# Taxonomia dos Algoritmos Evolutivos

# Parte I







## 1 Introdução

- Um algoritmo evolutivo pode ser entendido como um procedimento iterativo de busca (otimização) inspirado nos mecanismos evolutivos biológicos, sendo assim modelos simplificados de processos biológicos.
- Baseado na teoria neo-Darwiniana da evolução, é possível propor um *algoritmo evolutivo básico* ou *padrão* com as seguintes características:
  - *Uma população de candidatos à solução (denominados indivíduos ou cromossomos) que se reproduz com herança genética*: cada indivíduo corresponde a uma estrutura de dados que representa ou codifica um ponto em um espaço de busca. Estes indivíduos devem se reproduzir (de forma sexuada ou assexuada), gerando descendentes que herdam algumas das características de seu(s) progenitor(es). No caso de reprodução sexuada, há troca de material genético (crossover ou recombinação) entre dois ou mais indivíduos progenitores;

- *Variação genética*: durante o processo reprodutivo os descendentes não apenas herdam características de seu(s) progenitor(es), como eles também podem sofrer uma mutação genética responsável pela alteração de seu código genético;
- *Seleção natural*: a avaliação dos indivíduos em seu ambiente – através de uma função de avaliação ou fitness – resulta em um valor correspondente à adaptabilidade ou qualidade deste indivíduo. A comparação dos valores individuais de fitness resultará em uma competição pela sobrevivência e reprodução no ambiente, devendo ser promovida uma vantagem seletiva daqueles indivíduos com valores elevados de fitness.

- Quando todos os passos acima (reprodução, variação genética e seleção) tiverem sido executados, diz-se que ocorreu uma *geração*.
- Em cada iteração, sejam $P$ uma população contendo $N$ indivíduos $P = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ (estruturas de dados), $f_i$, $i = 1,...N$, o valor do fitness de cada indivíduo da população, e $pc$ e $pm$ as probabilidades de crossover e mutação, respectivamente.





- Com isso, é possível propor um algoritmo evolutivo padrão da seguinte forma:

```
procedimento [P] = AE_padrao(N,pc,pm)
  P'' ← inicializa(N)
  fit ← avalia(P'')
  t ← 1
  enquanto NÃO condição_de_parada faça,
    P ← seleciona(P'',fit)
    P' ← reproduz(P,fit,pc)
    P'' ← varia(P',pm)
    fit ← avalia(P'')
    t ← t + 1
  fim enquanto
fim procedimento
```

- O critério de parada pode variar bastante, sendo os mais simples baseados em um número fixo de gerações ou na detecção de que se atingiu uma solução com desempenho acima de um limiar pré-estabelecido.

# 2 Os principais Algoritmos Evolutivos

- Tipos:
  - Algoritmos Genéticos (AG) – Genetic Algorithms (GA)
  - Estratégias Evolutivas (EE) – Evolution Strategies (ES)
  - Programação Evolutiva (PE) – Evolutionary Programming (EP)
  - Programação Genética (PG) – Genetic Programming (GP)
  - Sistemas Classificadores (SC) – Classifier Systems (CS)

- **Quais são as diferenças entre eles?**
  - Representação (codificação): qual estrutura de dados?
  - Operadores genéticos: crossover e/ou mutação?
  - Operadores de seleção: determinística ou probabilística?





## 2.1 Características Principais

- *Algoritmos Genéticos*: formalizados por HOLLAND (1975) com o nome de *planos adaptativos*, enfatizam a recombinação como o principal operador de busca e aplicam mutação com baixas probabilidades (operador secundário). Operam com representação binária de indivíduos e possuem seleção probabilística proporcional ao fitness (implementada com um procedimento denominado de Roulette Wheel).

- *Estratégias Evolutivas*: desenvolvidas por RECHENBERG (1973) e SCHWEFEL (1975, 1977), utilizam mutações com distribuição normal para modificar vetores de números em ponto flutuante e enfatizam a mutação e a recombinação como operadores essenciais ao processo de busca no espaço de busca e no espaço de parâmetros. O operador de seleção é determinístico e o tamanho da população de progenitores e de descendentes pode ser distinto.

- *Programação Evolutiva*: formalizada por FOGEL *et al.* (1966), enfatiza a mutação e não incorpora a recombinação. Assim como as EEs, quando aplicada a problemas de otimização de valores reais, a PE também emprega mutações com distribuição normal e estende o processo evolutivo ao espaço de parâmetros. O operador de seleção é probabilístico e a maioria das aplicações atuais emprega vetores de números em ponto flutuante como codificação, embora ela tenha sido originalmente desenvolvida para evoluir máquinas de estado finito.

- *Programação Genética*: KOZA (1992) estendeu os algoritmos genéticos para o espaço de programas computacionais. As estruturas de dados são representadas utilizando árvores, e os operadores de crossover e mutação (adaptados para operarem com estruturas do tipo árvore) são empregados. O processo de seleção segue aquele dos algoritmos genéticos, ou seja, a seleção é probabilística e proporcional ao fitness.





- Sistemas Classificadores: ver Tópico 9.
- Em resumo (Adaptado de BÄCK, 1994):

| | **AG** | **EE** | **PE** | **PG** |
|---|---|---|---|---|
| Representação | Cadeias binárias | Vetores de números em ponto flutuante | Vetores de números em ponto flutuante | Árvores |
| Auto-adaptação | Nenhuma | Desvio padrão e co-variâncias | Desvio padrão e coeficiente de co-relação | Nenhuma |
| O fitness é | Valor escalonado da função objetivo | Valor da função objetivo | Valor (escalonado) da função objetivo | Valor escalonado da função objetivo |
| Mutação | Operador secundário | Principal operador | Único operador | Um dos operadores |
| Recombinação | Principal operador | Diferentes variações, importante para a auto-adaptação | Nenhuma | Um dos operadores |
| Seleção | Probabilística | Determinística | Probabilística | Probabilística |

- É importante salientar que todas as características acima correspondem aos algoritmos originais (padrões). Entretanto, todos os algoritmos são passíveis de variações e hibridizações de representação e operadores.
- De fato, a fronteira entre todos eles está cada dia menos perceptível e a comunidade científica já prefere descrever seus algoritmos como um algoritmo evolutivo com características específicas ao invés de dizer que está trabalhando com um algoritmo particular.

## 3 Origens da Computação Evolutiva

- Anos 50 e 60: cientistas da computação já estudavam sistemas evolutivos com a ideia de que o mecanismo de evolução poderia ser utilizado como uma ferramenta de otimização para problemas de engenharia:
  - ✓ FRASER (1959) – "Simulation of Genetic Systems by Automatic Digital Computers" – Australian Journal of Biological Science, 10:484-499.





  - ✓ FRIEDBERG (1958) – "A Learning Machine: Part I" – IBM Journal, pp. 2-13.
  - ✓ ANDERSON (1953) – "Recent Advances in Finding Best Operating Conditions" – Journal of American Statistic Association, 48, pp. 789-798.
  - ✓ BREMERMANN (1962) – "Optimization Through Evolution and Recombination" – in M.C. Yovits, G.T. Jacobi and D.G. Goldstein, editors – *Self-organizing Systems*, Spartan, Washington, D.C., pp. 93-106.

- RECHENBERG (1973) introduziu, nos anos 60, as ***estratégias evolutivas***, as quais foram utilizadas para otimizar parâmetros de valor real em sistemas dinâmicos. As estratégias evolutivas foram posteriormente desenvolvidas por SCHWEFEL (1975; 1977).
- FOGEL, OWENS e WALSH (1966) desenvolveram a ***programação evolutiva***, método em que os candidatos à solução de um dado problema são representados por máquinas de estado finito, as quais evoluem pela mutação aleatória de seus

diagramas de transição de estados, seguida pela seleção da mais bem adaptada. Uma formulação mais ampla da programação evolutiva pode ser encontrada em FOGEL (1999).

- Os ***algoritmos genéticos*** foram concebidos por Holland nos anos 60, tendo sido desenvolvidos até meados dos anos 70 por seu grupo de pesquisa na Universidade de Michigan. Em contraste com as estratégias evolutivas e a programação evolutiva, o objetivo original de Holland não foi o de desenvolver algoritmos para a solução de problemas específicos, mas sim estudar formalmente os fenômenos de adaptação, naturais ou artificiais, com o propósito de importar estes mecanismos de adaptação para ambientes computacionais. HOLLAND (1975/1992) apresentou os algoritmos genéticos como uma abstração da evolução biológica, tendo como inovações significativas a utilização conjunta de operadores de recombinação e inversão (além de operadores de mutação) e de um número elevado de indivíduos em cada geração.





- HOLLAND (1975/1992) também apresentou os ***sistemas classificadores***, especificamente implementados para operarem em ambientes não-estacionários, como requerido no caso de algumas aplicações de agentes autônomos.
- Já no início dos anos 90, KOZA (1992) estendeu as técnicas de algoritmo genético para o espaço de programas computacionais, resultando na ***programação genética***.
  - ✓ Em programação genética, os indivíduos que constituem a população sujeita ao processo evolutivo, ao invés de apresentarem cadeias cromossômicas de comprimento fixo, são na verdade programas que, quando executados, representam candidatos à solução do problema. A programação genética representa uma iniciativa de se desenvolver métodos suficientemente complexos e robustos para a geração automática de programas computacionais genéricos.

- Os seguintes eventos foram importantes para o estabelecimento da área de computação evolutiva.
  - ✓ Encontro de pesquisadores em algoritmos evolutivos na conferência *Parallel Problem Solving from Nature* (PPSN), realizada em Dortmund em 1990.
  - ✓ Organização da primeira *International Conference on Genetic Algorithms* (ICGA'91) em 1991.
  - ✓ Chegada em um consenso para o nome da área – *computação evolutiva* – e o estabelecimento de um jornal homônimo pelo MIT Press em 1993.
  - ✓ Inclusão da área na primeira *World Conference on Computational Intelligence* (WCCI) em 1994, que incluiu: redes neurais artificiais, sistemas nebulosos e algoritmos evolutivos.





## 4 Algumas Possíveis Aplicações da Computação Evolutiva

- Aplicações de computação evolutiva podem ser encontradas em diversas áreas. Por conveniência, elas serão divididas aqui em cinco grandes áreas não exaustivas e não mutuamente exclusivas (BÄCK, 1994):
  - ✓ Planejamento
  - ✓ Projeto (*Design*)
  - ✓ Identificação e Controle
  - ✓ Classificação

### 4.1 Aplicações em Planejamento

- *Roteamento*:
  - o *Caixeiro viajante*: qual é a melhor ordem de cidades a serem visitadas, dado o custo entre cidades e considerando o retorno à primeira cidade visitada?
  - o *Roteamento de veículos*: generalização do caixeiro viajante para mais de um veículo (caixeiro).

- *Robótica*: um caminho factível, seguro e sem colisões deve ser percorrido por um robô.

- *Sequenciamento de tarefas* (*scheduling*): desenvolver um plano para executar uma determinada quantidade de tarefas em um período de tempo, onde os recursos são limitados, existem restrições e pode haver mais do que um objetivo a ser otimizado.
  - *Job shop scheduling*: um número finito de processos (jobs) deve ser processado por um número finite de máquinas. Cada processo consiste de uma sequência pré-determinada de tarefas (tasks), cada qual requerendo uma dada máquina por um certo interval de tempo, sem interrupção. As tarefas do mesmo processo não podem ser executadas concorrentemente e cada processo deve utilizar cada máquina exatamente uma vez. Um sequenciamento factível é uma atribuição de tarefas em janelas de tempo (time slots) das máquinas, sem violar as restrições do job shop. O makespan é definido como o tempo consumido para se





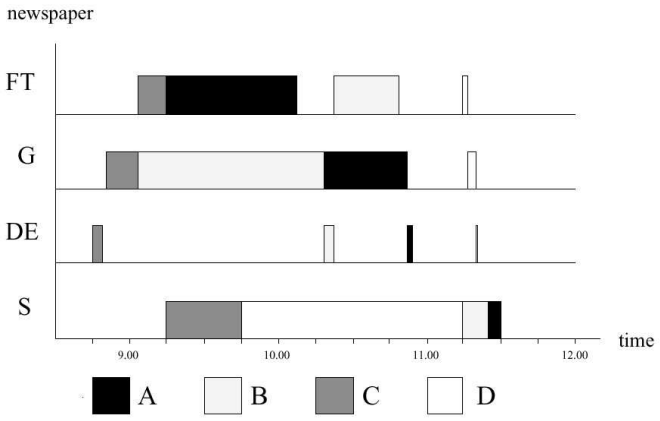
completer todos os processos. O objetivo, portanto, é encontrar um sequenciamento que minimize o makespan. Um bom sequenciamento é aquele que minimiza o tempo total em que as máquinas ficam ociosas.

*Exemplo em FRENCH (1982) – Algy (A), Bertie (B), Charlie (C) e Digby (D) dividem um apartamento e assinam quatro jornais diários: Financial Times (FT), Guardian (G), Daily Express (DE) e Sun (S). Cada um dos moradores se levanta a uma hora específica e lê todos os jornais. A ordem de leitura é específica para cada morador e o tempo de leitura também. Qual é o tempo mais curto em que eles podem concluir a leitura dos jornais?*

Algy se levanta às 8h30, Bertie e Charlie às 8h45 e Digby às 9h30.

| | First | Second | Third | Fourth |
|---|---|---|---|---|
| Algy | FT, 60m | G, 30m | DE, 2m | Sun, 5m |
| Bert | G, 75m | DE, 3m | FT, 25m | Sun, 10m |
| Charles | DE, 5m | G, 15m | FT, 10m | Sun, 30m |
| Digby | Sun, 90m | FT, 1m | G, 1m | DE, 1m |

| | First | Second | Third | Fourth |
|---|---|---|---|---|
| F.T. | C | A | B | D |
| G. | C | B | A | D |
| D.E. | C | B | A | D |
| Sun | C | D | B | A |








Outro exemplo (ainda sem considerar tempo de setup):

| | (m,t) | (m,t) | (m,t) | (m,t) | (m,t) | (m,t) |
|---|---|---|---|---|---|---|
| Job 1: | 3,1 | 1,3 | 2,6 | 4,7 | 6,3 | 5,6 |
| Job 2: | 2,8 | 3,5 | 5,10 | 6,10 | 1,10 | 4,4 |
| Job 3: | 3,5 | 4,4 | 6,8 | 1,9 | 2,1 | 5,7 |
| Job 4: | 2,5 | 1,5 | 3,5 | 4,3 | 5,8 | 6,9 |
| Job 5: | 3,9 | 2,3 | 5,5 | 6,4 | 1,3 | 4,1 |
| Job 6: | 2,3 | 4,3 | 6,9 | 1,10 | 5,4 | 3,1 |

- *Flow shop scheduling*: similar a job shop, mas com todos os processos apresentando a mesma sequência de tarefas.
- *Open shop scheduling*: similar a job shop, mas não existe uma ordem pré-definida das tarefas dentro de cada processo.
- *Tabelas de horários – agenda* (*Timetabling*): desenvolver uma agenda de provas, aulas, organização de horários de trabalho, etc.
- *Gerência de processos e de memória*: alocação de processos computacionais em diferentes processadores, políticas de substituição de processos na memória.

- *Generalized assignment problem*:
  - $m$ máquinas para executar $n$ tarefas, com $m < n$;
  - $a_{ik}$ : recursos consumidos pela máquina $i$ quando executa a tarefa $k$;
  - $b_i$ : capacidade da máquina $i$;
  - $c_{ik}$: custo para se atribuir a tarefa $k$ à máquina $i$;
  - $x_{ik} = 1$ se a máquina $i$ vai executar a tarefa $k$, e $x_{ik} = 0$ caso contrário;
  - Objetivo: Encontre uma atribuição de custo mínimo.

$$\min \sum_{i=1}^{m}\sum_{k=1}^{n} c_{ik} x_{ik}$$

s.a.

$$\sum_{i=1}^{m} x_{ik} = 1,\ k = 1,\ldots,n$$

$$\sum_{k=1}^{n} a_{ik} x_{ik} \le b_i,\ i = 1,\ldots,m$$

$$x_{ik} \in \{0,1\},\ i = 1,\ldots,m;\ k = 1,\ldots,n$$





- *Empacotamento* (*Knapsack*): dado um pacote (p.ex. mochila, caixa, etc.) com uma determinada capacidade e um conjunto de itens cada um com seu tamanho e valor, encontre o conjunto de itens que pode ser acomodado no pacote de modo a maximizar o valor no pacote.

Exemplo:

- ✓ capital $b$ disponível para investimento e $n$ projetos a serem selecionados;
- ✓ $a_i$: capital necessário para se implementar o projeto $i$ ;
- ✓ $c_i$ : retorno financeiro associado ao projeto $i$ ;
- ✓ $x_i = 1$ se o projeto $i$ for selecionado, e $x_i = 0$ caso contrário;
- ✓ Objetivo: Encontre um subconjunto de projetos que não exceda o capital disponível e que maximize o retorno financeiro.

$$\max \sum_{i=1}^{n} c_i x_i$$

$$\text{s.a.} \quad \sum_{i=1}^{n} a_i x_i \leq b$$

$$x_i \in \{0,1\},\ i = 1,\ldots,n$$

## 4.2 Projeto (*Design*)

- *Filtros*: projeto de sistemas eletrônicos ou digitais que implementam uma determinada resposta em frequência, tanto resposta ao impulso finita (FIR) quanto resposta ao impulso infinita (IIR).
- *Processamento de sinais*: otimização do projeto de sistemas de processamento de sinais e desenho de circuitos integrados.





- *Sistemas inteligentes*: definição de arquitetura e/ou parâmetros de redes neurais artificiais, sistemas nebulosos, autômatos celulares, e sistemas imunológicos artificiais, dentre outros.
- *Aplicações em engenharia*: redes de telecomunicações, projetos estruturais, projeto de aeronaves, projeto de estruturas espaciais, projeto de reatores químicos, teste e diagnóstico de falhas, etc.

## 4.3 Identificação e Controle

- A *identificação* envolve a determinação de parâmetros de modelos a partir de um comportamento desejado.
- Duas abordagens distintas para controle de processos: *on-line* e *off-line*.
  - *Off-line*: um algoritmo evolutivo é utilizado para projetar um controlador que é depois utilizado para controlar um sistema.
  - *On-line*: um algoritmo evolutivo é utilizado como uma parte ativa do controlador.

- Uma vantagem de um controlador evolutivo é que ele pode ser utilizado em ambientes dinâmicos, embora com limitações (evolução × adaptação).

### 4.4 Aplicações em Classificação

- *Sistemas classificadores* têm sido utilizados como partes de outros sistemas como, por exemplo, sistemas de controle.
- Algoritmos evolutivos também têm sido aplicados a problemas de jogos (*game playing*).
- Muitas outras aplicações de AEs existem em classificação como, por exemplo, processamento de imagens e mineração de dados.

## 5 Algoritmo Genético Tradicional

- População de tamanho fixo e estrutura de dados do tipo cadeias binárias;
- Seleção natural proporcional ao fitness via algoritmo Roulette Wheel;





- Crossover simples (crossover de um ponto);
- Mutação pontual.

### 5.1 Representação (Estrutura de Dados)

- Cadeias binárias de comprimento fixo.

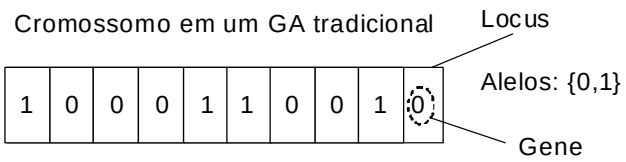
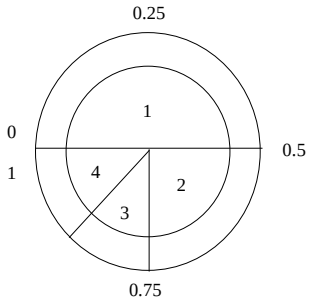



**Figura 1:** Cadeia binária de comprimento $l = 10$ correspondente à estrutura de dados de um algoritmo genético tradicional.

## 5.2 Mecanismo de Seleção

- A seleção no GA tradicional é proporcional ao fitness e é geralmente implementada utilizando um algoritmo denominado de *Roulette Wheel*.
- **Exemplo**:

| N | Cromossomo | Fitness | Graus |
|---|---|---|---|
| 1 | 0001100101010 | 6,0 | 180 |
| 2 | 0101001010101 | 3,0 | 90 |
| 3 | 1011110100101 | 1,5 | 45 |
| 4 | 1010010101001 | 1,5 | 45 |




- Implementação: gerador de números pseudo-aleatórios com distribuição uniforme.
- Note que este procedimento permite a perda ("morte") do melhor indivíduo e também permite que um indivíduo seja selecionado mais do que uma vez.
- As probabilidades de reprodução de cada indivíduo irão resultar na geração de uma nova população composta por indivíduos probabilisticamente selecionados a partir da população atual.





- Os indivíduos selecionados irão gerar probabilisticamente descendentes através de operadores genéticos específicos, particularmente, *crossover* e *mutação*.

## 5.3 Operador de Recombinação (Crossover Simples)

- Nos sistemas biológicos o crossover pode ocorrer durante a reprodução sexuada permitindo a troca de material genético entre dois indivíduos.
- Este processo pode ser abstraído como um operador geral para as estruturas de dados do tipo cadeia binária utilizada no GA tradicional (HOLLAND, 1975):
  - Duas cadeias $\boldsymbol{x} = x_1x_2\ldots x_l$ e $\boldsymbol{y} = y_1y_2\ldots y_l$ de comprimento $l$ são selecionadas com probabilidade de crossover *pc*.
  - Um número $r \in \{1,2,\ldots,l-1\}$ indicando o ponto de cruzamento (crossover) é selecionado.
  - Duas novas cadeias são formadas a partir de $\boldsymbol{x}$ e $\boldsymbol{y}$ através da troca de um conjunto de atributos à direita da posição $r$, resultando em $\boldsymbol{x'} = x_1\ldots x_i y_{i+1}\ldots y_l$ e $\boldsymbol{y'} = y_1\ldots y_i x_{i+1}\ldots x_l$.

- Os dois novos cromossomos gerados ***x'*** e ***y'*** são os descendentes de ***x*** e ***y***.

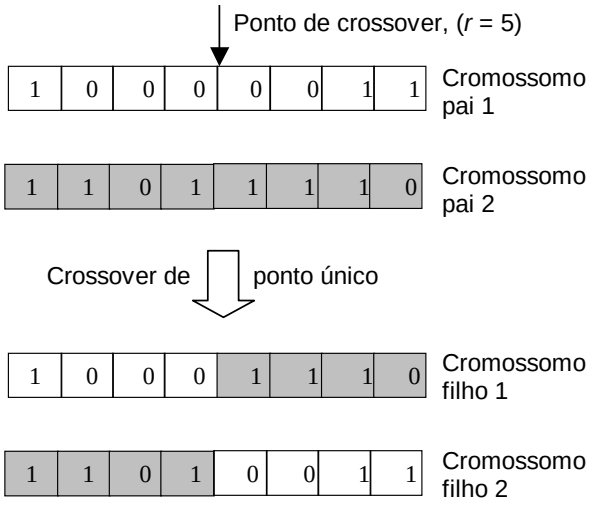


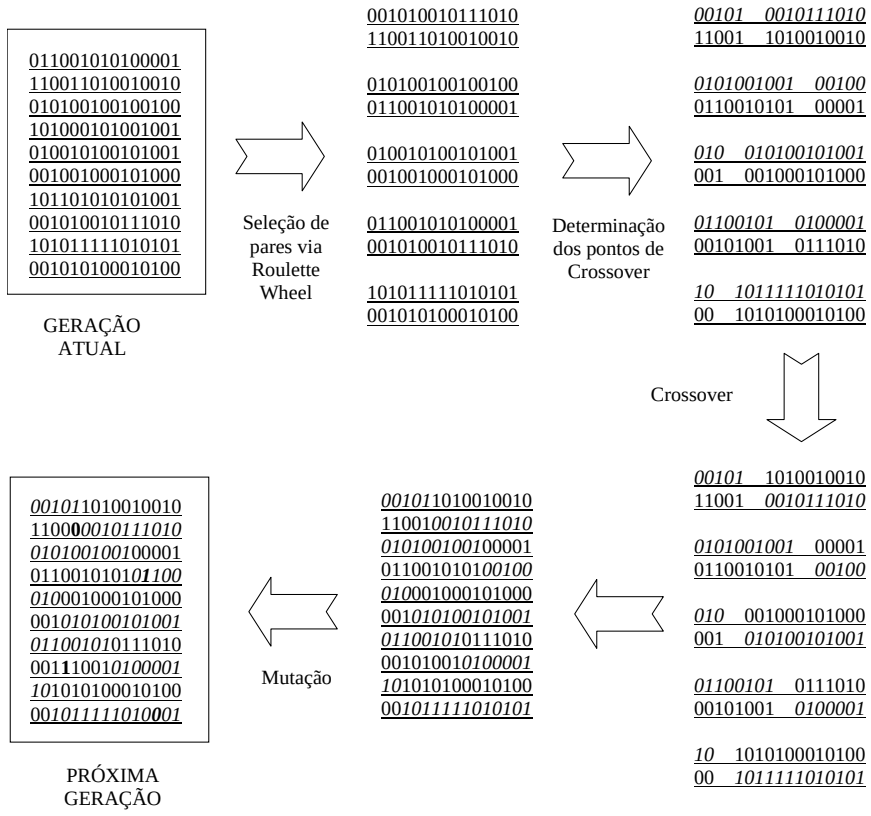

**Figura 2:** Crossover de um único ponto para cadeias de comprimento $l = 8$.





## 5.4 Operador de Mutação

- Em genética a mutação pontual é um processo no qual um alelo de um gene é aleatoriamente substituído (ou modificado) por outro, resultando em um novo cromossomo.
- Geralmente existe uma baixa probabilidade de mutar cada gene de um cromossomo.
- Isso significa que cada bit na população $P$ é operado da seguinte forma:
  - Os números $r,\ldots,u$ indicando as posições que irão sofrer mutação são determinadas aleatoriamente de forma que cada posição possui uma pequena probabilidade $pm$ de sofrer mutação, independente das outras posições.
  - Uma nova cadeia $\boldsymbol{x'} = x_1 \ldots x_r \ldots x_u \ldots x_l$ é gerada onde $x_r \ldots x_u$ são determinadas aleatoriamente partindo do conjunto de alelos para cada gene. No caso de cadeias binárias, se uma posição possui alelo '0' então ela se torna '1', e vice-versa.

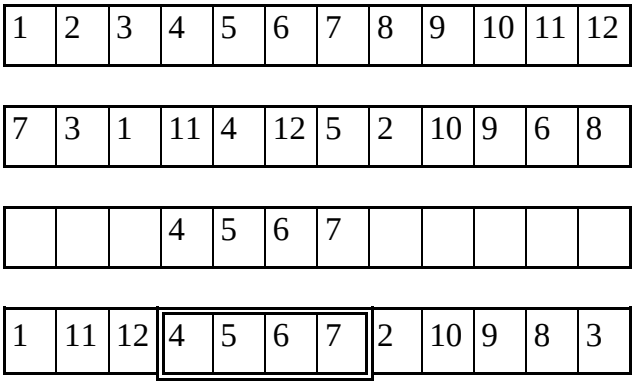




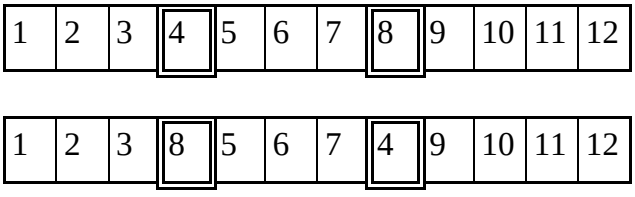



## 5.5 Operador de mutação para permutações

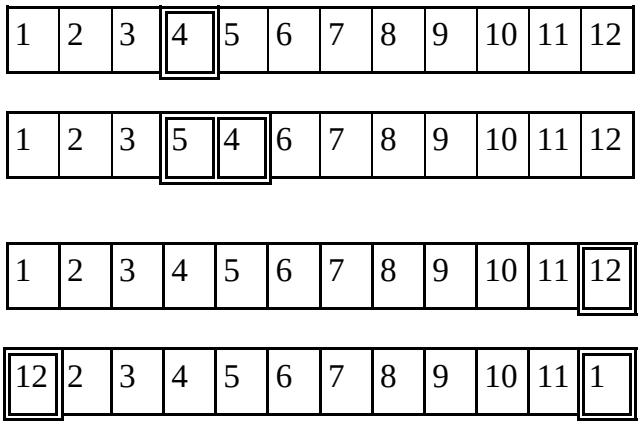
- Crossover OX:




- Mutação Inversiva



- Variação de Mutação Inversiva




## 5.6 Problemas com o algoritmo padrão

- Política de reprodução/seleção permite a perda do melhor indivíduo;





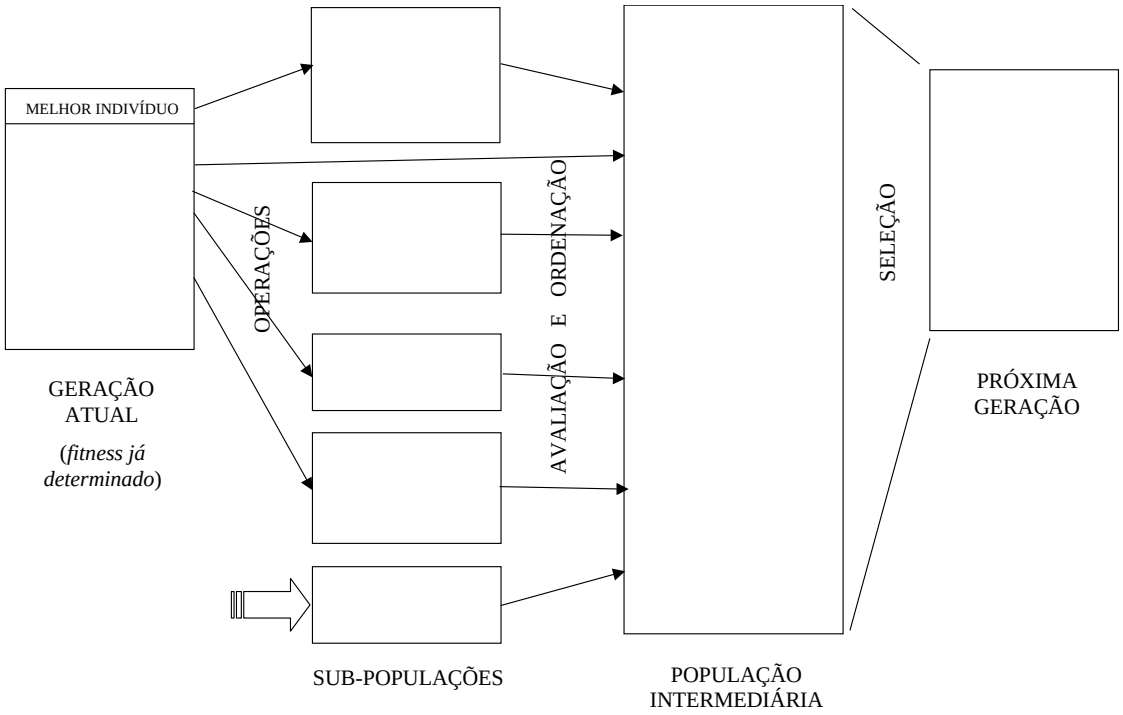
- Posição dos genes no cromossomo influi na probabilidade de permanecerem no mesmo cromossomo após crossover;
- Dificuldades na codificação binária de números reais.

## 5.7 Algumas estratégias de solução

- Mecanismos alternativos de seleção;
- Crossover uniforme;
- Codificação em arranjos de números em ponto flutuante.

## 5.8 Algoritmo Genético Modificado

- Geração de sub-populações por meio da aplicação de operadores genéticos e outros operadores sobre membros da geração atual;
- Avaliação (cálculo do *fitness*) e ordenação da população intermediária;
- Seleção para nova geração.








## 5.9 Teoria dos Esquemas (*schemata theory*)

- A teoria dos esquemas foi proposta por HOLLAND (1975; 1992) para tentar explicar por que os algoritmos genéticos funcionam.
- Nesta seção, apresentaremos os principais resultados da teoria dos esquemas: o teorema de crescimento dos esquemas e a hipótese dos blocos construtivos.
- Um *esquema* é uma representação capaz de descrever diversos cromossomos simultaneamente. Um esquema é construído inserindo um caractere *don't care* ($*$) no alfabeto dos genes, indicando que aquele gene representa qualquer alelo.
  - Por exemplo, o esquema $[1 * 0 1 0 0 1]$ representa os cromossomos [1 **0** 0 1 0 0 1] e [1 **1** 0 1 0 0 1]. O esquema $[1 * 0 * 1 1 0]$ representa quatro cromossomos: [1 **0** 0 **0** 1 1 0], [1 **0** 0 **1** 1 1 0], [1 **1** 0 **0** 1 1 0] e [1 **1** 0 **1** 1 1 0].
- Obviamente, o esquema [1 1 1 0 0 1 0] representa apenas um cromossomo, enquanto que o esquema $[* * * * * * *]$ representa todos os cromossomos de comprimento 7, ou seja, $2^7$ cromossomos.

- Observe que cada esquema representa $2^r$ cromossomos, onde $r$ é o número de caracteres *don't care* "$*$" presentes no esquema. Por outro lado, cada cromossomo de comprimento $m$ pode ser representado por $2^m$ esquemas.
  - Por exemplo, considere o cromossomo [0 1 0 0 1 0 0]. Este cromossomo é representado pelos seguintes $2^7$ esquemas:

[0 1 0 0 1 0 0]
[* 1 0 0 1 0 0]
[0 * 0 0 1 0 0]
⋮
[0 1 0 0 1 0 *]
[* * 0 0 1 0 0]
[* 1 * 0 1 0 0]
⋮
[0 1 0 0 1 * *]
[* * * 0 1 0 0]
⋮
[* * * * * * *]





- Considerando cromossomos de comprimento $m$, há um total de $3^m$ possíveis esquemas.
- A *ordem* de um esquema $S$, $o(S)$, é definida como o número de 0's e 1's presentes no esquema, isto é, o número de *posições fixas* (caracteres diferentes de *don't care*) presentes no esquema. A ordem de um esquema define sua especificidade, de modo que quanto maior a ordem, mais específico é o esquema.
- O *comprimento definitório* de um esquema $S$, denotado por $\delta(S)$, é a maior distância entre posições fixas de um cromossomo. O comprimento definitório define o nível de compactação da informação contida no esquema.
- O *fitness* de um esquema $S$ na geração $t$, eval($S$, $t$), é definido como a média dos *fitness* de todos os cromossomos na população representados pelo esquema $S$. Considere que há $p$ cromossomos $\left\{\mathbf{x}_{i_1}^t,\ldots,\mathbf{x}_{i_p}^t\right\}$ representados pelo esquema $S_i$ na geração $t$. Então:

$$\text{eval}(S_i, t) = \frac{1}{p}\sum_{j=1}^{p}\text{eval}\left(\mathbf{x}_{i_j}^t\right),$$

onde $\text{eval}\left(\mathbf{x}_{i_j}^t\right)$ é o *fitness* do indivíduo $\mathbf{x}_{i_j}^t$.

- Seja *tam_pop* o tamanho da população. O *fitness* médio da população na geração *t*, $\overline{F}(t)$, é dado por

$$\overline{F}(t) = \frac{1}{tam\_pop}\sum_{i=1}^{tam\_pop}\text{eval}\left(\mathbf{x}_i^t\right).$$

- Sejam $p_c$ e $p_m$ as probabilidades de *crossover* e mutação, respectivamente, e *m* o comprimento dos cromossomos. Seja $\xi(S_i, t)$ o número de cromossomos representados pelo esquema $S_i$ na geração *t*. Pode-se mostrar que (MICHALEWICZ, 1996):

$$\xi(S_i, t+1) \geq \frac{\xi(S_i, t)\text{eval}(S_i, t)}{\overline{F}(t)}\left[1 - p_c\frac{\delta(S_i)}{m-1} - o(S_i)p_m\right]$$





- A equação acima é conhecida como *equação de crescimento reprodutivo do esquema*. Esta equação é deduzida supondo que a função de *fitness* $f(\cdot)$ produz apenas valores positivos. Se a função a ser otimizada produz valores negativos, um mapeamento entre as funções de otimização e de *fitness* é necessário.
- Esta equação de crescimento mostra que a seleção aumenta a amostragem de esquemas cujo *fitness* está acima da média da população, e este aumento é exponencial (MICHALEWICZ, 1996).
- A seleção, por si só, não introduz nenhum novo esquema (não representado na geração inicial em $t = 0$). Esta é a razão da introdução do operador de *crossover*: possibilitar a troca de informação estruturada, ainda que aleatória. Além disso, o operador de mutação introduz uma variabilidade maior na população.
- O efeito (destrutivo) combinado destes operadores não é significativo se o esquema é curto e de ordem baixa. O resultado final da equação de crescimento pode ser formulado como segue:

**Teorema dos Esquemas:** Esquemas com comprimento definitório curto, de ordem baixa e com *fitness* acima da média têm um aumento exponencial de sua participação em gerações consecutivas de um algoritmo genético.

Prova: Veja HOLLAND (1975; 1992).

- Uma consequência imediata deste teorema é que os algoritmos genéticos tendem a explorar o espaço por meio de esquemas curtos e de baixa ordem que, subsequentemente, são usados para troca de informação durante o *crossover*.

**Hipótese dos Blocos Construtivos**: Um algoritmo genético busca desempenho quase-ótimo através da justaposição de esquemas curtos, de baixa ordem e alto desempenho, chamados de *blocos construtivos*.

- Em uma população de tamanho *tam_pop*, indivíduos de comprimento *m* processam pelo menos $2^m$ e no máximo $2^{tam\_pop}$ esquemas. Alguns deles são processados de forma útil: são amostrados a uma taxa crescente exponencial (desejável); e outros são quebrados por meio de *crossover* e mutação.





- HOLLAND (1975; 1992) mostrou que, em uma população de tamanho *tam_pop*, pelo menos $tam\_pop^3$ são processados de forma útil. Esta propriedade foi denominada *paralelismo implícito*, pois é obtida sem nenhuma exigência extra de memória e processamento. Entretanto, BERTONI & DORIGO (1993) mostraram que a estimativa $tam\_pop^3$ é válida apenas para o caso particular em que *tam_pop* é proporcional a $2^l$, onde $l = \frac{1}{2} m\varepsilon$ e $\varepsilon$ é a probabilidade de um esquema ser rompido por *crossover*.
- Note, entretanto, que em certos problemas alguns blocos construtivos (esquemas curtos, de ordem baixa) podem direcionar erroneamente o algoritmo, levando-o a convergir para pontos sub-ótimos. Este fenômeno é conhecido como **decepção**. Assim, a hipótese dos blocos construtivos não fornece uma explicação definitiva para o funcionamento dos algoritmos genéticos. Ela é apenas uma indicação do motivo pelo qual os algoritmos genéticos funcionam para uma certa classe de problemas.

### *5.10 Deception Problems*

- Alguns blocos construtivos podem direcionar erroneamente o algoritmo genético, levando-o a convergir para pontos sub-ótimos:
  - ⟨1 1 1 * * * * * * * *⟩ – fitness *acima* da média
  - ⟨* * * * * * * * * 1 1⟩ – fitness *acima* da média
  - ⟨1 1 1 * * * * * * 1 1⟩ – fitness *muito menor* que ⟨0 0 0 * * * * * * 0 0⟩
  - solução ótima – ⟨1 1 1 1 1 1 1 1 1 1⟩
  - tendência a convergir para pontos como ⟨0 0 0 1 1 1 1 1 1 0 0⟩

- Algumas alternativas foram propostas para combater o problema da decepção (GOLDBERG, 1989; MICHALEWICZ, 1996). A primeira considera que há algum conhecimento a priori da função-objetivo para que seja possível codificá-la de forma apropriada (que forme blocos construtivos "coesos").





- A segunda opção é utilizar algoritmos genéticos *messy* (GOLDBERG, 1989), que diferem do algoritmo genético tradicional de várias maneiras: codificação, operadores, presença de cromossomos de tamanho distinto e fases evolutivas.
- A terceira opção é o emprego de algoritmos de estimação de distribuição, a serem abordados no Tópico 14 deste curso.

## 6 Referências bibliográficas

BÄCK, T., FOGEL, D.B. & MICHALEWICZ, Z. (eds.) "Evolutionary Computation 1: Basic Algorithms and Operators", Institute of Physics Publishing, 2000a.

BÄCK, T., FOGEL, D.B. & MICHALEWICZ, Z. (eds.) "Evolutionary Computation 2: Advanced Algorithms and Operators", Institute of Physics Publishing, 2000b.

BÄCK, T., "Evolutionary Algorithms: Comparison of Approaches", in R. Paton (ed.) *Computing with Biological Metaphors*, Chapman & Hall, Capítulo 14, pp. 227-243, 1994.

BERTONI, A. & DORIGO, M. "Implicit Parallelism in Genetic Algorithms", *Artificial Intelligence*, vol. 61, no. 2, pp. 307-314, 1993.

FOGEL, D. B. Evolutionary Computation – Toward a New Philosophy of Machine Intelligence, IEEE Press, 1999.

FOGEL, L.J., OWENS, A.J. & WALSH, M.J. Artificial Intelligence through Simulated Evolution. John Wiley, 1966.

FRENCH, S. "Sequencing and Scheduling: An introduction to the mathematics of the job-shop", Ellis Horwood Limited, 1982.

GOLDBERG, D. E. "Messy Genetic Algorithms: Motivation, Analysis, and First Results", *Complex Systems*, 3: 493-530, 1989.

HOLLAND, J.H. Adaptation in Natural and Artificial Systems. The University of Michigan Press, 1st. ed. 1975 (The MIT Press, 2nd. ed. 1992).

KOZA, J.R. Genetic Programming: On the Programming of Computers by means of Natural Selection, MIT Press, 1992.

MICHALEWICZ, Z. "Genetic Algorithms + Data Structures = Evolution Programs", Springer, 1996.

RECHENBERG, I. Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution. Frommann-Holzboog Verlag, 1973.

SCHWEFEL, H.-P. Evolutionsstrategie und numerische Optimierung. Tese de Doutorado, Technische Universität Berlin, 1975.

SCHWEFEL, H.-P. Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. Basel: Birkhäuser, 1977.

*<u>Observação</u>: Parte dessas notas de aula foi baseada em material gerado no ano de 2002 por Fernando J. Von Zuben e Leandro Nunes de Castro, quando ambos ministraram a disciplina de IA707 na FEEC/Unicamp.*