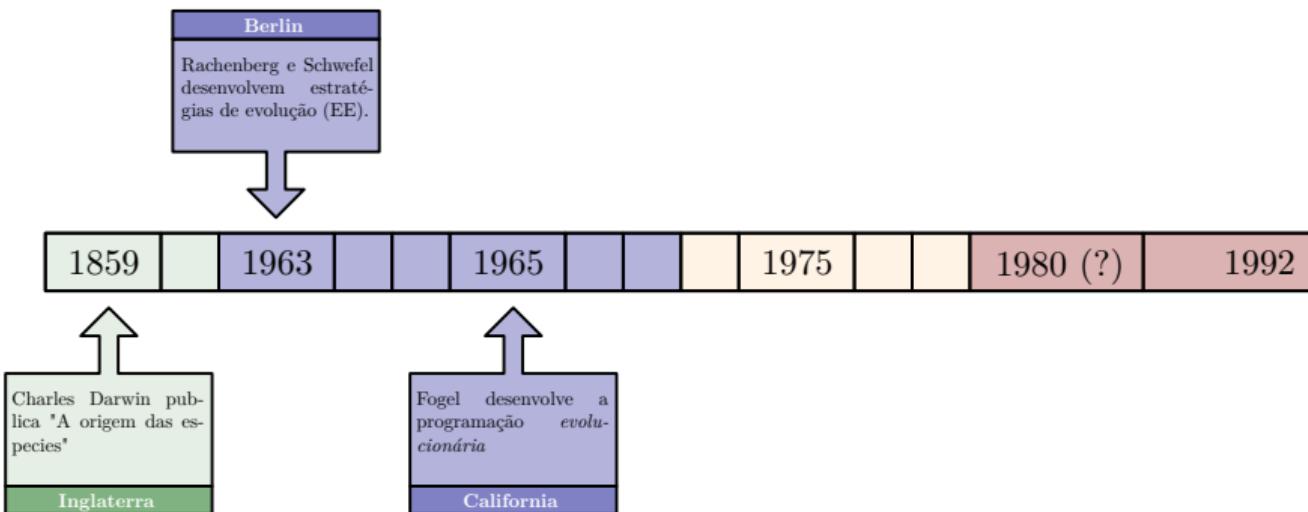
## Linha do tempo



Esses algoritmos foram usados para resolver problemas de mecânica dos fluidos, especificamente em túneis de vento para a otimização de placas que minimizam a resistência ao vento.

# Algoritmos evolucionários

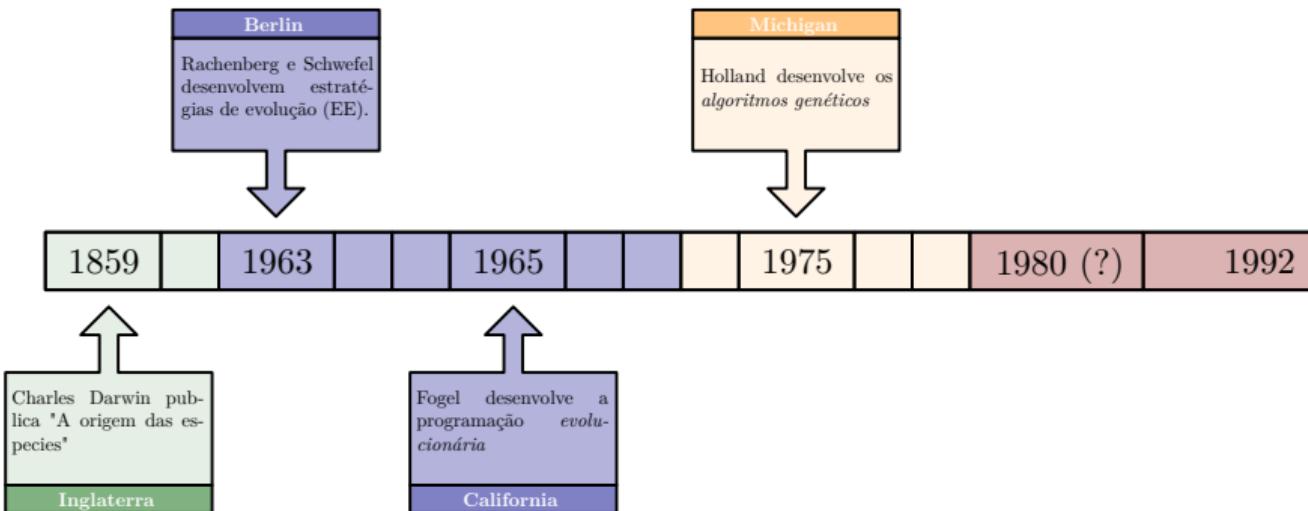
## Linha do tempo



Na Califórnia, Fogel desenvolve a estrutura evolucionária chamada **programação evolucionária** (PE), usado em máquinas de estado finitos com o objetivo de gerar inteligência artificial.

# Algoritmos evolucionários

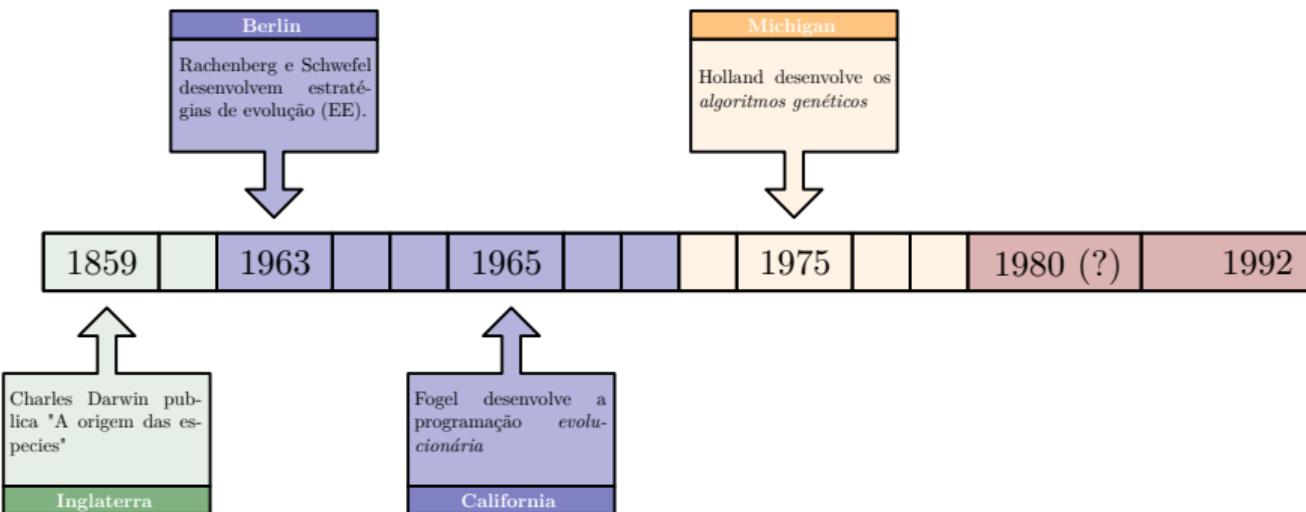
## Linha do tempo



Em Michigan, Holland publica seu livro sobre **algoritmos genéticos** (AG).

# Algoritmos evolucionários

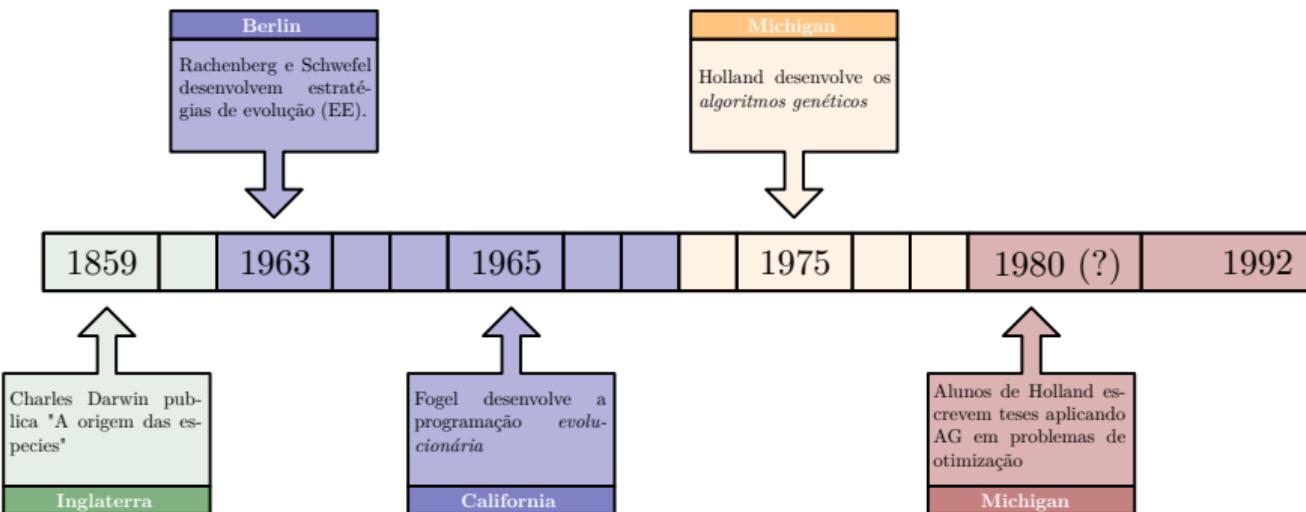
## Linha do tempo



Originalmente os AG não foram usados em problemas de otimização, mas sim na determinação de um número ideal de observações das variáveis aleatórias.

# Algoritmos evolucionários

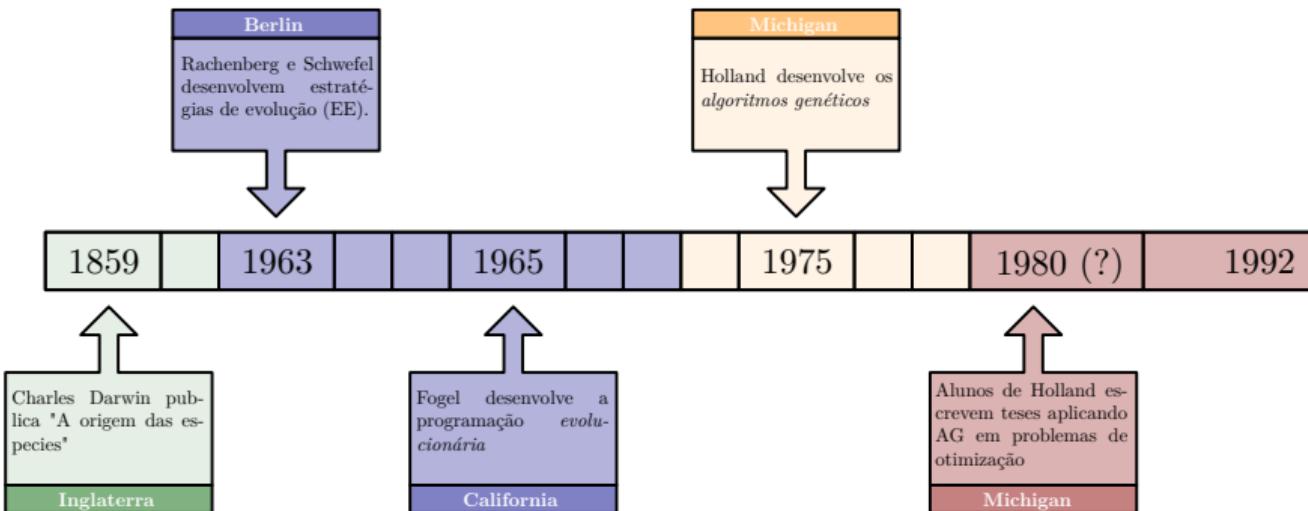
## Linha do tempo



Por sorte, seus alunos de doutorado começaram a usar seu método para resolver problemas de otimização.

# Algoritmos evolucionários

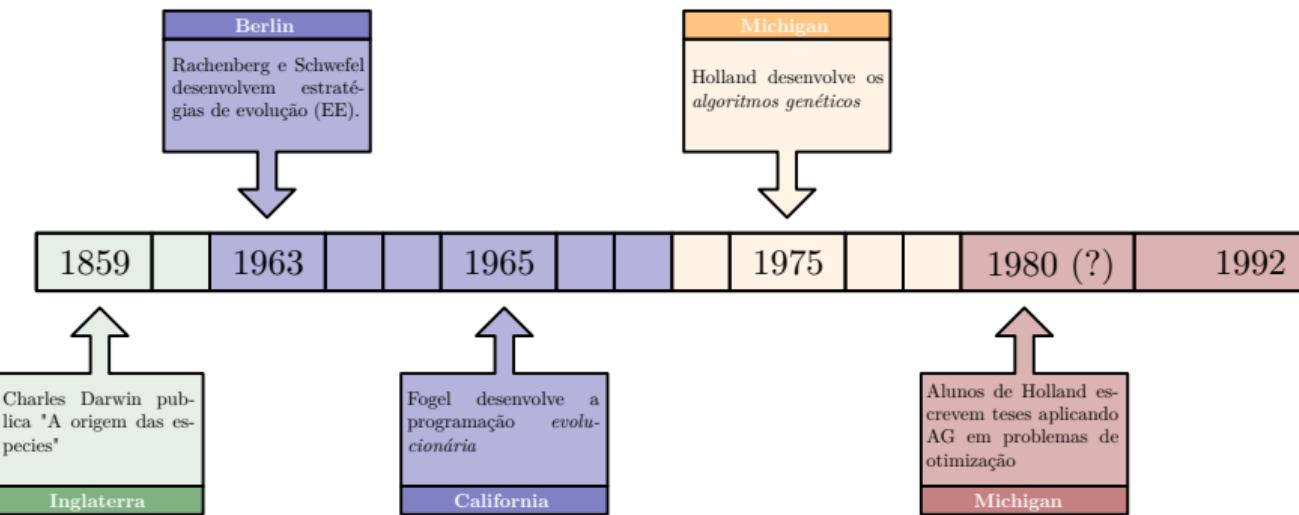
## Linha do tempo



Até esse momento as pesquisas eram realizadas de forma quase que independente, com pouca interação entre os grupos de pesquisa.

# Algoritmos evolucionários

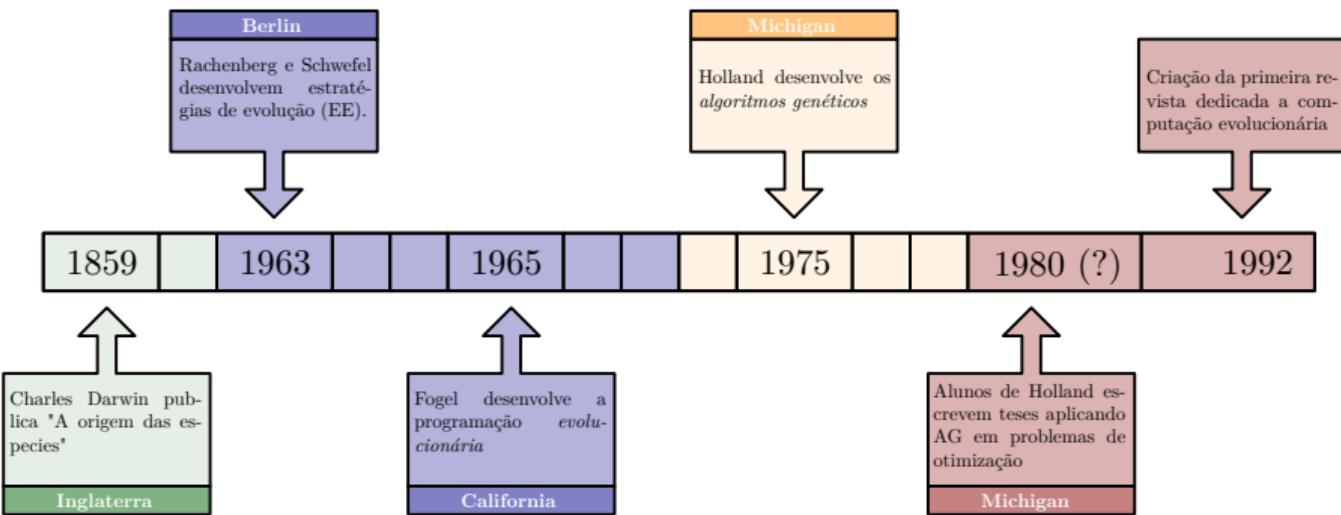
## Linha do tempo



No fim da década 80 os pesquisadores começaram a se encontrar em congressos e a trocar conhecimentos.

# Algoritmos evolucionários

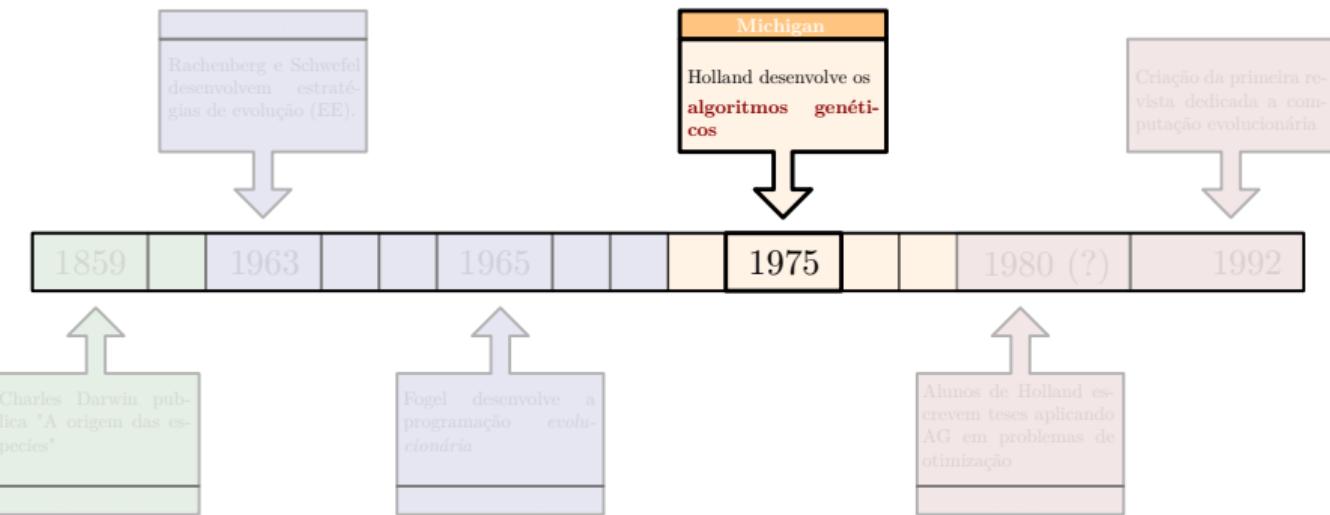
## Linha do tempo



Em 1992 a área é consolidada com o termo **computação evolucionária**, e a primeira revista sobre o tema é criada, chamada *Evolutionary Computing*.

# Algoritmos evolucionários

## Linha do tempo

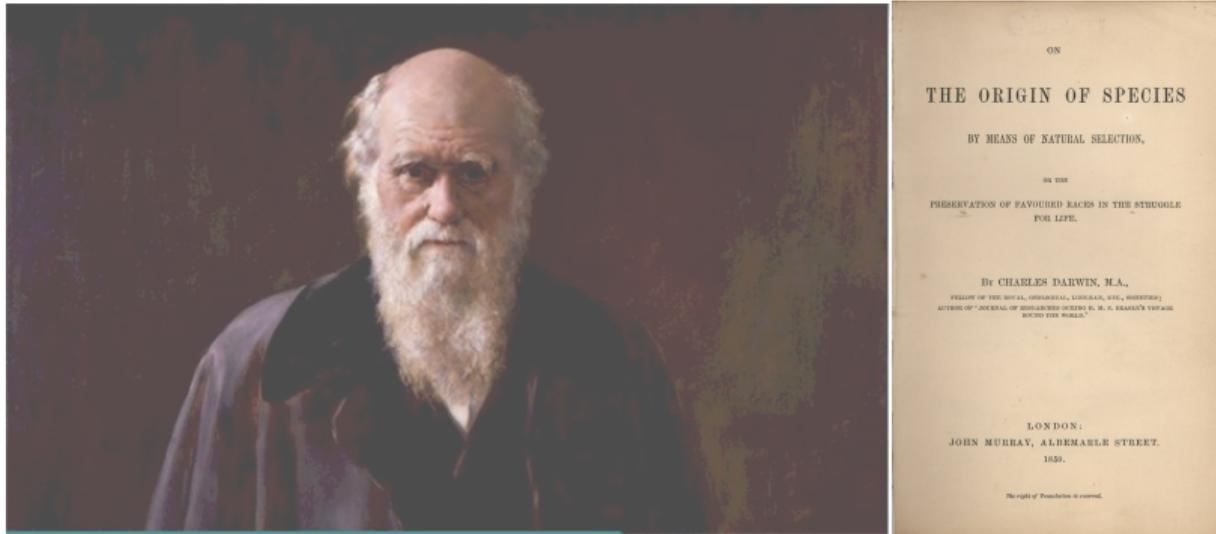


Nós estudaremos justamente os **algoritmos genéticos** de Holland e seus alunos.

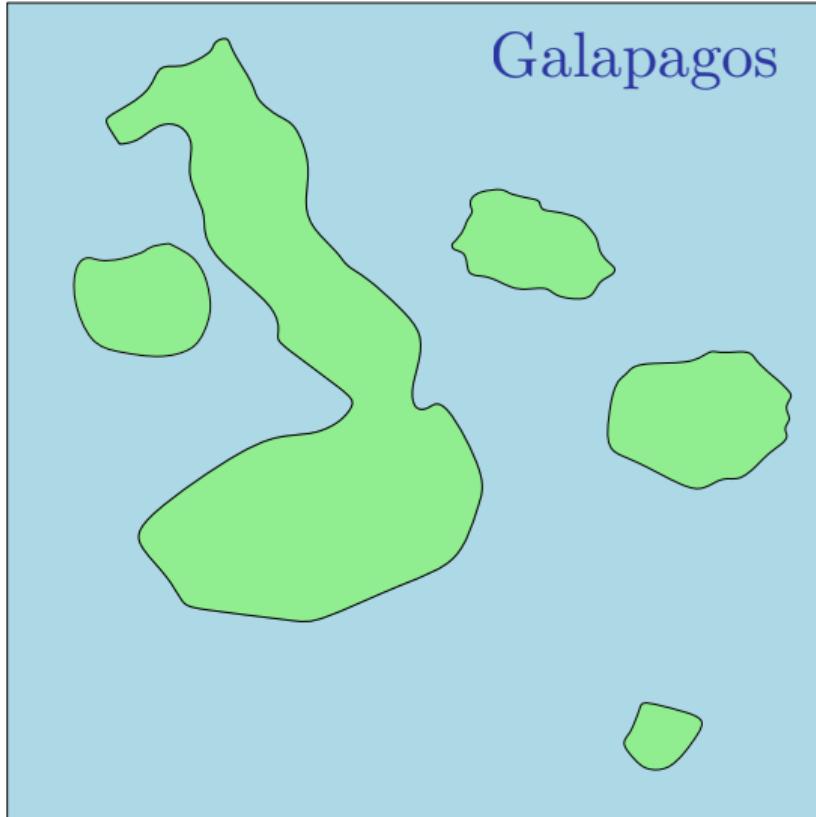
# Algoritmos genéticos

# Algoritmos genéticos

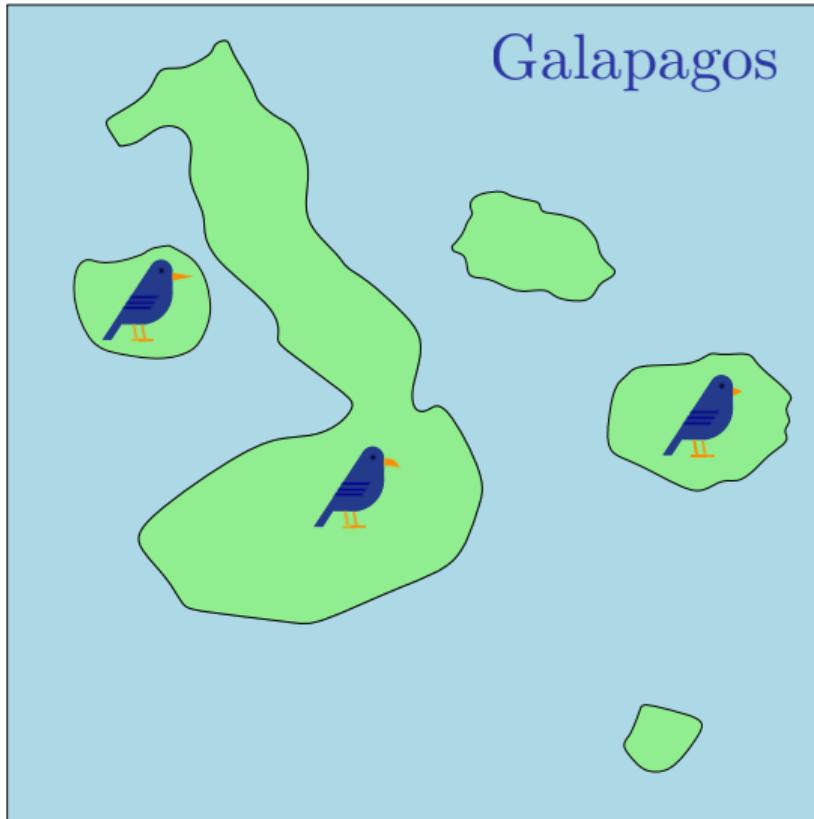
## A evolução segundo Darwin



Para isso, precisamos primeiro entender a dinâmica básica da teoria da evolução de *Charles Darwin*.



É sabido que Darwin fez diversas incursões pelo mundo a bordo do *Beagle*, e nessas incursões ele estudava fauna e flora dos locais, com anotações e desenhos. Um dos lugares que ele visitou foi Galápagos e suas diversas ilhas.

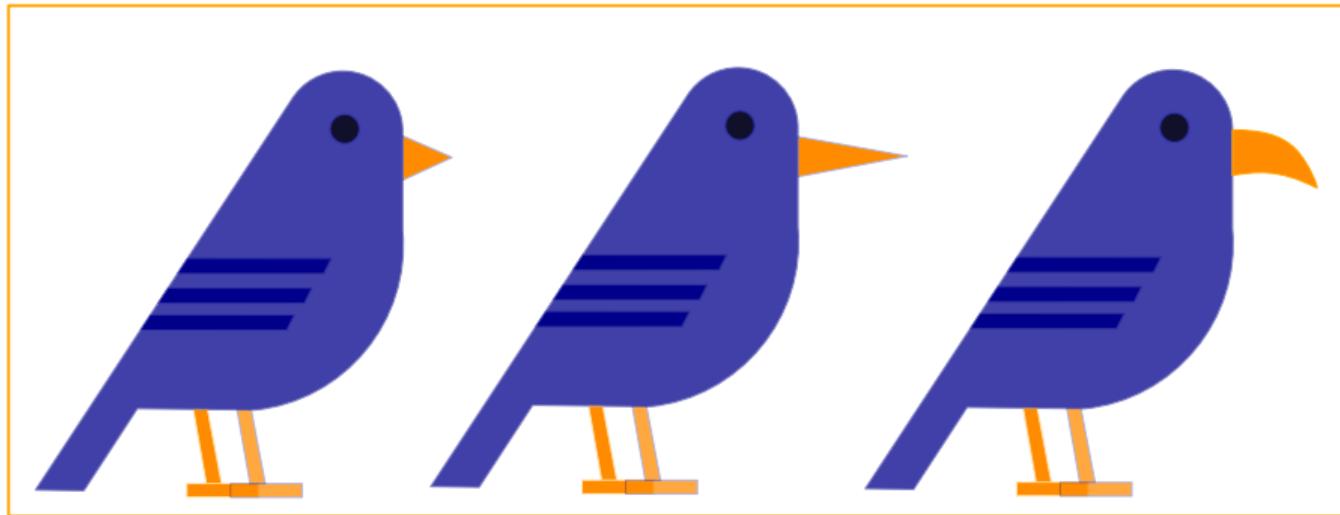


## Galapagos

Em Galápagos, Darwin reparou que em diversas ilhas existiam espécies de pássaros muito similares entre si. A única diferença notável era no formato de seus bicos.

# Algoritmos genéticos

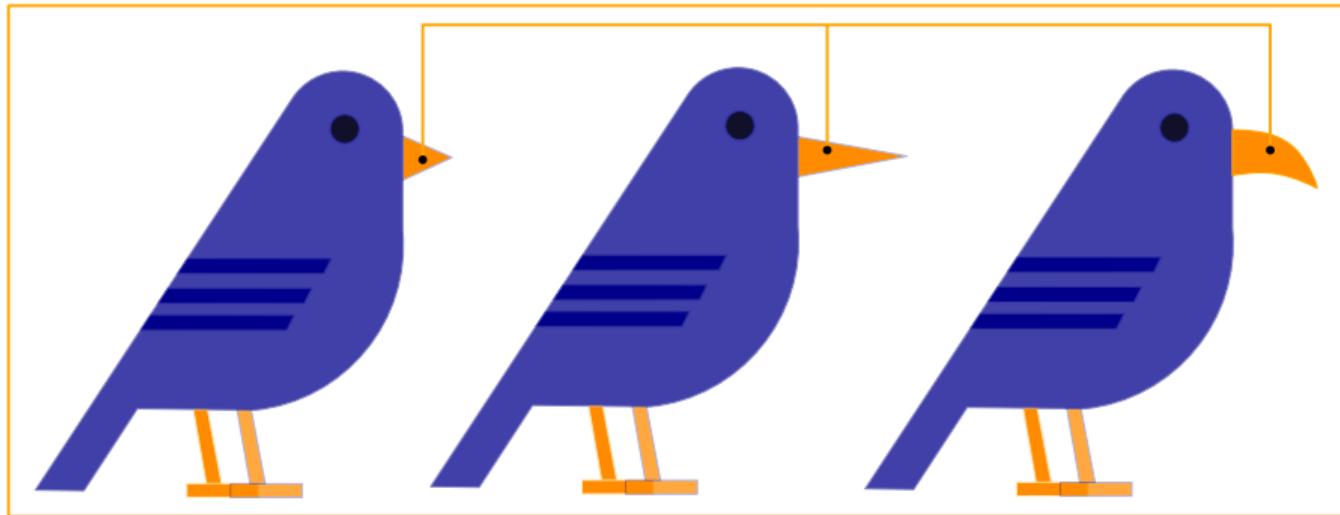
A evolução segundo Darwin



Os pássaros de cada ilha aparentavam ter um tipo de bico próprio para coletar alimentos específicos daquela ilha.

# Algoritmos genéticos

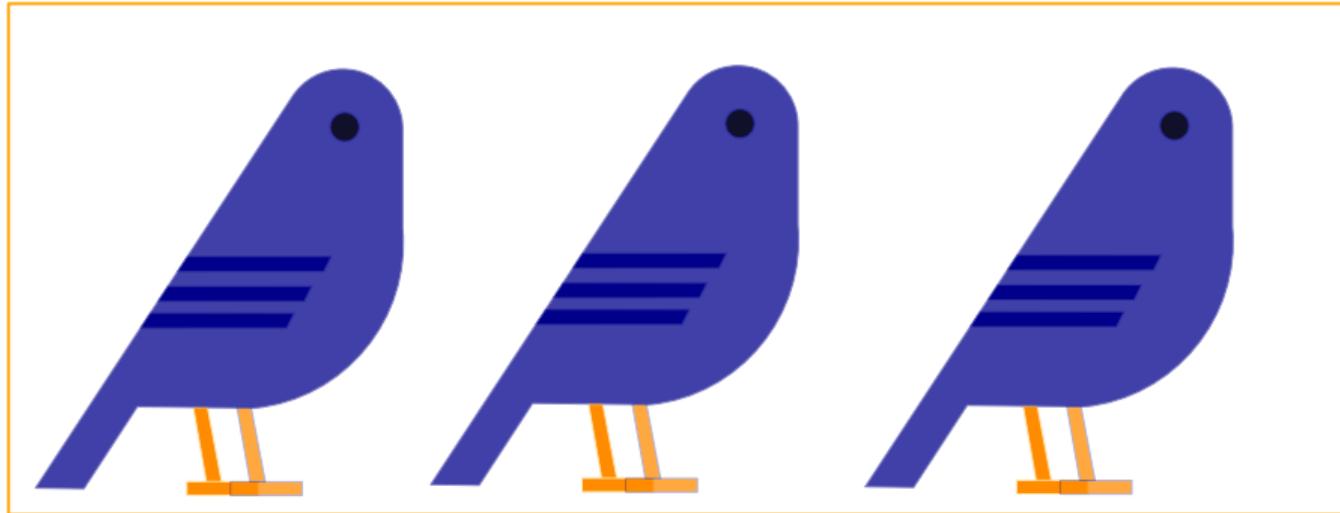
A evolução segundo Darwin



Darwin reparou que apesar de possuírem bicos diferentes...

# Algoritmos genéticos

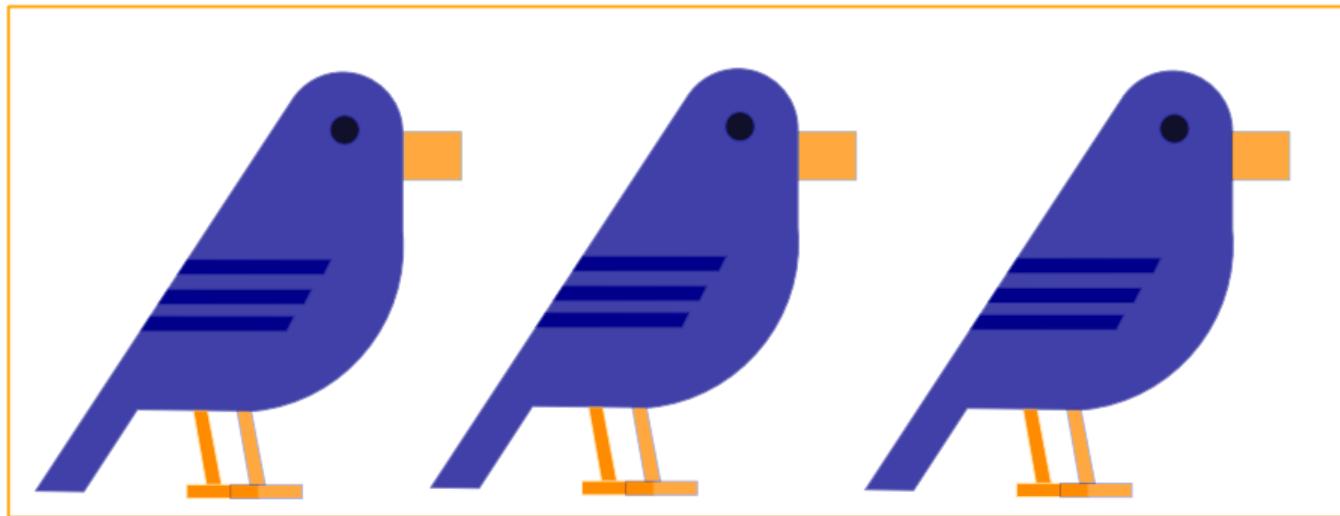
A evolução segundo Darwin



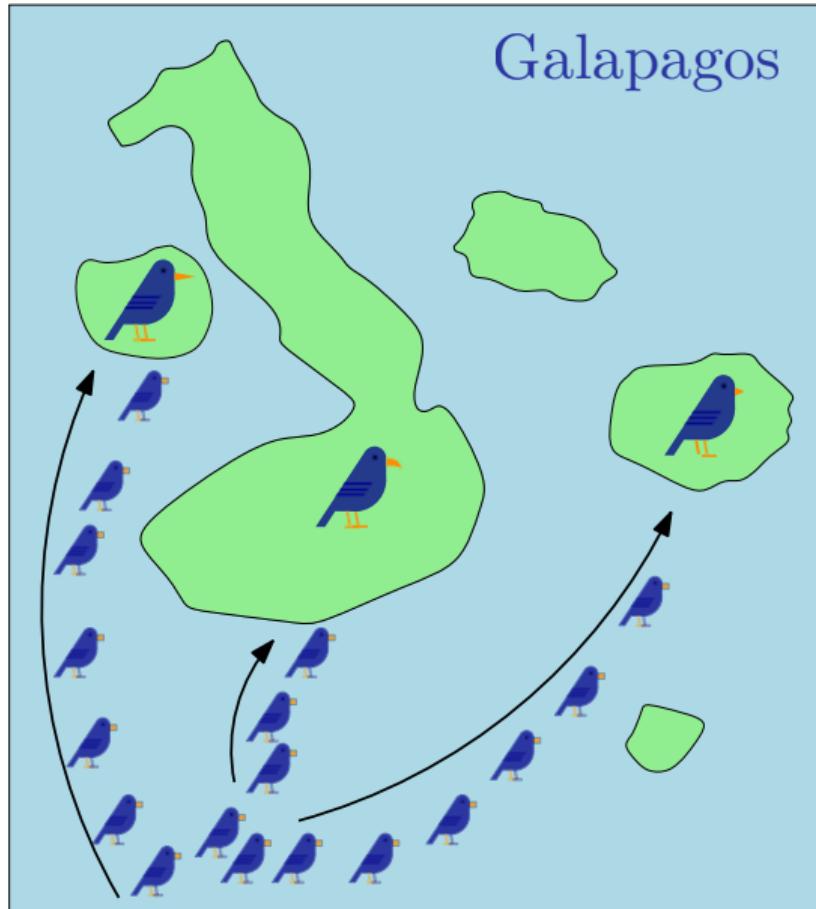
Os pássaros eram **muito semelhantes entre si!**

# Algoritmos genéticos

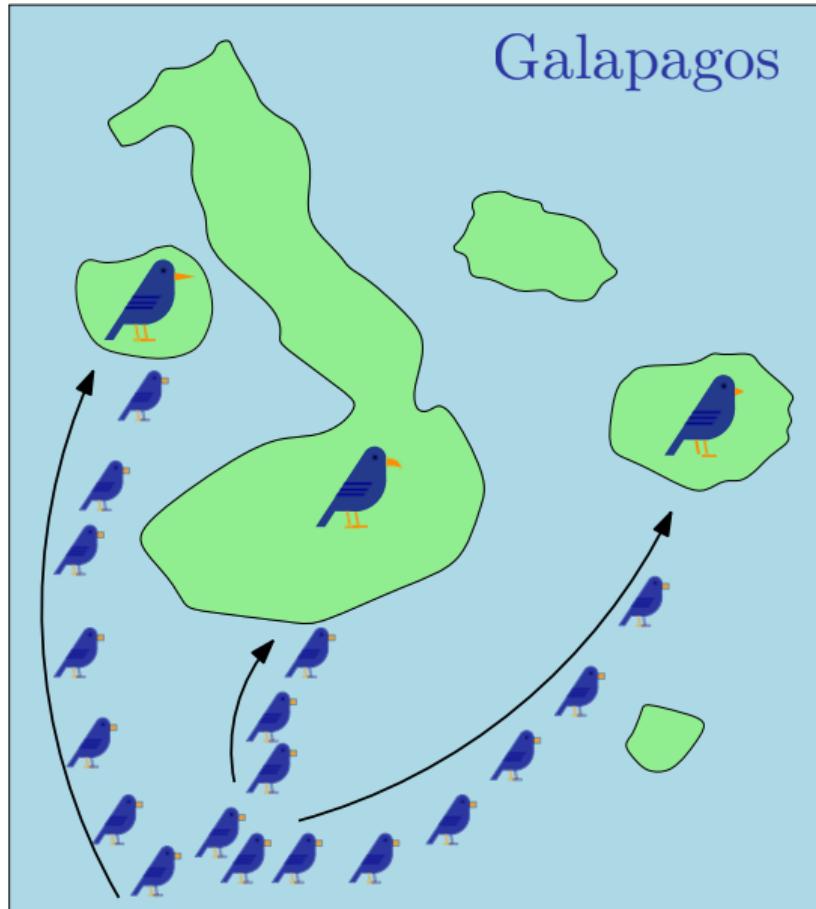
A evolução segundo Darwin



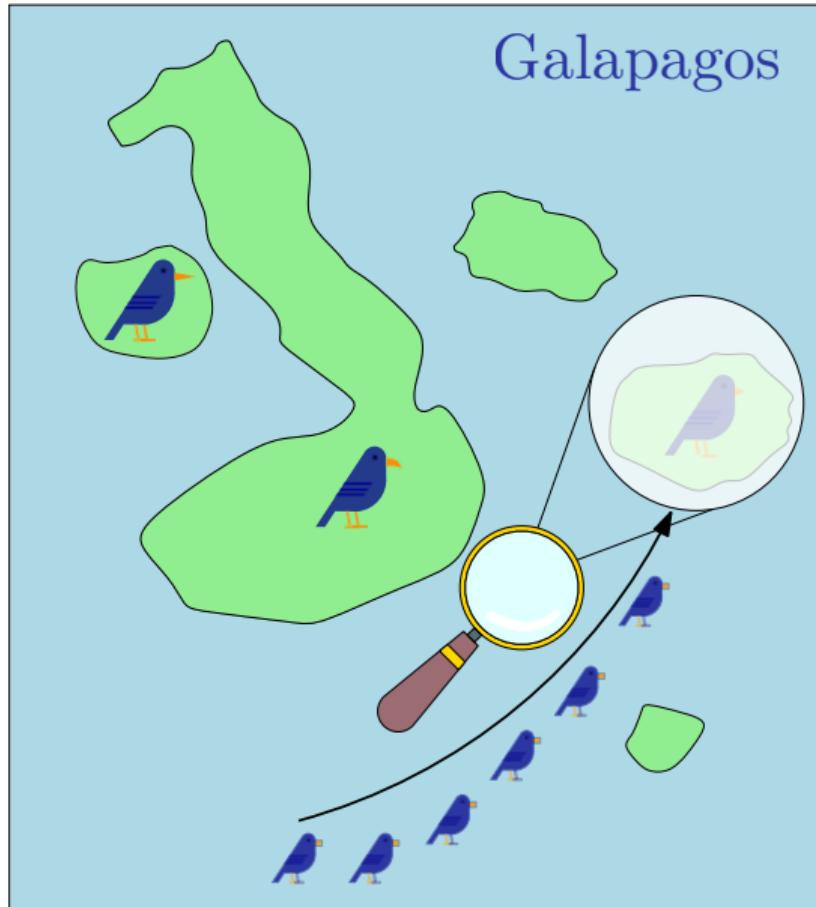
E isso o fez pensar em uma conjectura: **Será que todos os pássaros vieram de um ancestral comum?**



Para sustentar a conjectura, Darwin teria que explicar como um mesmo tipo de pássaro teria migrado para as ilhas, e **ao longo do tempo se diferenciado**.



A explicação para a conjectura de Darwin é conhecida como a **Teoria da Seleção Natural** (ou evolução das espécies).

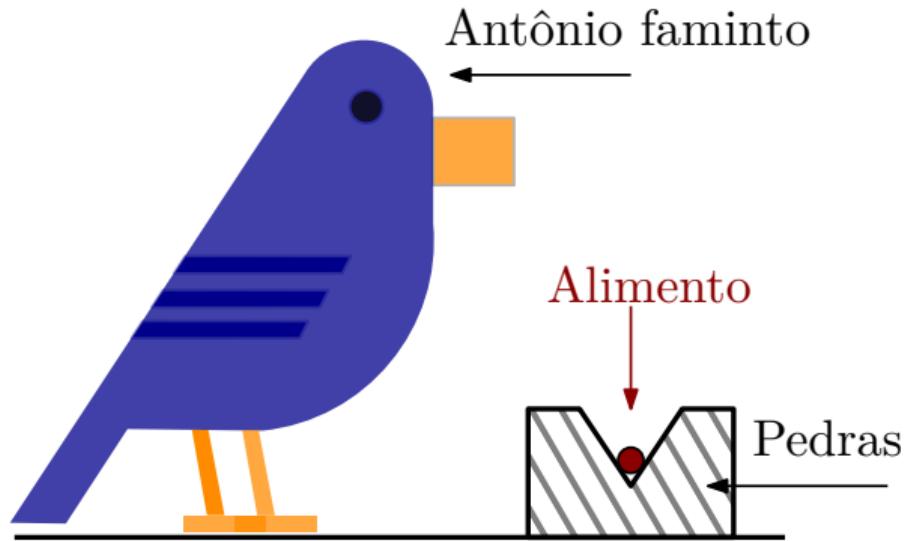


## Galapagos

Vejamos qual é a dinâmica da Teoria da Seleção Natural, em uma das ilhas de Galápagos.

# Algoritmos genéticos

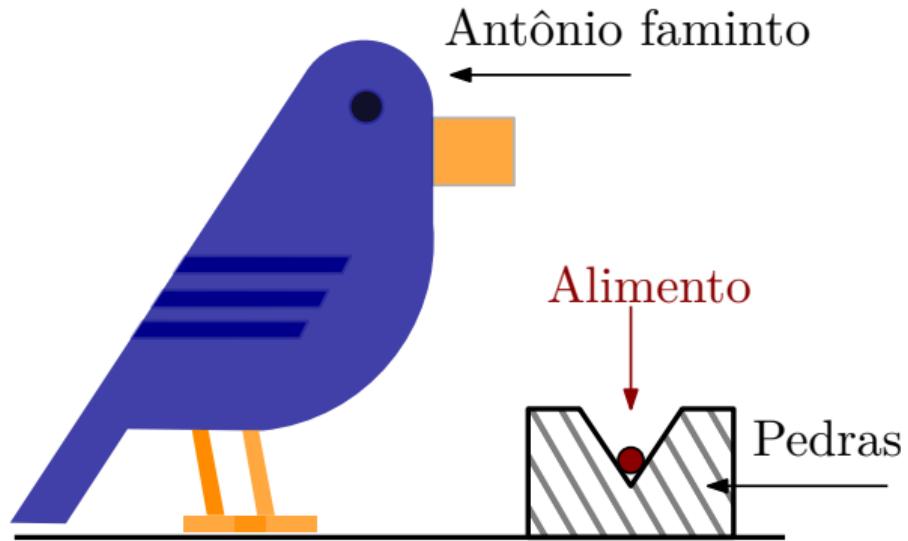
A evolução segundo Darwin



Suponha que a ilha é muito pedregosa, de forma que o alimento do pássaro Antônio (que está faminto) cai e se posiciona como na Figura acima.

# Algoritmos genéticos

## A evolução segundo Darwin

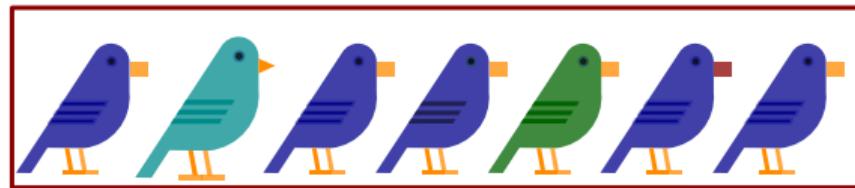


Podemos afirmar por mera observação, que o pássaro Antônio não está muito **Apto ou Adaptado** para se alimentar nesta ilha.

# Algoritmos genéticos

A evolução segundo Darwin

População inicial

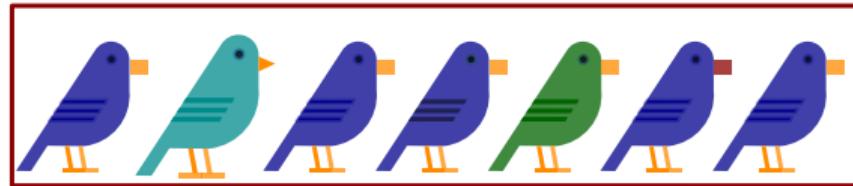


Inicialmente uma população desses ancestrais chegou a ilha. Note que embora a maioria tenha o bico característico, devido à **mutação** pode existir uma pequena parte com variações.

# Algoritmos genéticos

A evolução segundo Darwin

População inicial

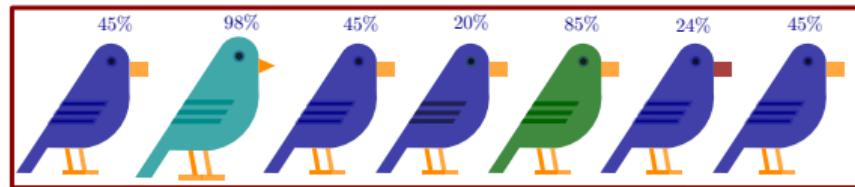


Embora eles sejam do mesmo tipo, cada um possui **características** diferentes além do bico, nesse caso as cores e/ou tamanhos.

# Algoritmos genéticos

A evolução segundo Darwin

População inicial

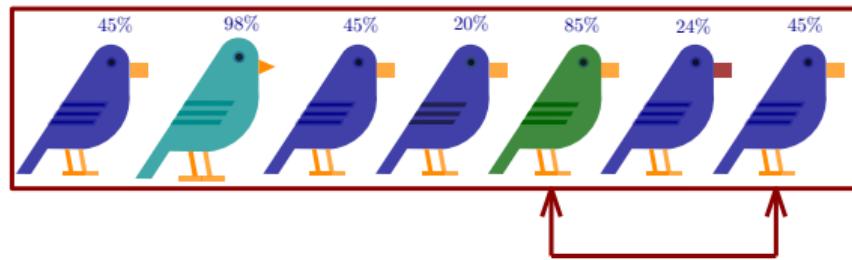


Cada uma das características ajuda na **adaptabilidade** do indivíduo ao ambiente.  
Vamos chamar essa pontuação de **fitness** do indivíduo.

# Algoritmos genéticos

A evolução segundo Darwin

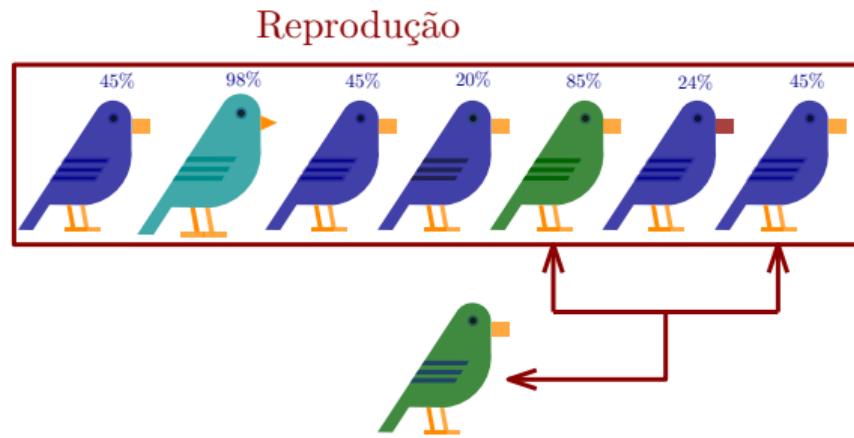
Seleção para reprodução



Com o tempo os indivíduos da população se reproduzem. Como ocorre com os seres humanos, existe uma **competição** para a reprodução.

# Algoritmos genéticos

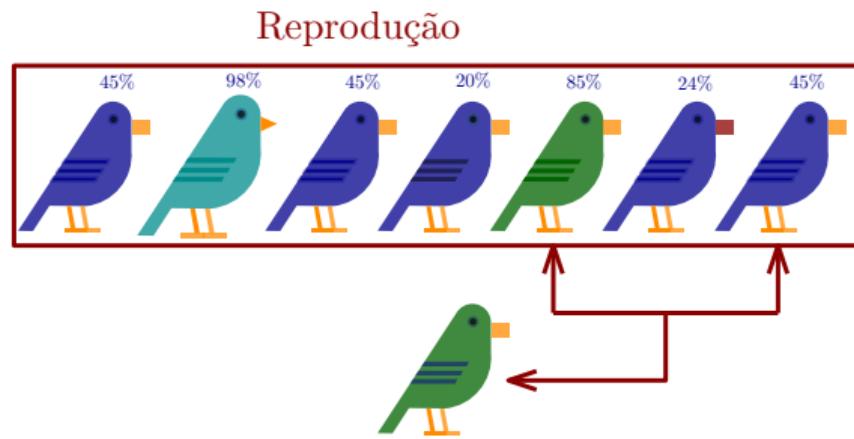
A evolução segundo Darwin



E a reprodução gera novos indivíduos para a população.

# Algoritmos genéticos

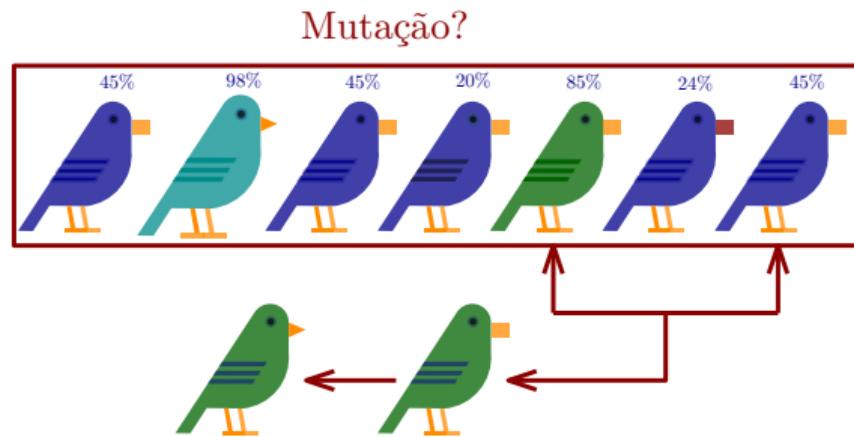
A evolução segundo Darwin



Existe uma probabilidade pequena de ocorrer uma mutação nesses novos indivíduos (como o albinismo ou nanismo em seres humanos).

# Algoritmos genéticos

A evolução segundo Darwin



Essa mutação altera alguma característica do indivíduo.

# Algoritmos genéticos

A evolução segundo Darwin



Com esse novo indivíduo temos uma nova população (1 individuo maior do que a população inicial.)

# Algoritmos genéticos

A evolução segundo Darwin

Seleção natural (substituição)



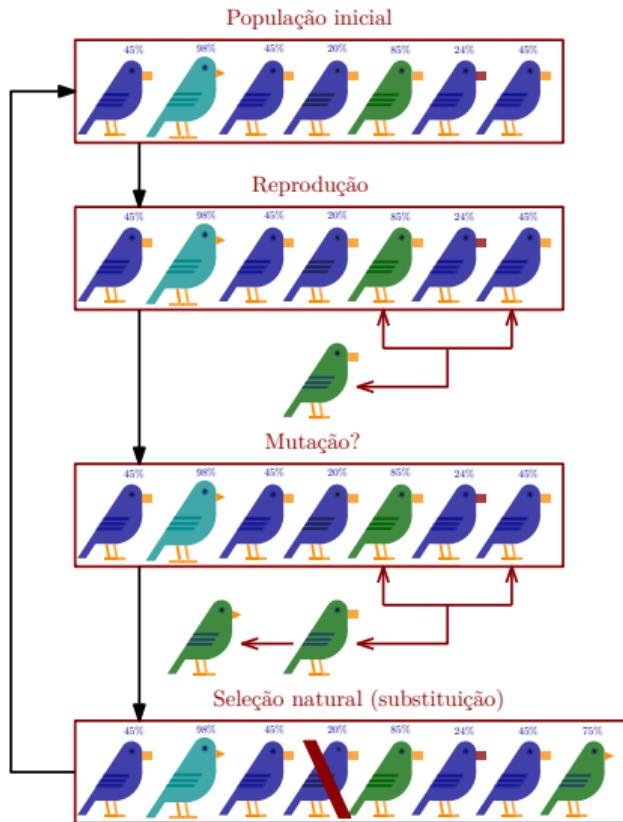
Os indivíduos menos aptos (com menor **fitness**) são mais propensos a morrer (porém existe pequena chance de sobreviverem).

# Algoritmos genéticos

A evolução segundo Darwin



E assim uma nova população repete o mesmo ciclo.



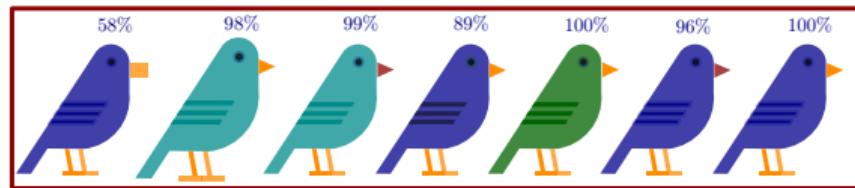
O ciclo completo é mostrado ao lado. Consiste nas etapas:

1. Geração da pop. inicial
2. Reprodução
3. Mutação
4. Seleção natural

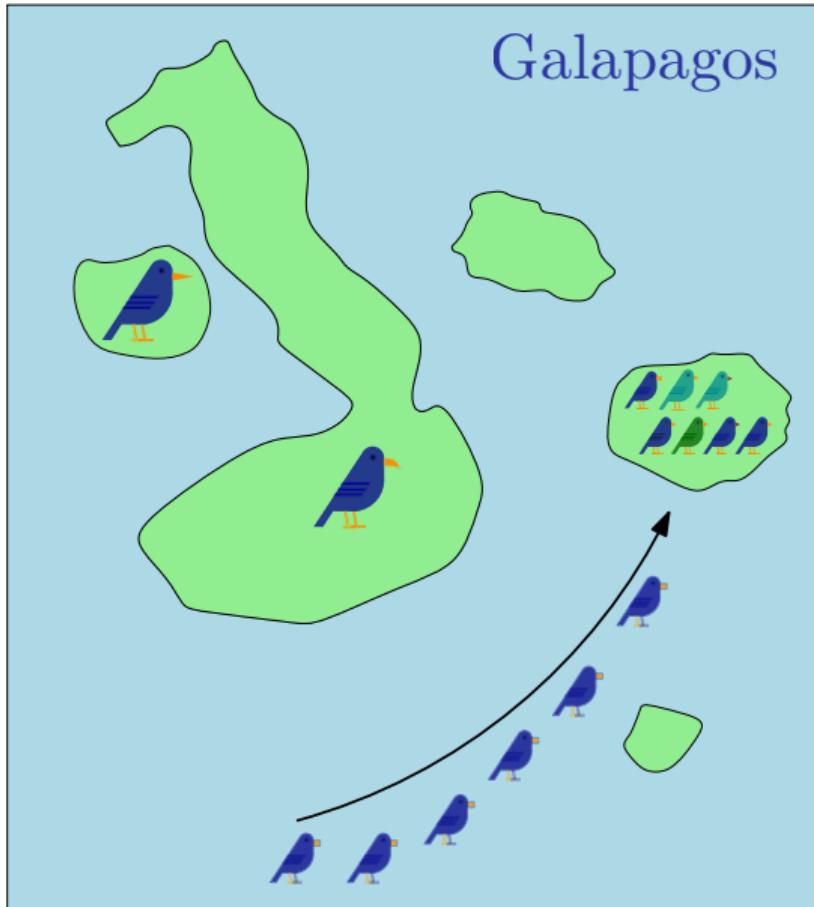
# Algoritmos genéticos

## A evolução segundo Darwin

População final

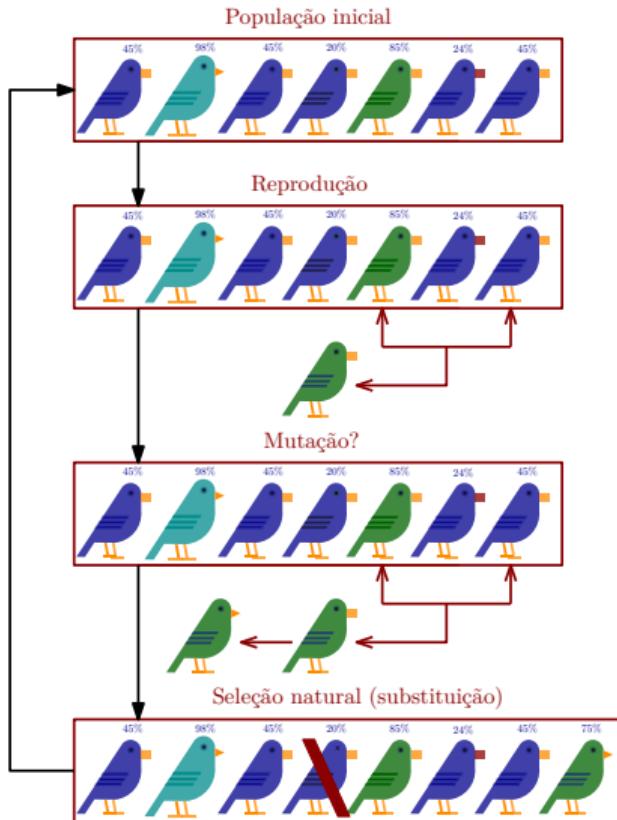


Segundo Darwin, após um intervalo de tempo suficiente, com muita mutação e seleção natural, a população se torna a mais apta para habitar naquele ambiente (com as características de comida que vimos). Nesse caso a característica predominante seria o formato do bico.

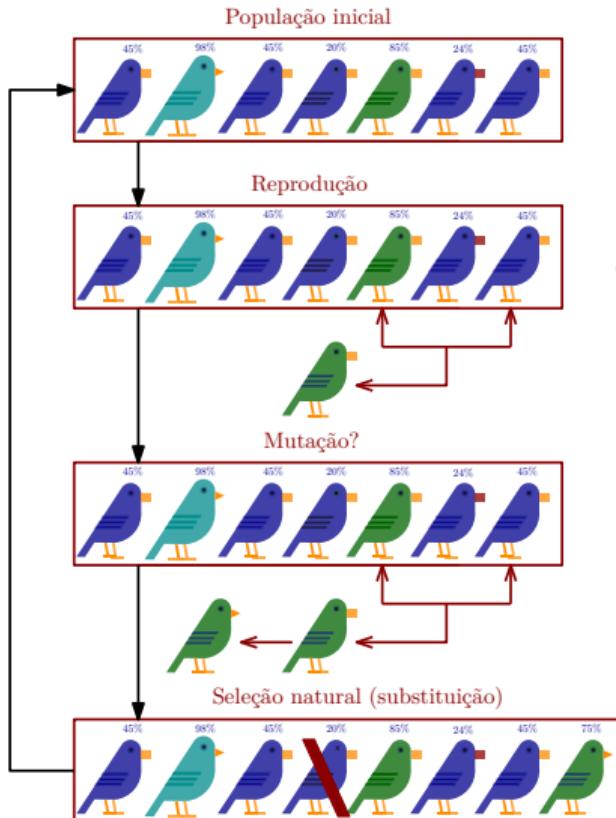


## Galapagos

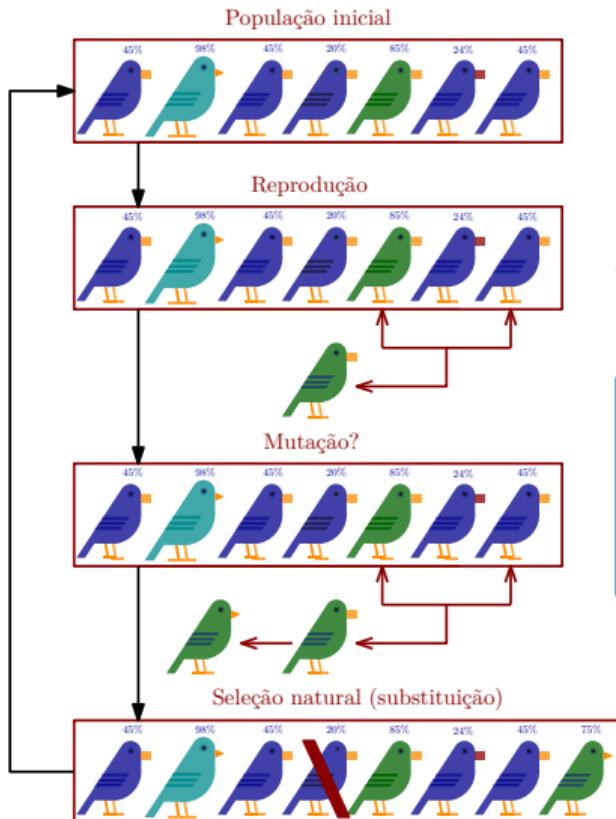
E assim se explicaria a conjectura de Darwin de como **todos os pássaros** de todas as ilhas possuíam um **ancestral comum**.



MAS O QUE ISSO TÊM A VER COM ALGORITMOS DE OTIMIZAÇÃO?

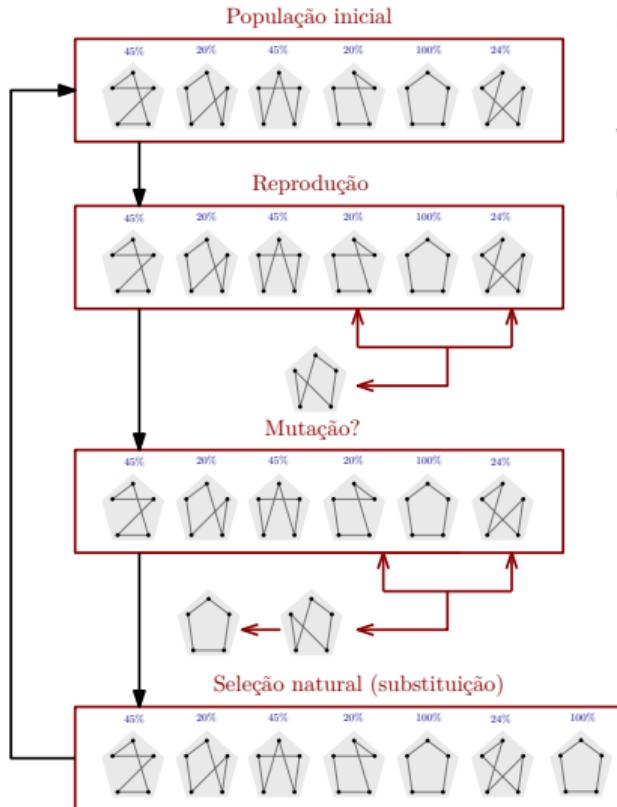


Podemos interpretar a teoria da seleção natural de Darwin como um mecanismo de **maximização** da adaptabilidade (*fitness*) dos indivíduos de uma população.

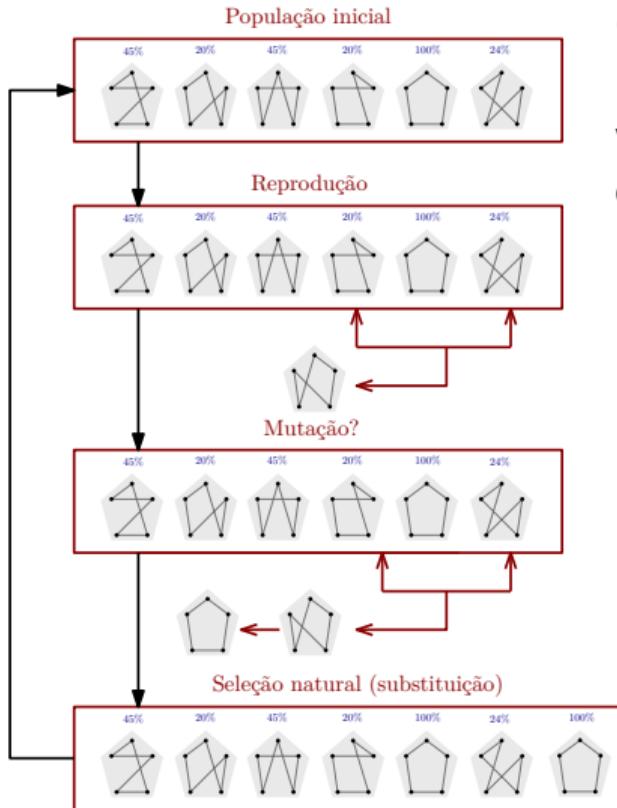


Podemos interpretar a teoria da seleção natural de Darwin como um mecanismo de **maximização** da adaptabilidade (*fitness*) dos indivíduos de uma população.

Essa ideia então pode ser usada em problemas de otimização, e é a partir dela que surgem os **algoritmos genéticos** de Holland.

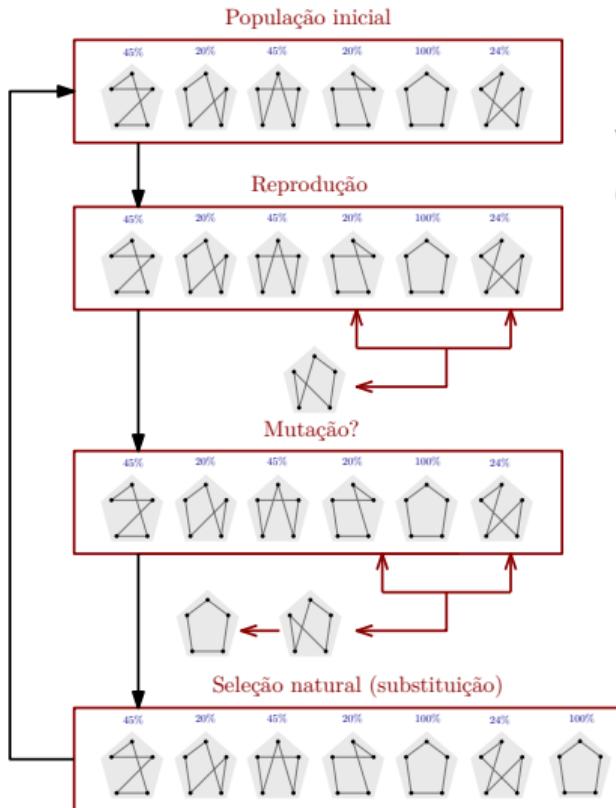


Considere a Figura ao lado como sendo as etapas de um algoritmo genético para o problema do caixeiro viajante. Temos as seguintes associações com a teoria de Darwin:



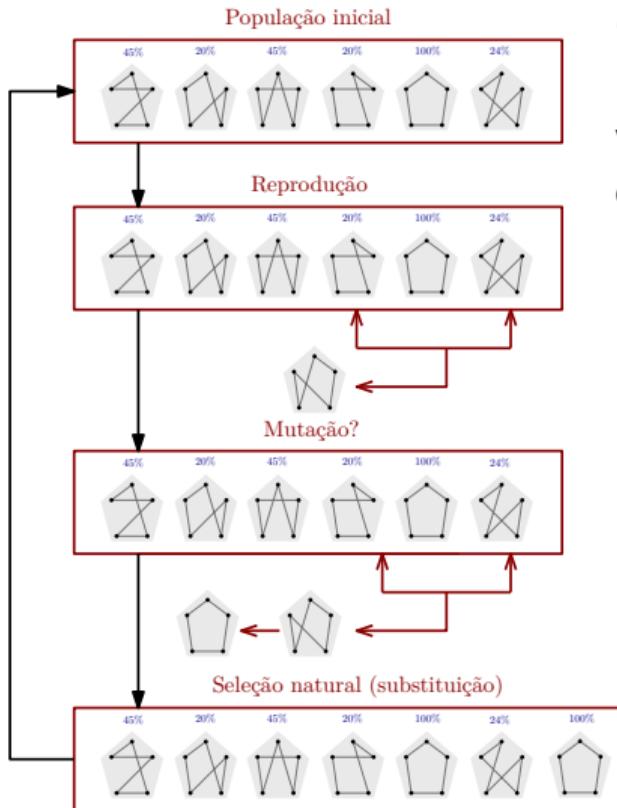
Considere a Figura ao lado como sendo as etapas de um algoritmo genético para o problema do caixeiro viajante. Temos as seguintes associações com a teoria de Darwin:

1. Individuo da população → solução do TSP.



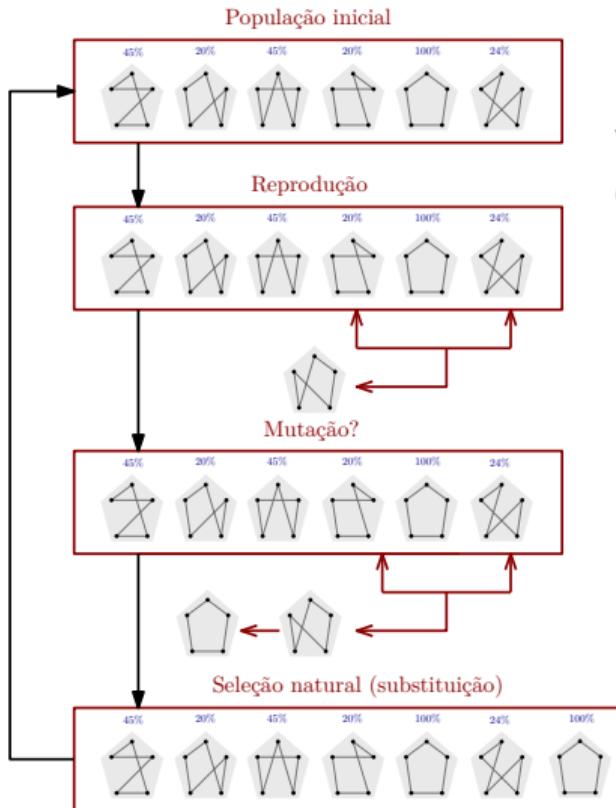
Considere a Figura ao lado como sendo as etapas de um algoritmo genético para o problema do caixeiro viajante. Temos as seguintes associações com a teoria de Darwin:

1. Individuo da população → solução do TSP.
2. Fitness → medida de valoração da solução (pode ser a fo).



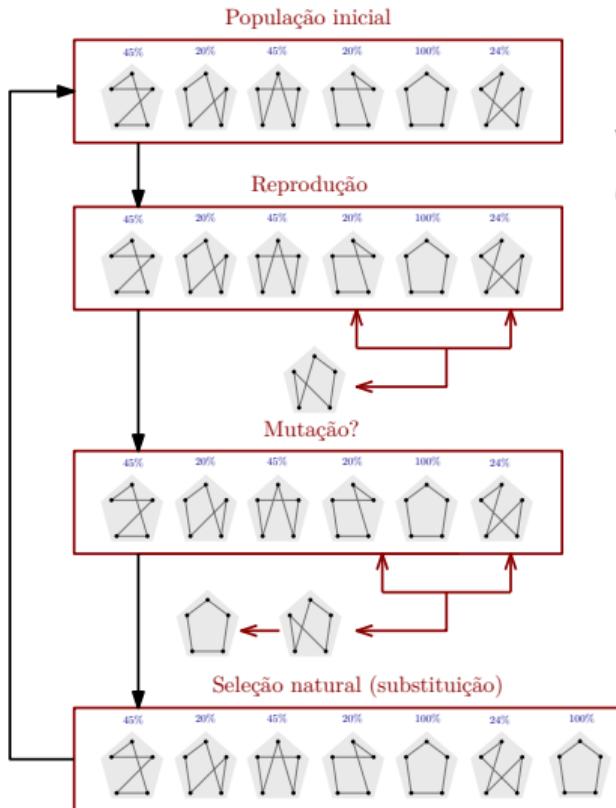
Considere a Figura ao lado como sendo as etapas de um algoritmo genético para o problema do caixeiro viajante. Temos as seguintes associações com a teoria de Darwin:

1. Individuo da população → solução do TSP.
2. Fitness → medida de valoração da solução (pode ser a fo).
3. Reprodução → método para selecionar e combinar duas soluções.



Considere a Figura ao lado como sendo as etapas de um algoritmo genético para o problema do caixeiro viajante. Temos as seguintes associações com a teoria de Darwin:

1. **Individuo da população** → solução do TSP.
2. **Fitness** → medida de valoração da solução (pode ser a fo).
3. **Reprodução** → método para selecionar e combinar duas soluções.
4. **Mutação** → pequena alteração na solução.



Considere a Figura ao lado como sendo as etapas de um algoritmo genético para o problema do caixeiro viajante. Temos as seguintes associações com a teoria de Darwin:

1. **Individuo da população** → solução do TSP.
2. **Fitness** → medida de valoração da solução (pode ser a fo).
3. **Reprodução** → método para selecionar e combinar duas soluções.
4. **Mutação** → pequena alteração na solução.
5. **Seleção natural** → como excluir soluções ruins da população.

# Algoritmos populacionais

## No fitness landscape

Sempre precisamos entender como os algoritmos de otimização se comportam no *fitness landscape*, pois de forma geral é isso que determina o *design* do método.

# Algoritmos populacionais

## No fitness landscape

Sempre precisamos entender como os algoritmos de otimização se comportam no *fitness landscape*, pois de forma geral é isso que determina o *design* do método.

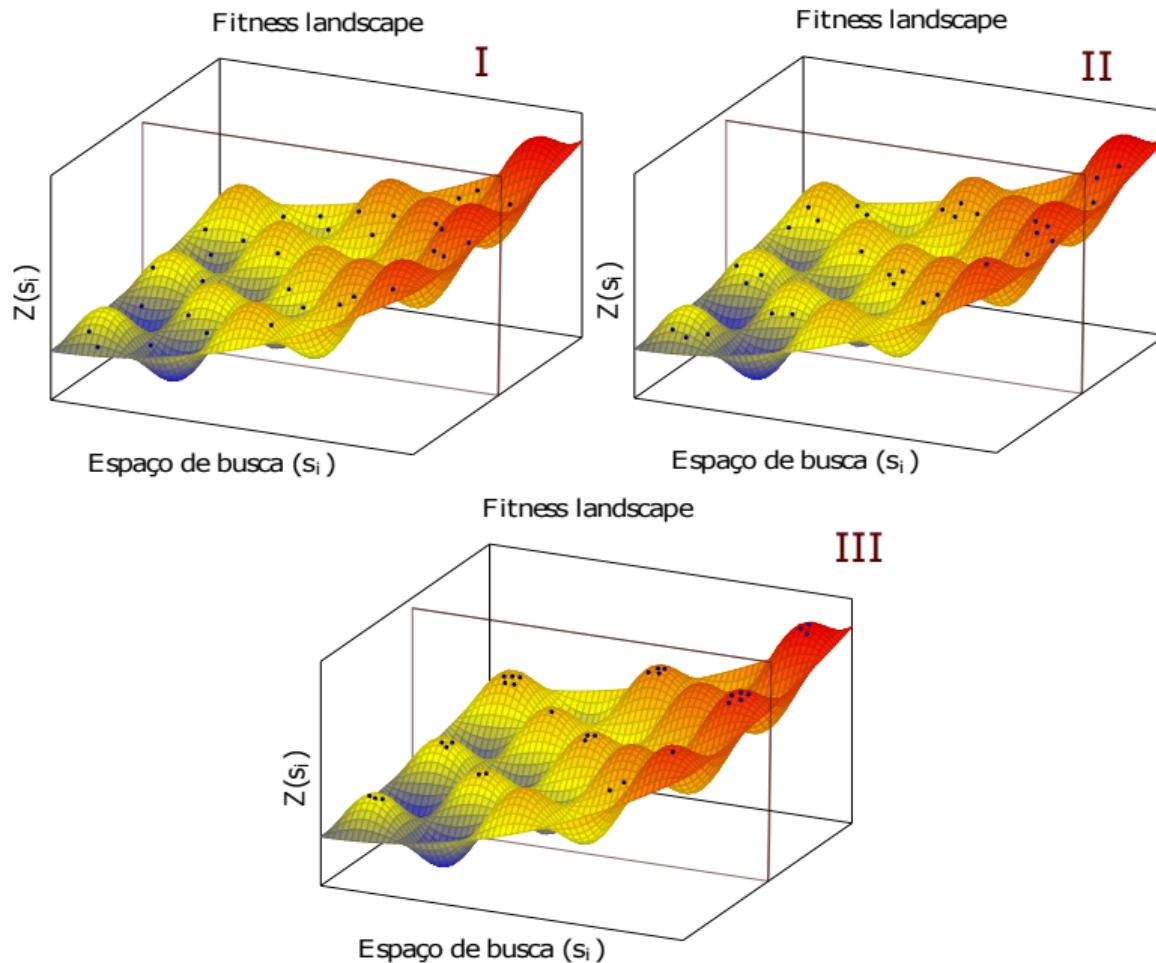
### ATENÇÃO

Não se esqueça que o *fitness landscape* é um conceito vinculado à **vizinhança**. É **errado** pensarmos em *fitness landscape* em algoritmos genéticos, pois não usamos vizinhança alguma! Vamos supor que o *fitness landscape* não seja ligado por um operador de movimento (vizinhança), mas sim como uma representação de todas as soluções possíveis para o problema (só para entendermos a convergência dos AG).

# Algoritmos populacionais

## No fitness landscape

Os algoritmos evolucionários (genéticos inclusos), começam com um população de indivíduos bem separados, e ao longo das iterações as soluções convergem para ótimos locais ou globais (ot. local/global também está ligado ao conceito de vizinhança...).



# Algoritmos genéticos

## Pseudocódigo

Abaixo é mostrado um pseudocódigo de algoritmos genéticos e evolucionários.

---

### Algorithm 2 Template de algoritmo evolucionário

---

```
 $P_0 = \text{Gere-Pop}()$                                 ▷ Criação da pop. inicial
 $t = 0$ 
while Critério de parada não atingido do
    Avaliar( $P_t$ )                                         ▷ Calcular o fitness
    Selecao( $P_t$ )                                         ▷ Selecionar indivíduos para reprod.
     $P'_t = \text{Reproducao}(P_t)$ 
     $P_{t+1} = \text{Substituicao}(P_t, P'_t)$                   ▷ Elimina indivíduos
     $t = t + 1$ 
end while
return Melhor individuo da População.
```

---

# Algoritmos genéticos

## Componentes de design

Dessa forma, temos que os principais componentes no design de um algoritmo genético são os seguintes:

1. **Geração da população inicial:** Como a primeira geração de soluções será criada?
2. **Avaliação:** Como será calculado o *fitness* de uma solução?
3. **Seleção:** Qual é o critério usado para selecionar os indivíduos que vão se reproduzir?
4. **Reprodução/mutação:** Como combinar duas (ou mais) soluções para gerar novas?
5. **Substituição:** Qual critério utilizar para remover soluções da população?

# Algoritmos genéticos

## Geração da população inicial

Dessa forma, temos que os principais componentes no design de um algoritmo genético são os seguintes:

1. **Geração da população inicial:** Como a primeira geração de soluções será criada?
2. **Avaliação:** Como será calculado o *fitness* de uma solução?
3. **Seleção:** Qual é o critério usado para selecionar os indivíduos que vão se reproduzir?
4. **Reprodução/mutação:** Como combinar duas (ou mais) soluções para gerar novas?
5. **Substituição:** Qual critério utilizar para remover soluções da população?

# Algoritmos genéticos

## Geração da população inicial

Devido a grande variabilidade que pode existir na população inicial, algoritmos populacionais focam mais na **exploração** do que na **intensificação** do *fitness landscape*.

O aspecto mais importante na definição da pop. inicial é a **diversificação**. Se a população é muito próxima (ou seja, são soluções parecidas), uma convergência prematura pode ocorrer.

Algumas formas de inicialização são mostradas abaixo:

# Algoritmos genéticos

## Geração da população inicial

Método	<i>Design</i>
Início aleatório	Os indivíduos são gerados aleatoriamente
Diversificação sequencial	Cria-se soluções de forma sequencial, de forma que a solução $i + 1$ está sempre a uma distância $\delta$ da solução $i$ .
Heuristica	Usa-se uma heurística que possibilite gerar diversas soluções, por exemplo a heurística do vizinho mais próximo do TSP com diversos pontos iniciais.
Aleatória/Heurística	Uma parte das soluções é gerada de forma aleatória e outra de forma heurística.

# Algoritmos genéticos

## Avaliação

Dessa forma, temos que os principais componentes no design de um algoritmo genético são os seguintes:

1. **Geração da população inicial:** Como a primeira geração de soluções será criada?
2. **Avaliação:** Como será calculado o *fitness* de uma solução?
3. **Seleção:** Qual é o critério usado para selecionar os indivíduos que vão se reproduzir?
4. **Reprodução/mutação:** Como combinar duas (ou mais) soluções para gerar novas?
5. **Substituição:** Qual critério utilizar para remover soluções da população?

# Algoritmos genéticos

## Avaliação

A avaliação da qualidade de uma solução em algoritmos genéticos é usualmente feita pela *função fitness*. Essa medida deve ser do tipo "maior melhor", considerando os métodos de seleção de indivíduos.

**A função *fitness* não precisa necessariamente ser igual ao valor da função objetivo!**

Por exemplo, sabemos que uma população diversificada é importante, de forma que é possível imbutir um "bonus" no *fitness* para as soluções que são mais diferentes da população, dessa forma mesmo que elas tenham uma *fitness* ruim, esse aspecto pode ser compensado por outras características gerando um *fitness* mais alto.

# Algoritmos genéticos

## Avaliação

Ainda, se ela precisa ser do tipo "maior melhor", como gerar o fitness de soluções de problemas de minimização (em que a fo menor é melhor)?

Nesses casos podemos transformar a fo da seguinte forma:

1. Primeiro calcular  $b_i = \frac{1}{z(x_i)}$  (o inverso da fo de toda solução).
2. Em seguida padronizar todos os valores dividindo pela soma:  $f_i = \frac{b_i}{\sum_0^n b_i}$

# Algoritmos genéticos

## Seleção

Dessa forma, temos que os principais componentes no design de um algoritmo genético são os seguintes:

1. **Geração da população inicial:** Como a primeira geração de soluções será criada?
2. **Avaliação:** Como será calculado o *fitness* de uma solução?
3. **Seleção:** Qual é o critério usado para selecionar os indivíduos que vão se reproduzir?
4. **Reprodução/mutação:** Como combinar duas (ou mais) soluções para gerar novas?
5. **Substituição:** Qual critério utilizar para remover soluções da população?

# Algoritmos genéticos

## Seleção

A etapa de seleção visa escolher quais indivíduos da população serão usados como pais, tendo seus materiais genéticos (partes da solução) mescladas de forma a gerar novos indivíduos. A idéia da seleção é que indivíduos com um alto *fitness* possuem componentes em sua estrutura que melhoram a função objetivo, e talvez mesclando esses indivíduos combine seus pontos fortes, criando soluções melhores.

# Algoritmos genéticos

## Seleção

A etapa de seleção visa escolher quais indivíduos da população serão usados como pais, tendo seus materiais genéticos (partes da solução) mescladas de forma a gerar novos indivíduos. A idéia da seleção é que indivíduos com um alto *fitness* possuem componentes em sua estrutura que melhoram a função objetivo, e talvez mesclando esses indivíduos combine seus pontos fortes, criando soluções melhores.

**PERGUNTA:** Se queremos mesclar "bons" indivíduos, por que simplesmente não selecionamos aqueles com os maiores valores de *fitness* para a reprodução?

# Algoritmos genéticos

## Seleção

A etapa de seleção visa escolher quais indivíduos da população serão usados como pais, tendo seus materiais genéticos (partes da solução) mescladas de forma a gerar novos indivíduos. A idéia da seleção é que indivíduos com um alto *fitness* possuem componentes em sua estrutura que melhoram a função objetivo, e talvez mesclando esses indivíduos combine seus pontos fortes, criando soluções melhores.

**PERGUNTA:** Se queremos mesclar "bons" indivíduos, por que simplesmente não selecionamos aqueles com os maiores valores de *fitness* para a reprodução?

**RESPOSTA:** Essa estratégia pode levar a uma convergência prematura para ótimos locais, e ainda existe o fator de diversificação: pode ser que embora uma solução tenha um *fitness* ruim, uma pequena parte dela faz parte do ótimo!

# Algoritmos genéticos

## Seleção

### Conclusão

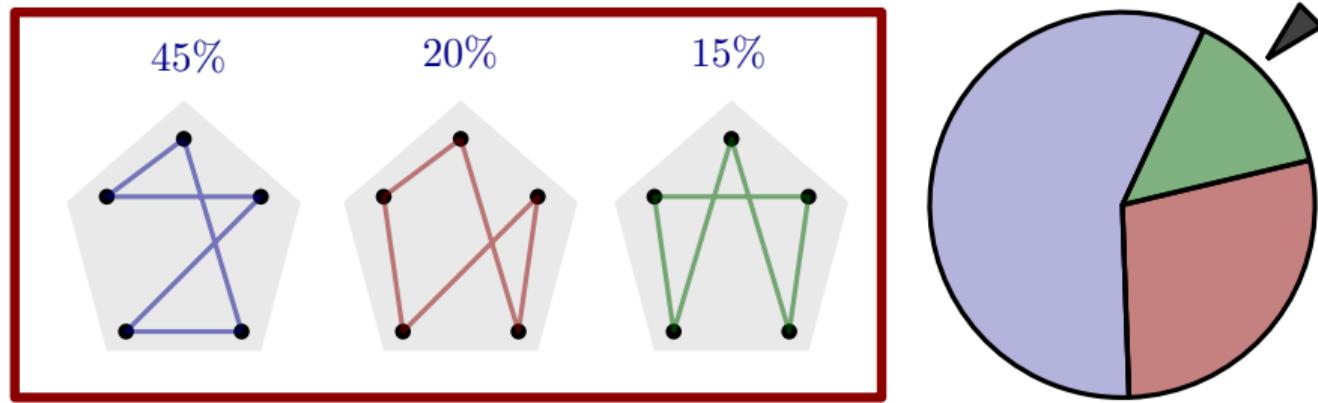
Embora soluções com baixo *fitness* não sejam boas, é uma boa prática manter elas na lista de possíveis seleções, porém com uma **baixa probabilidade**. Assim, os métodos mais comuns de seleção trabalham com essa ideia: todos podem ser selecionados, porém com probabilidades diferentes, e que refletem o seu *fitness* (quanto maior o *fitness* maior a chance de ser escolhido para reprodução).

Os três métodos mais comuns para seleção são:

1. Roleta (*Roulette Wheel Selection*)
2. Amostragem Estocástica Universal (*Stochastic Universal Sampling*)
3. Torneio (*Tournament selection*)

# Algoritmos genéticos

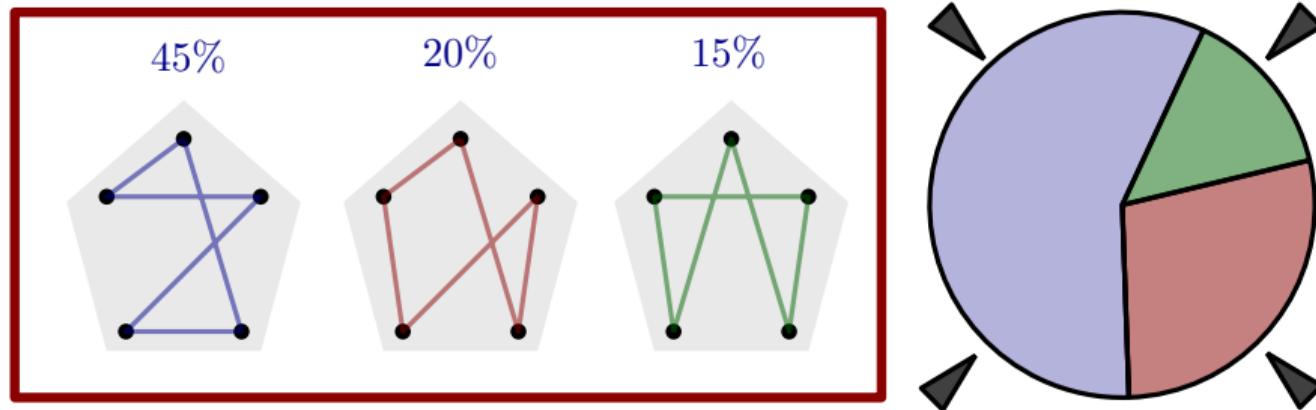
## Seleção - Roleta



No **método da roleta** atribuímos uma probabilidade de seleção a cada indivíduo que é proporcional ao seu *fitness* como em um gráfico de pizza. Em seguida atribuímos um marcador na roleta e a "rolamos", o indivíduo selecionado pelo marcador é usado na reprodução.

# Algoritmos genéticos

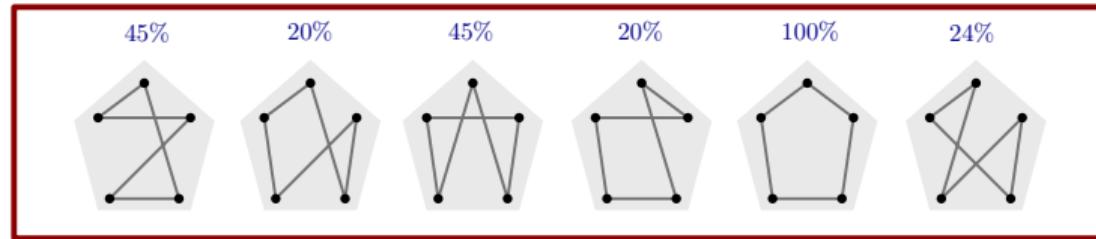
## Seleção - Amostragem Estocástica Universal



Já no método da **Amostragem Estocástica Universal**, a roleta é **girada uma única vez** para todos os indivíduos, porém existe um número  $n$  de marcadores equidistantes na roleta, **um para cada indivíduo** que irá participar da reprodução.

# Algoritmos genéticos

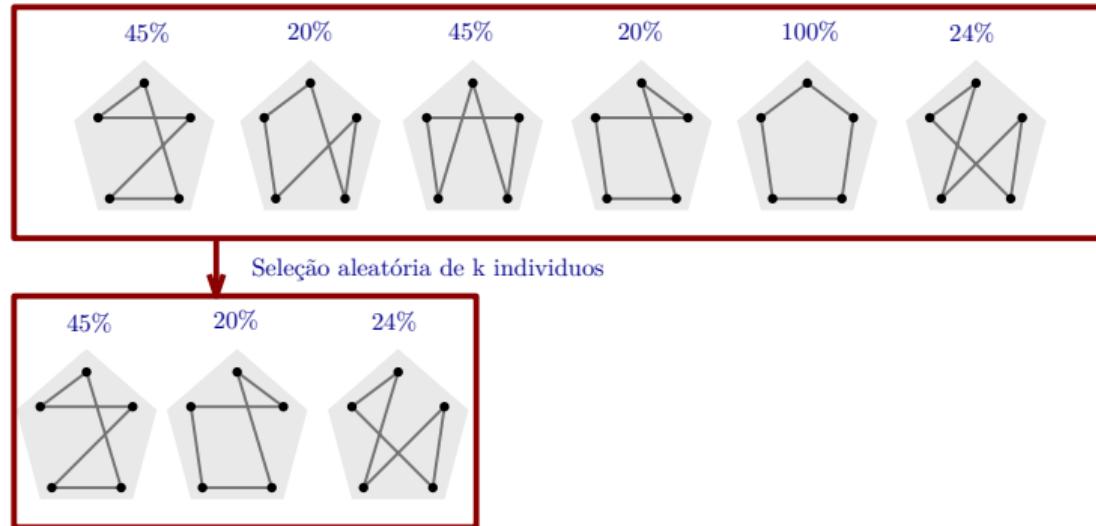
## Seleção - Amostragem Estocástica Universal



No método do **Torneio**, começamos com todos os indivíduos da população

# Algoritmos genéticos

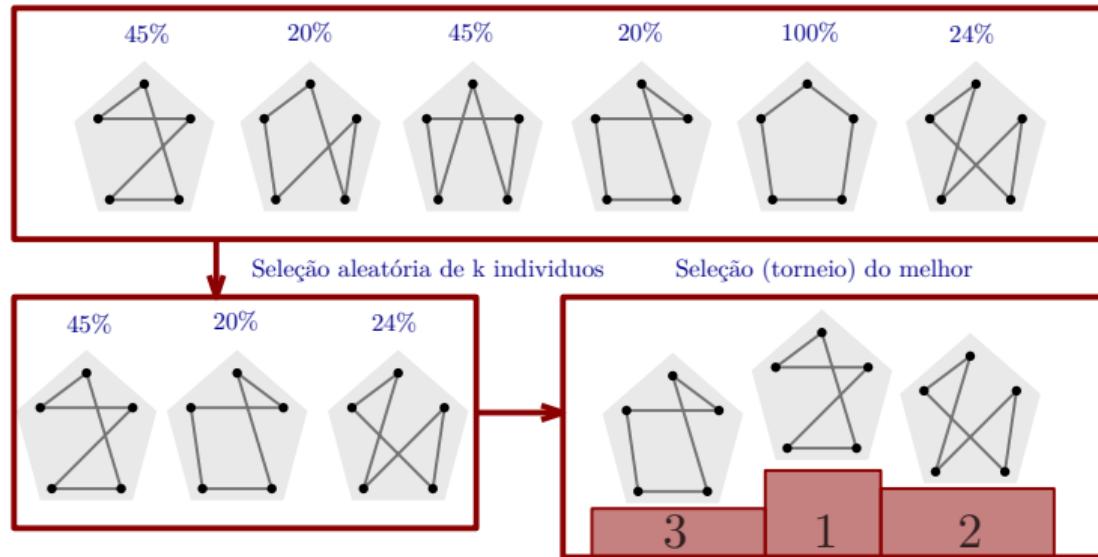
## Seleção - Amostragem Estocástica Universal



Em seguida  $k$  indivíduos são selecionados aleatoriamente dentre a população (cada um com a mesma probabilidade).

# Algoritmos genéticos

## Seleção - Amostragem Estocástica Universal



Dentre esses indivíduos o torneio é realizado: o indivíduo com maior *fitness* é selecionado para reproduzir. O método é executado uma vez para cada indivíduo que for reproduzir.

# Algoritmos genéticos

## Seleção - Reprodução/Mutação

Dessa forma, temos que os principais componentes no design de um algoritmo genético são os seguintes:

1. **Geração da população inicial:** Como a primeira geração de soluções será criada?
2. **Avaliação:** Como será calculado o *fitness* de uma solução?
3. **Seleção:** Qual é o critério usado para selecionar os indivíduos que vão se reproduzir?
4. **Reprodução/mutação:** Como combinar duas (ou mais) soluções para gerar novas?
5. **Substituição:** Qual critério utilizar para remover soluções da população?

# Algoritmos genéticos

## Seleção - Reprodução/Mutação

A etapa da reprodução tem por objetivo **mesclar boas partes das soluções dos pais e transmitir para os filhos**. Essa etapa é muito dependente das estruturas de dados usadas para a representação das soluções, factibilidade, etc... (como as vizinhanças). De forma que cada problema possui métodos específicos para reprodução.

Veremos 2 métodos muito utilizados para a reprodução de soluções do TSP, são elas:

1. Crossover Parcialmente Mapeado (*Partially Mapped Crossover* - PMX)
2. Crossover Ordenado (*Ordered Crossover* - OX)

$p_1$ 

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

 $p_2$ 

4	5	2	1	8	7	6	9	3
---	---	---	---	---	---	---	---	---

No **PMX** a combinação é feita com 2 pais (*parents*  $p_1$  e  $p_2$ ) gerando 2 filhos (*offsprings*  $o_1$  e  $o_2$ ).

$p_1$ 

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

 $p_2$ 

4	5	2	1	8	7	6	9	3
---	---	---	---	---	---	---	---	---

 $o_1$ 

			1	8	7	6		
--	--	--	---	---	---	---	--	--

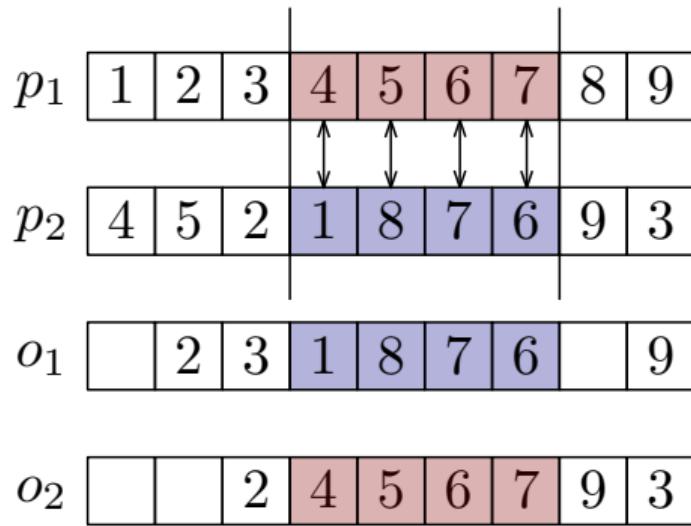
 $o_2$ 

			4	5	6	7		
--	--	--	---	---	---	---	--	--

Dois pontos de corte aleatório são determinados para ambos os pais. Em seguida, os elementos entre os pontos de  $p_1$  são alocados em  $o_2$  e os elementos entre os pontos de  $p_2$  são alocados em  $o_1$ .

$p_1$	1	2	3	4	5	6	7	8	9
$p_2$	4	5	2	1	8	7	6	9	3
$o_1$		2	3	1	8	7	6		9
$o_2$			2	4	5	6	7	9	3

Para todos os elementos restantes (tanto de  $o_1$  quanto de  $o_2$ ), segue-se a ordem dos pais que não geraram seus interiores, alocando nas mesmas posições os valores que ainda não foram alocados. Os outros ficam ainda vagos.



Para alocar os elementos faltantes, um [mapeamento](#) é feito entre os elementos internos aos pontos dos pais.

$p_1$ 

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

 $p_2$ 

4	5	2	1	8	7	6	9	3
---	---	---	---	---	---	---	---	---

 $o_1$ 

4	2	3	1	8	7	6	5	9
---	---	---	---	---	---	---	---	---

 $o_2$ 

1	8	2	4	5	6	7	9	3
---	---	---	---	---	---	---	---	---

Esse mapeamento é usado no preenchimento dos pontos faltantes.

$p_1$ 

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

 $p_2$ 

4	5	2	1	8	7	6	9	3
---	---	---	---	---	---	---	---	---

 $o_1$ 

			1	8	7	6		
--	--	--	---	---	---	---	--	--

 $o_2$ 

			4	5	6	7		
--	--	--	---	---	---	---	--	--

O **método OX** começa da mesma forma do PMX, selecionando pontos de corte e copiando os elementos internos aos 2 filhos.

$p_1$	1	2	3	4	5	6	7	8	9
$p_2$	4	5	2	1	8	7	6	9	3
$o_1$				1	8	7	6		
$o_2$	2	1	8	4	5	6	7	9	3

Para o preenchimento dos elementos faltantes, segue-se a ordem dos pais (não o que gerou a parte interna), não adicionando os elementos já adicionados, porém continuando na busca pelos que ainda não foram.

$p_1$	<table border="1"> <tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr> </table>	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9		
										
$p_2$	<table border="1"> <tr><td>4</td><td>5</td><td>2</td><td>1</td><td>8</td><td>7</td><td>6</td><td>9</td><td>3</td></tr> </table>	4	5	2	1	8	7	6	9	3
4	5	2	1	8	7	6	9	3		
$o_1$	<table border="1"> <tr><td>3</td><td>4</td><td>5</td><td>1</td><td>8</td><td>7</td><td>6</td><td>9</td><td>2</td></tr> </table>	3	4	5	1	8	7	6	9	2
3	4	5	1	8	7	6	9	2		
$o_2$	<table border="1"> <tr><td>2</td><td>1</td><td>8</td><td>4</td><td>5</td><td>6</td><td>7</td><td>9</td><td>3</td></tr> </table>	2	1	8	4	5	6	7	9	3
2	1	8	4	5	6	7	9	3		

Essa busca continua "dando a volta" na solução, até que todos os elementos do filho estejam preenchidos.

# Algoritmos genéticos

## Reprodução/Mutação

A mutação é um **operador unário** (afeta uma solução), e tem por objetivo realizar **pequenas alterações no indivíduo**. Conforme as gerações vão passando, a tendência é que os indivíduos fiquem mais parecidos devido à convergência. A mutação tem o objetivo de dar uma pequena pivotada nesse processo, para **evitar uma convergência prematura**, e ainda aumentar o fator de exploração.

Como a reprodução, a mutação é um **processo específico para cada problema**, porém é padrão que deve existir uma probabilidade de mutação  $p_m$  para cada indivíduo gerado, estudo sugerem que essa probabilidade deve estar no intervalo  $p_m \in [0.001, 0.01]$  (obviamente isso não é uma regra.)

É possível usar as próprias estruturas de vizinhança do problema para gerar a mutação. Para o TSP, por exemplo, podemos fazer o *swap* de  $k$  elementos de uma rota, com  $k << n$ .

# Algoritmos genéticos

## Substituição

Dessa forma, temos que os principais componentes no design de um algoritmo genético são os seguintes:

1. **Geração da população inicial:** Como a primeira geração de soluções será criada?
2. **Avaliação:** Como será calculado o *fitness* de uma solução?
3. **Seleção:** Qual é o critério usado para selecionar os indivíduos que vão se reproduzir?
4. **Reprodução/mutação:** Como combinar duas (ou mais) soluções para gerar novas?
5. **Substituição:** Qual critério utilizar para remover soluções da população?

# Algoritmos genéticos

## Substituição

Após a geração dos novos indivíduos, a população é aumentada. Ao longo do algoritmo a mesma deve ter um tamanho fixo, de forma que alguns indivíduos devem ser selecionados para serem removidos. Essa etapa é chamada **substituição**. Novamente podemos nos perguntar:

# Algoritmos genéticos

## Substituição

Após a geração dos novos indivíduos, a população é aumentada. Ao longo do algoritmo a mesma deve ter um tamanho fixo, de forma que alguns indivíduos devem ser selecionados para serem removidos. Essa etapa é chamada **substituição**. Novamente podemos nos perguntar:

**PERGUNTA:** Se queremos manter "bons" indivíduos, por que não eliminar os  $k$  piores?

# Algoritmos genéticos

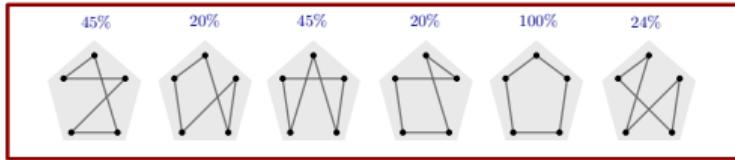
## Substituição

Após a geração dos novos indivíduos, a população é aumentada. Ao longo do algoritmo a mesma deve ter um tamanho fixo, de forma que alguns indivíduos devem ser selecionados para serem removidos. Essa etapa é chamada **substituição**. Novamente podemos nos perguntar:

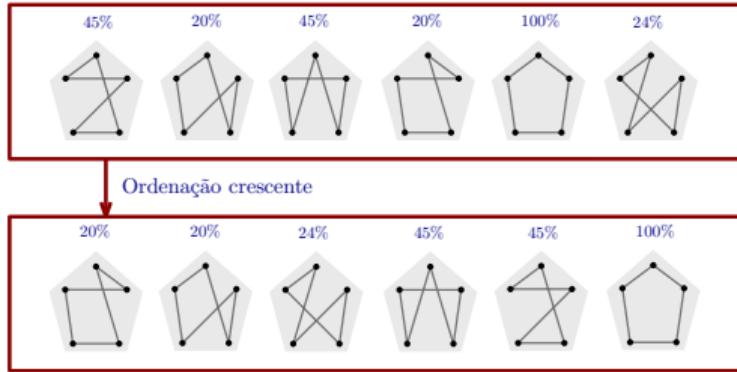
**PERGUNTA:** Se queremos manter "bons" indivíduos, por que não eliminar os k piores?

E a resposta é a mesma...

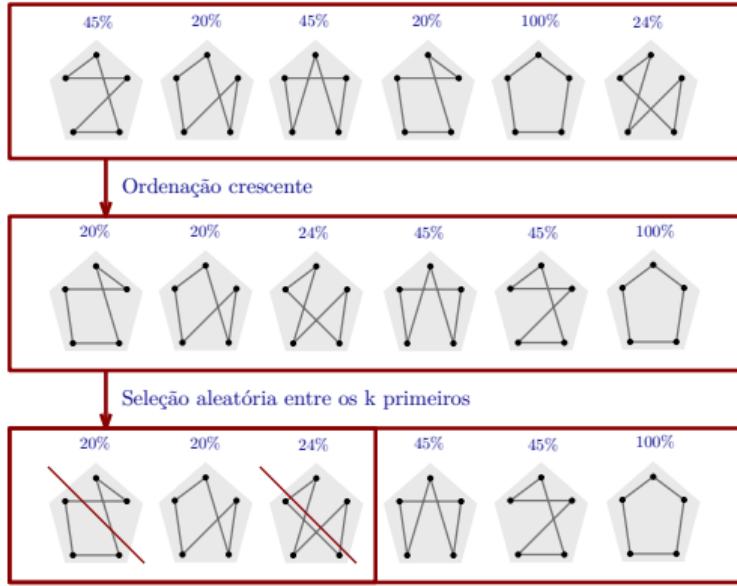
**RESPOSTA:** Essa estratégia pode levar a uma convergência prematura para ótimos locais, e ainda existe o fator de diversificação: pode ser que embora uma solução tenha um *fitness* ruim, uma pequena parte dela faz parte do ótimo!



Uma forma de resolver esse problema é dando uma **margem** para selecionarmos indivíduos melhores para serem eliminados



Basta ordenarmos os elementos de forma crescente em relação ao seu *fitness*.



Em seguida **selecionamos aleatoriamente** o número que deve ser eliminado  $m$  dentre os  $k$  primeiros elementos, com  $m < k$ . No exemplo ao lado temos  $m = 2$  e  $k = 3$ .

# AG e Estruturas de Dados

## AG e Estruturas de Dados

Lembrando da aula de representação de soluções: existe algum tipo de estrutura de dados e/ ou representação de soluções que facilita a implementação de algoritmos genéticos?

# AG e Estruturas de Dados

Lembrando da aula de representação de soluções: existe algum tipo de estrutura de dados e/ ou representação de soluções que facilita a implementação de algoritmos genéticos?

## Genetic Algorithms and Random Keys for Sequencing and Optimization

JAMES C. BEAN / Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan 48109-2117; Email: James.Bean@umich.edu

(Received: June 1992; revised: October 1993; accepted: January 1994)

---

In this paper we present a general genetic algorithm to address a wide variety of sequencing and optimization problems including multiple machine scheduling, resource allocation, and the quadratic assignment problem. When addressing such problems, genetic algorithms typically have difficulty maintaining feasibility from parent to offspring. This is overcome with a robust representation technique called random keys. Computational results are shown for multiple machine scheduling, resource allocation, and quadratic assignment problems.

solution as an offspring. See [12] for details on genetic algorithms.

Many authors have developed problem-specific representations of solutions that overcome the offspring feasibility difficulty. Some of these include PMX crossover,<sup>[13]</sup> the subsequence-swap operator,<sup>[15]</sup> the subsequence-chunk operator,<sup>[16]</sup> other subsequence operators,<sup>[7]</sup> edge recombination,<sup>[33]</sup> the ARGOT strategy,<sup>[28]</sup> and forcing.<sup>[25]</sup> Most of

Lembrando da aula de representação de soluções: existe algum tipo de estrutura de dados e/ ou representação de soluções que facilita a implementação de algoritmos genéticos?

## Genetic Algorithms and Random Keys for Sequencing and Optimization

JAMES C. BEAN / Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan  
48109-2117; Email: James.Bean@umich.edu

(Received: June 1992; revised: October 1993; accepted: January 1994)

In this paper we present a general genetic algorithm to address a wide variety of sequencing and optimization problems including multiple machine scheduling, resource allocation, and the quadratic assignment problem. When addressing such problems, genetic algorithms typically have difficulty maintaining feasibility from parent to offspring. This is overcome with a robust representation technique called random keys. Computational results are shown for multiple machine scheduling, resource allocation, and quadratic assignment problems.

"Ao lidar com esses problemas, os algoritmos genéticos tipicamente têm dificuldade em manter a viabilidade de pais para filhos."

algorithms.

Many authors have developed problem-specific representations of solutions that overcome the offspring feasibility difficulty. Some of these include PMX crossover,<sup>[13]</sup> the subsequence-swap operator,<sup>[15]</sup> the subsequence-chunk operator,<sup>[16]</sup> other subsequence operators,<sup>[7]</sup> edge recombination,<sup>[33]</sup> the ARGOT strategy,<sup>[28]</sup> and forcing.<sup>[25]</sup> Most of

As **chaves aleatórias** foram criadas com o intuito de serem utilizadas em algoritmos genéticos! Usando-as, todas as operações de cruzamentos são executadas **no espaço indireto**. Só retomamos a solução quando a decodificamos - ou seja, já fazemos a decodificação para a factibilidade.

$p_1$ 

0.41	0.70	0.24	0.89	0.63	0.74	0.12	0.23	0.66
------	------	------	------	------	------	------	------	------

 $p_2$ 

0.97	0.44	0.07	0.70	0.28	0.54	0.08	0.98	0.40
------	------	------	------	------	------	------	------	------

 $o_1$ 

--	--	--	--	--	--	--	--	--

 $o_2$ 

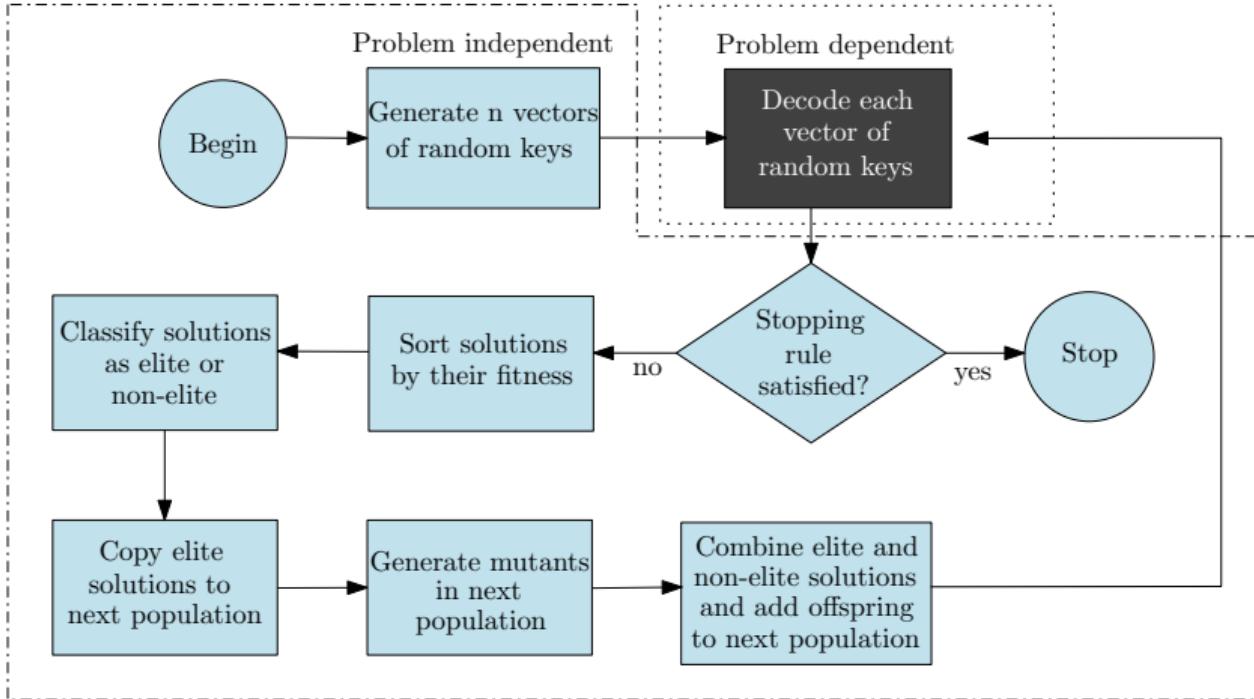
--	--	--	--	--	--	--	--	--

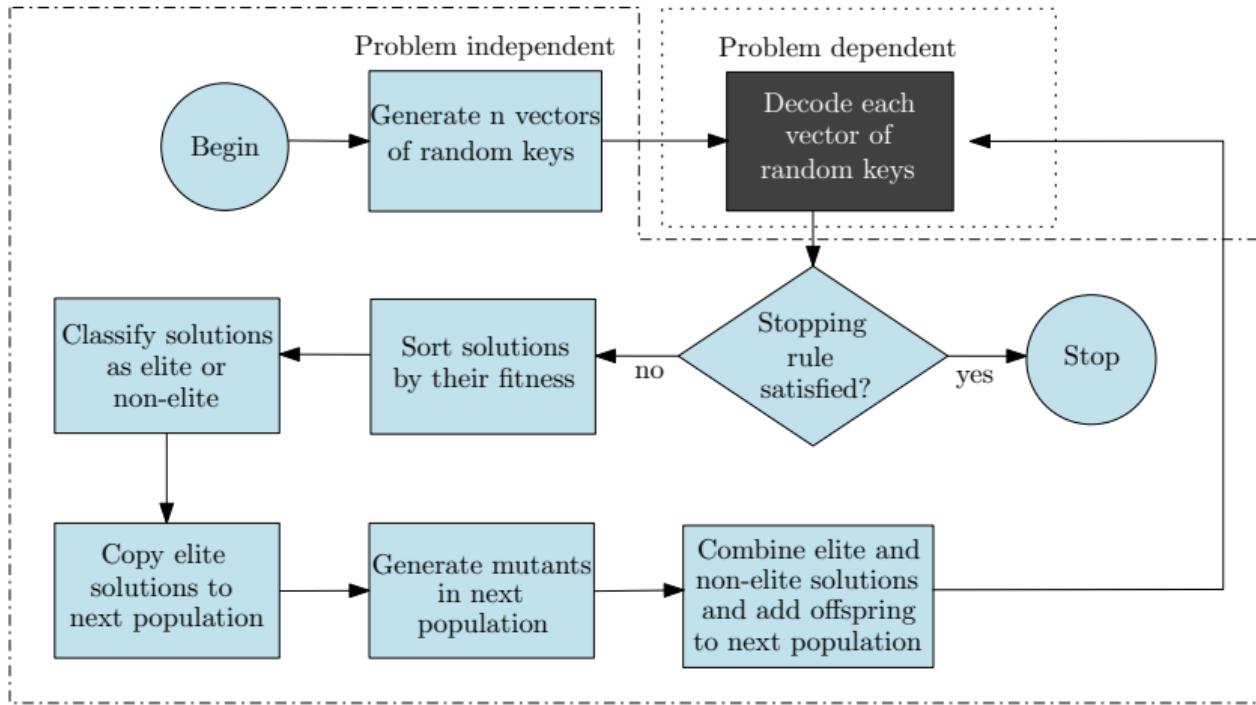
Podemos executar o mesmo PMX que anteriormente.

$p_1$	0.41	0.70	0.24	0.89	0.63	0.74	0.12	0.23	0.66
$p_2$	0.97	0.44	0.07	0.70	0.28	0.54	0.08	0.98	0.40
$O_1$	0.41	0.70	0.24	0.70	0.28	0.54	0.08	0.23	0.66
$O_2$	0.97	0.44	0.07	0.89	0.63	0.74	0.12	0.98	0.40

No entanto, nenhuma correção é necessária agora! Podemos simplesmente realizar a troca de materiais genéticos e em seguida **fazer a decodificação** dos filhos gerados.

Note que com as **random keys**, o algoritmo genético se torna **independente do problema a ser resolvido** - todo o algoritmo é aplicado no espaço das chaves aleatórias. A única parcela que depende do problema é a decodificação. Vejamos o fluxograma de um algoritmo genético que usa RK.





A única parte que é específica do problema é a **decodificação da solução**.

## Conclusão

Ou seja, se você conseguir pensar em uma codificação por random keys para o seu problema, é possível usar pacotes de algoritmos genéticos prontos, em que a única coisa que é necessário implementar é o decodificador, para extrair o custo de uma chave aleatória. Uma versão desses AGs que está sendo muito usada é o BRKGA (**B**iased **R**andom **K**eys **G**enetic **A**lgorithms) - com códigos prontos em Python e C++.

# Pacotes BRKGA

## Pacotes prontos Python e C++ BRKGA:



The screenshot shows the GitHub page for the BRKGA-MP-IPR repository. The page title is "BRKGA-MP-IPR - C++ version 3.1.1". The repository features a logo composed of red, blue, and green wavy bands forming a stylized 'S' shape, with the text "BRKGA", "MP", and "IPR" stacked vertically to its right. Below the logo, the text "BRKGA-MP-IPR - C++ version 3.1.1" is displayed. At the bottom of the page, there are links for Documentation, Tutorial API, License, License BSD-3-Clause, and License MIT.


The screenshot shows the GitHub page for the BRKGA-MP-IPR repository. The page title is "BRKGA-MP-IPR - Python version". The repository features the same logo as the C++ version, consisting of red, blue, and green wavy bands forming a stylized 'S' shape, with the text "BRKGA", "MP", and "IPR" stacked vertically to its right. The text "BRKGA-MP-IPR - Python version" is displayed below the logo.

# Artigo BRKGA

## Artigo de revisão sobre BRKGA:



Invited review

### Biased random-key genetic algorithms: A review

Mariana A. Londe <sup>a</sup>, Luciana S. Pessoa <sup>a</sup>, Carlos E. Andrade <sup>b</sup>, Mauricio G.C. Resende <sup>c,\*</sup>

<sup>a</sup> Department of Industrial Engineering, PUC-Rio, Rua Marquês de São Vicente, 225, Gávea 22453-900 Rio de Janeiro, RJ, Brazil

<sup>b</sup> AT&T Labs Research, 200 South Laurel Avenue, Middletown, NJ 07748, USA

<sup>c</sup> Industrial & Systems Engineering, University of Washington, 3900 E Stevens Way NE, Seattle, WA 98195, USA

---

#### ARTICLE INFO

Keywords:  
Biased random-key genetic algorithms  
Literature review  
Metaheuristics  
Applications

---

#### ABSTRACT

This paper is a comprehensive literature review of Biased Random-Key Genetic Algorithms (BRKGA). BRKGA is a metaheuristic that employs random-key-based chromosomes with biased, uniform, and elitist mating strategies in a genetic algorithm framework. The review encompasses over 150 papers with a wide range of applications, including classical combinatorial optimization problems, real-world industrial use cases, and non-orthodox applications such as neural network hyperparameter tuning in machine learning. Scheduling is by far the most prevalent application area in this review, followed by network design and location problems. The most frequent hybridization method employed is local search, and new features aim to increase population diversity. We also detail challenges and future directions for this method. Overall, this survey provides a comprehensive overview of the BRKGA metaheuristic and its applications and highlights important areas for future research.

## Exemplo - TSP

## Exemplo - TSP

1. **EXEMPLO:** Implemente um AG para o problema do TSP com as seguintes características:
  - 1.1 **Geração da população inicial:** aleatória.
  - 1.2 **Seleção:** Roleta
  - 1.3 **Cruzamento:** OX (Order Crossover)
  - 1.4 **Mutação:** swap aleatório

## Atividade 1

Considerando o problema que você escolheu para tratar na disciplina.

1. Encontre um artigo que o resolva usando AG.
2. Descreva os como os autores resolvem os principais componentes do design para o algoritmo.