Algoritmos Genéticos (AG's)

1 Introdução

- os algoritmos genéticos (AG's) foram desenvolvidos por HOLLAND (1975; 1992), da Universidade de Michigan.
- metas:
  - abstrair e rigorosamente explicar os processos adaptativos em sistemas naturais
  - desenvolver simulações em computador que retenham os mecanismos originais encontrados em sistemas naturais
- características principais:
  - existência de 2 espaços de trabalho: espaço **genotípico** e espaço **fenotípico**
  - AG's fazem busca sobre uma **população** de pontos e não sobre um único ponto
  - AG's fazem uso de descrições genéricas do que se quer ver presente na solução, através de **funções de *fitness*** (funções-objetivo)
  - AG's utilizam regras de transição **probabilísticas**, e não regras determinísticas

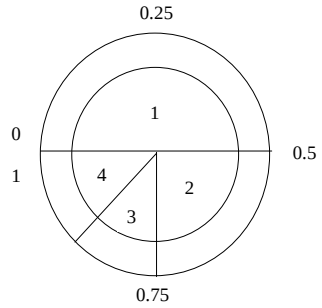
2 Algoritmo Genético Clássico

- codificação binária
- reprodução e seleção natural via algoritmo Roulette Wheel
- crossover simples (crossover de um ponto)
- mutação

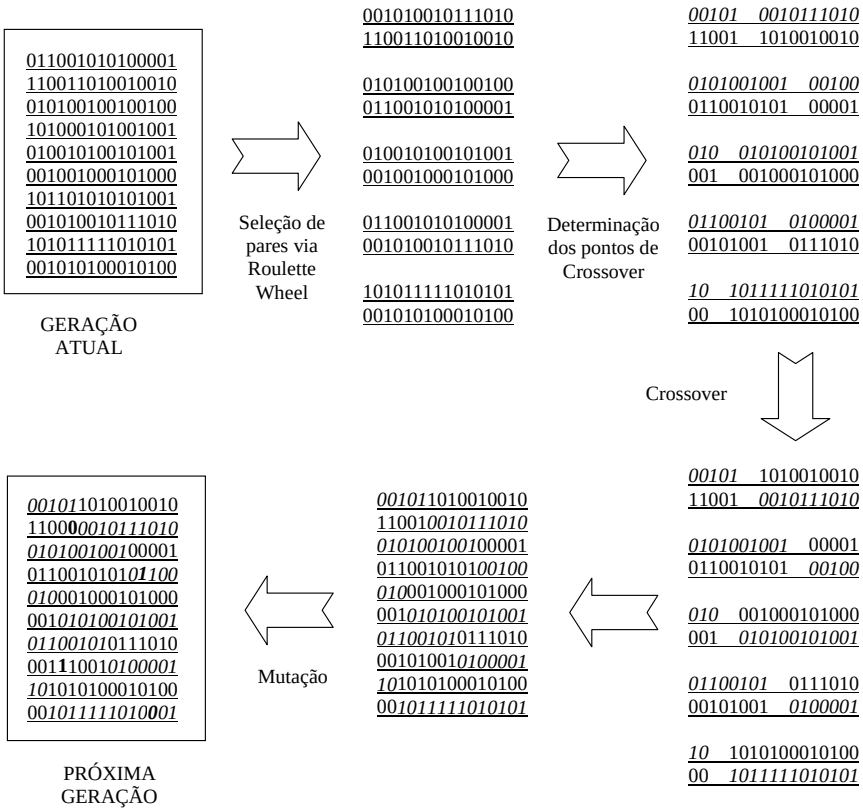
- **Roullete Wheel** : probabilidade de seleção de um cromossomo é diretamente proporcional a seu valor de *fitness*.

- **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



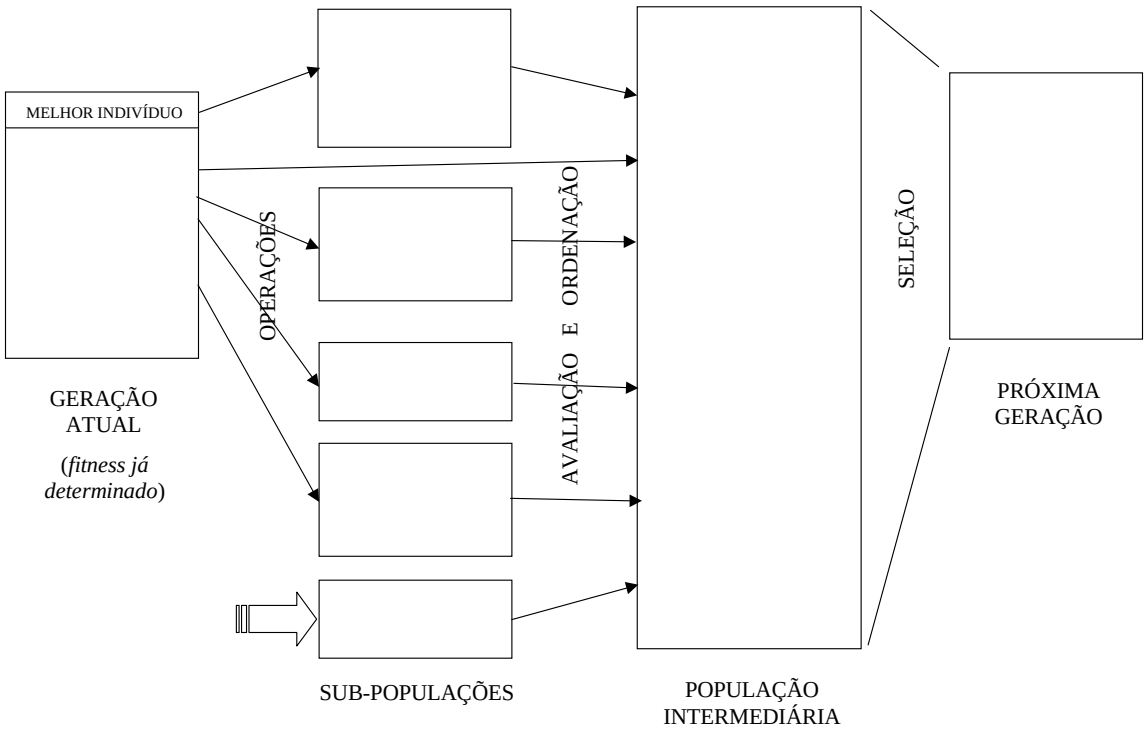
- implementação: gerador de números pseudo-aleatórios com distribuição uniforme



- **problemas com o algoritmo clássico:**
  - 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
- **algumas estratégias de solução:**
  - mecanismos alternativos de seleção
  - crossover uniforme
  - codificação em arranjos de números reais

3 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



4 Exemplos de Operações (operadores genéticos)

- crossover de 1 ou *n* pontos entre indivíduos aleatórios (Roulette Wheel), ou então entre indivíduo aleatório (Roulette Wheel) e melhor indivíduo. Neste caso, os segmentos definem as características.
- crossover uniforme (SYSWERDA, 1989) ou baseado em máscara entre indivíduos aleatórios (Roulette Wheel), ou então entre indivíduo aleatório (Roulette Wheel) e melhor indivíduo. Neste caso, os genes definem as características.
- mutação de indivíduos aleatórios (Roulette Wheel), ou do melhor indivíduo, da geração atual:

- mutação aleatória
- mutação indutiva (somente para codificação usando números reais)

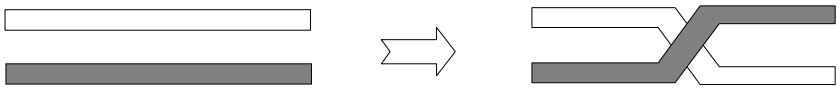
- outros operadores projetados para realização de buscas locais guiadas (heurísticas dependentes da natureza do problema) → **algoritmos meméticos**

4.1 Crossover de 1 ou *n* pontos entre indivíduos aleatórios (Roulette Wheel)

- são escolhidos dois indivíduos da geração atual, aleatoriamente ou por meio do Roulette Wheel, determina-se o(s) ponto(s) de corte e efetua-se o crossover
- parâmetros: ponto de corte, método de seleção dos indivíduos

4.2 Crossover de 1 ou *n* pontos entre indivíduo aleatório (Roulette Wheel) e melhor indivíduo

- é escolhido um indivíduo da geração atual, aleatoriamente ou por meio do Roulette Wheel, seleciona-se o melhor indivíduo da geração atual, determina-se o(s) ponto(s) de corte e efetua-se o crossover
- parâmetros: ponto de corte, método de seleção de um dos indivíduos



Exemplo de crossover de 1 ponto

4.3 Crossover uniforme ou baseado em máscara entre indivíduos aleatórios (Roulette Wheel)

- são escolhidos dois indivíduos da geração atual, aleatoriamente ou por meio do Roulette Wheel, determina-se a percentagem de troca e escolhe-se os genes que vão ser trocados ou então aplica-se a máscara, efetuando o crossover somente dos genes envolvidos
- parâmetros: percentagem de genes a serem trocados ou então máscara, método de seleção dos indivíduos

4.4 Crossover Uniforme ou baseado em máscara entre indivíduo aleatório (Roulette Wheel) e melhor indivíduo

- análogo aos casos anteriores.

Exemplo de crossover uniforme, com taxa de 50%

4.5 Mais sobre crossover

- ESHELMAN *et al.* (1989) relata diversos experimentos com vários operadores de *crossover*. Os resultados indicam que o operador com pior desempenho médio é o *crossover* de um ponto; entretanto, não há nenhum operador de *crossover* que claramente apresente um desempenho superior aos demais.

- uma conclusão a que se pode chegar é que cada operador de *crossover* é particularmente eficiente para uma determinada classe de problemas e extremamente ineficiente para outras.

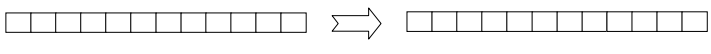
- os operadores de *crossover* descritos até aqui também podem ser utilizados em cromossomos com codificação em ponto flutuante (valores reais). Entretanto, existem operadores de *crossover* especialmente desenvolvidos para uso com codificação em ponto flutuante. Um exemplo é o chamado *crossover aritmético* (MICHALEWICZ, 1996).

- este operador é definido como uma combinação linear de dois vetores (cromossomos): sejam $\mathbf{x}_1$ e $\mathbf{x}_2$ dois indivíduos selecionados para *crossover*, então os dois filhos resultantes serão $\mathbf{x}_1' = a\mathbf{x}_1 + (1-a)\mathbf{x}_2$ e $\mathbf{x}_2' = (1-a)\mathbf{x}_1 + a\mathbf{x}_2$, onde a é um número aleatório pertencente ao intervalo $[0, 1]$. Este operador é particularmente apropriado para problemas de otimização numérica com restrições, onde a região factível é convexa. Isto porque, se $\mathbf{x}_1$ e $\mathbf{x}_2$ pertencem à região factível, combinações convexas de $\mathbf{x}_1$ e $\mathbf{x}_2$ serão também factíveis. Assim, garante-se que o *crossover* não gera indivíduos inválidos para o problema em questão.

- outros exemplos de *crossover* especialmente desenvolvidos para utilização em problemas de otimização numérica restritos e codificação em ponto flutuante são o *crossover geométrico* e o *crossover esférico,* descritos em MICHALEWICZ & SCHOENAUER (1996). Veja também BÄCK *et al.* (2000b).

4.6 Mutação de indivíduos aleatórios (Roulette Wheel), ou do melhor indivíduo

- mutação aleatória
  - é escolhido um indivíduo da geração atual, aleatoriamente ou por meio do Roulette Wheel, ou então é tomado o melhor indivíduo da população (aquele que apresenta o maior *fitness*), determina-se a percentagem de genes que vão ser trocados e procede-se a um sorteio de genes na quantidade determinada, ou então emprega-se uma máscara, e para cada gene sorteado define-se aleatoriamente um dentre os alelos disponíveis para troca.
  - parâmetros: percentagem de genes a serem trocados ou então máscara, conjunto de alelos, método de seleção de indivíduos.
- mutação indutiva (somente para codificação usando números reais)
  - Semelhante à mutação aleatória, só que ao invés de sortear um novo valor para o gene, sorteia-se um valor a ser somado ao valor corrente do gene



4.7 Mais sobre mutação

- no caso de problemas com codificação em ponto flutuante, o operador de mutação mais popular é a *mutação gaussiana* (MICHALEWICZ & SCHOENAUER, 1996), que modifica todos os componentes de um cromossomo $\mathbf{x} = [x_1 \ldots x_n]$ na forma:

$$\mathbf{x}' = \mathbf{x} + N(0, \sigma),$$

  onde $N(0, \sigma)$ é um vetor de variáveis aleatórias gaussianas independentes, com média zero e desvio padrão σ.

- um outro operador importante para problemas em que os indivíduos empregam codificação em ponto flutuante é a *mutação uniforme* (MICHALEWICZ ,1996). Este operador seleciona aleatoriamente um componente $k \in \{1, 2, \ldots, n\}$ do cromossomo $\mathbf{x} = [x_1 \ldots x_k \ldots x_n]$ e gera um indivíduo $\mathbf{x}' = [x_1 \ldots x_k' \ldots x_n]$, onde x_k' é um número aleatório (com distribuição de probabilidade uniforme) amostrado no intervalo $[LB, UB]$ e LB e UB são, respectivamente, os limites inferior e superior da variável x_k.

- outro operador de mutação, especialmente desenvolvido para problemas de otimização com restrição e codificação em ponto flutuante, é a chamada *mutação não-uniforme* (MICHALEWICZ, 1996; MICHALEWICZ & SCHOENAUER, 1996). A mutação não-uniforme é um operador dinâmico, destinado a melhorar a sintonia fina de uma elemento simples. Podemos defini-lo da seguinte forma: seja $\mathbf{x}^t = [x_1 \ldots x_n]$ um cromossomo e suponha que o elemento x_k foi selecionado para mutação; o cromossomo resultante será $\mathbf{x}^{t+1} = [x_1 \ldots x_k' \ldots x_n]$, onde

$$x_k' = \begin{cases} x_k + \Delta(t, UB - x_k), & \text{com 50\% de probabilidade} \\ x_k - \Delta(t, x_k - LB), & \text{com 50\% de probabilidade} \end{cases}$$

  onde LB e UB são os limites inferiores e superiores da variável x_k. A função $\Delta(t, y)$ retorna um valor no intervalo $[0, y]$ tal que a probabilidade de $\Delta(t, y)$ ser próximo de zero aumenta à medida que t aumenta.

- esta propriedade faz com que este operador inicialmente explore o espaço globalmente (quando *t* é pequeno) e localmente em gerações avançadas (quando *t* é grande);

- MICHALEWICZ (1996) propõe a seguinte função:

$$\Delta(t, y) = y \cdot \left(1 - r^{(1-t/T)^b}\right)$$

  onde *r* é um número aleatório no intervalo [0, 1], *T* é o número máximo de gerações e *b* é um parâmetro que determina o grau de dependência do número de iterações (valor proposto por MICHALEWICZ (1996): $b = 5$).

- outros exemplos de operadores de mutação para problemas de otimização numérica e com codificação em ponto flutuante podem ser encontrados em MICHALEWICZ & SCHOENAUER (1996) e em BÄCK *et al.* (2000b).

5 Mecanismos de Seleção

- o algoritmo genético clássico utiliza um esquema de seleção de indivíduos para a próxima geração chamado *roulette wheel* (MICHALEWICZ, 1996).
- o *roulette wheel* atribui a cada indivíduo de uma população uma probabilidade de passar para a próxima geração proporcional ao seu *fitness* medido, em relação à somatória do *fitness* de todos os indivíduos da população.
- assim, quanto maior o *fitness* de um indivíduo, maior a probabilidade dele passar para a próxima geração.
- observe que a seleção de indivíduos por *roulette wheel* pode fazer com que o melhor indivíduo da população seja perdido, ou seja, não passe para a próxima geração.
- uma alternativa é escolher como solução o melhor indivíduo encontrado em todas as gerações do algoritmo.

- outra opção é simplesmente manter sempre o melhor indivíduo (ou os *P*% melhores indivíduos) da geração atual na geração seguinte, estratégia essa conhecida como *seleção elitista* (FOGEL, 1994; MICHALEWICZ, 1996).
- outro exemplo de mecanismo de seleção é a *seleção baseada em rank* (BÄCK *et al.*, 2000a). Esta estratégia utiliza as posições dos indivíduos quando ordenados de acordo com o *fitness* para determinar a probabilidade de seleção.
- podem ser usados mapeamentos lineares ou não-lineares para determinar a probabilidade de seleção. Para um exemplo de mapeamento não-linear, veja (MICHALEWICZ, 1996). Uma variação deste mecanismo é simplesmente passar os *N* melhores indivíduos para a próxima geração.

- a seguir, citamos alguns outros possíveis mecanismos de seleção:

➢ <u>seleção por diversidade</u>: são selecionados os indivíduos mais diversos da população. Aqui podem ser adotados critérios adicionais relacionados ao *fitness*.

➢ <u>seleção bi-classista</u>: são selecionados os *P*% melhores indivíduos e os (100 – *P*)% piores indivíduos.

➢ <u>seleção aleatória</u>: são selecionados aleatoriamente *N* indivíduos da população. Podemos subdividir este mecanismo de seleção em:

- Salvacionista: seleciona-se o melhor indivíduo e os outros aleatoriamente.
- Não-salvacionista: seleciona-se aleatoriamente todos os indivíduos.

➢ <u>seleção por torneio</u>: é um dos mais refinados processos de seleção, por permitir ajustar a pressão seletiva. A seleção é feita em função do número de vitórias de cada indivíduo em *q* competições contra oponentes aleatórios da população, sendo que vence uma competição aquele que apresentar o maior *fitness* (comparado ao de seu oponente). Para propósitos práticos, $q \geq 10$ conduz a uma forte pressão seletiva, enquanto valores de *q* entre 3 e 5 levam a uma fraca pressão seletiva. Para $q = 1$, temos essencialmente *random walk* e para $q \rightarrow \infty$ temos simplesmente a seleção por ordem de *fitness*, sem nenhuma aleatoriedade.

6 Codificação

- cada indivíduo de uma população representa um candidato em potencial à solução do problema em questão. No algoritmo genético clássico, proposto por HOLLAND (1975; 1992), as soluções candidatas são codificadas em arranjos binários de tamanho fixo.
- a motivação para o uso de codificação binária vem da teoria dos esquemas (*schemata theory*), que será apresentada mais adiante. HOLLAND (1975; 1992) argumenta que seria benéfico para o desempenho do algoritmo maximizar o paralelismo implícito inerente ao algoritmo genético, e prova que um alfabeto binário maximiza o paralelismo implícito.
- entretanto, em diversas aplicações práticas a utilização de codificação binária leva a um desempenho insatisfatório. Em problemas de otimização numérica com parâmetros reais, algoritmos genéticos com representação em ponto flutuante freqüentemente apresentam desempenho superior à codificação binária.

- MICHALEWICZ (1996) argumenta que a representação binária apresenta desempenho pobre quando aplicada a problemas numéricos com alta dimensionalidade e onde alta precisão é requerida. Suponha por exemplo, que temos um problema com 100 variáveis com domínio no intervalo [−500, 500] e que precisamos de 6 dígitos de precisão após a casa decimal. Neste caso precisaríamos de um cromossomo de comprimento 3000, e teríamos um espaço de busca de dimensão aproximadamente 10^{1000}. Neste tipo de problema o algoritmo genético clássico apresenta desempenho pobre.
- MICHALEWICZ (1996) apresenta também simulações computacionais comparando o desempenho de algoritmos genéticos com codificação binária e com ponto flutuante, aplicados a um problema de controle. Os resultados apresentados mostram uma clara superioridade da codificação em ponto flutuante.
- a argumentação de MICHALEWICZ (1996) de que o desempenho de um algoritmo genético com codificação binária é pobre quando o espaço de busca é de dimensão

elevada, não é universalmente aceita na literatura referente a algoritmos genéticos. FOGEL (1994) argumenta que o espaço de busca por si só (sem levar em conta a escolha da representação) não determina a eficiência do algoritmo genético.

- espaços de busca de dimensão elevada podem às vezes ser explorados eficientemente, enquanto que espaços de busca de dimensão reduzida podem apresentar dificuldades significativas. FOGEL (1994), entretanto, concorda que a maximização do paralelismo implícito nem sempre produz um desempenho ótimo.
- fica claro, portanto, que a codificação é uma das etapas mais críticas na definição de um algoritmo genético. A definição inadequada da codificação pode levar a superfícies de *fitness* extremamente "acidentadas".
- em problemas de otimização restrita, a codificação adotada pode fazer com que indivíduos modificados por *crossover*/mutação sejam inválidos.
- nestes casos, cuidados especiais devem ser tomados na definição da codificação e/ou dos operadores (BÄCK *et al.*, 2000b).

7 Abordagens baseadas em população

- **Abordagem Michigan** – a população como um todo é a solução para o problema
- **Abordagem Pittsburgh** – cada elemento da população corresponde a uma solução do problema

8 Definição da População Inicial

- o método mais comum utilizado na criação da população é a inicialização aleatória dos indivíduos. Se algum conhecimento inicial a respeito do problema estiver disponível, pode ser utilizado na inicialização da população.
- por exemplo, se é sabido que a solução final (assumindo codificação binária) vai apresentar mais 0's do que 1's, então esta informação pode ser utilizada, mesmo que não se saiba exatamente a proporção.
- em problemas com restrição, deve-se tomar cuidado para não gerar indivíduos inválidos já na etapa de inicialização.

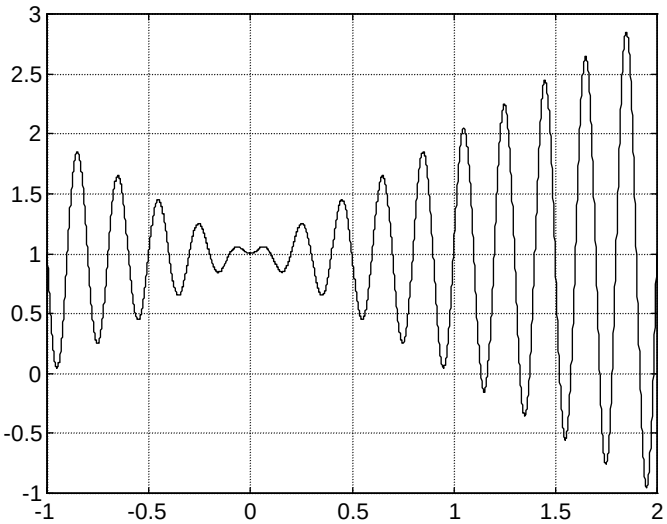
9 Decisões críticas

- tipo de codificação: binária ou real?
- representar toda ou parte da solução?
- só crossover, só mutação, ou ambos?
- populações de tamanho fixo ou variável ?
- cromossomos de tamanho fixo ou variável?
- cromossomos haplóides ou diplóides?
- qual critério de parada?
- usar controle de diversidade?
- usar busca local?
- usar *niching*?
- usar co-evolução?
- abordagem Michigan ou Pittsburgh?
- algoritmos genéticos ou programação genética?

10 Exercícios computacionais

10.1 Otimização de uma Função Simples

$$f(x) = x * \text{sen}(10\,\pi * x) + 1, \quad \text{no intervalo } x \in [-1,2]$$



Soluções:

- $x_i = \frac{2i-1}{20} + \varepsilon_i$, para i = 1,2,...
- $x_0 = 0$
- $x_i = \frac{2i+1}{20} - \varepsilon_i$, para i = –1, –2, ...

Codificação: binária (22 bits) – 6 casas decimais

$$v = \langle b_{21} \quad b_{20} \quad \cdots \quad b_0 \rangle \qquad x' = \sum_{i=0}^{21} b_i \cdot 2^i \qquad x = -1 + x' \frac{2-(-1)}{2^{22}-1}$$

Exemplo: $v = \langle 1000101110110101000111 \rangle \rightarrow x = 0{,}637197$, pois

$$x' = 2288967 \quad \text{e} \quad x = -1 + 2288967 \cdot \frac{3}{4194303} = 0{,}637197$$

$$v = \langle 00000000000000000000 \rangle \rightarrow x = -1$$

$$v = \langle 11111111111111111111 \rangle \rightarrow x = 2$$

Função de Avaliação: eval(v) = $f(x)$

Operadores Genéticos: Crossover simples, mutação simples

População Inicial - Aleatória com 50 indivíduos

Parâmetros:

- probabilidade de crossover p_c = 0,25 e probabilidade de mutação p_m = 0,01

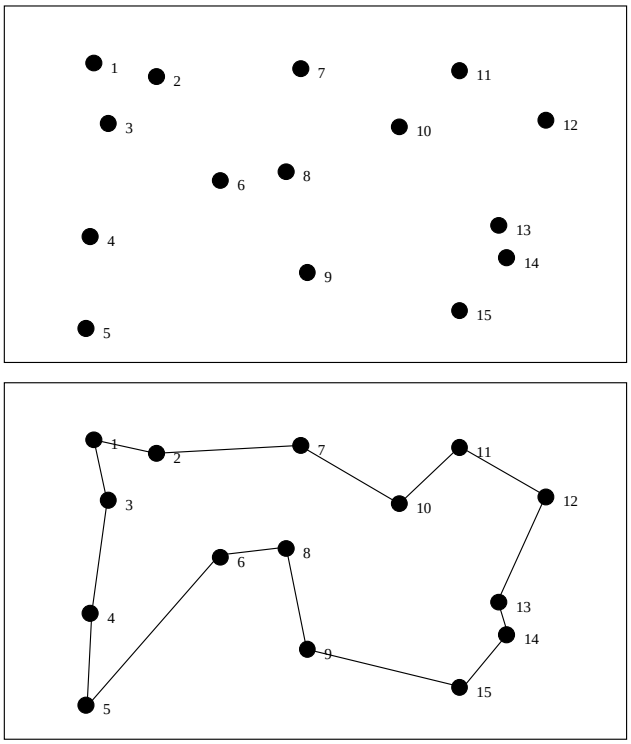
EC3 - Implementar o algoritmo genético padrão para este caso e testá-lo para até 150 gerações.

EC4 - Implementar o algoritmo genético modificado para o mesmo caso, e comparar o desempenho. Arbitrar os parâmetros do algoritmo genético modificado.

10.2 Problema do Dilema do Prisioneiro

EC5 – Implementar o algoritmo genético modificado para o problema do dilema do prisioneiro, testando (a) contra uma população fixa; (b) contra os próprios indivíduos da mesma geração, incluindo o próprio.

10.3 Problema do Caixeiro Viajante (Traveling Salesman Problem – TSP)



Codificação - arranjo de números inteiros: ⟨1,3,4,5,6,8,9,15,14,13,12,11,10,7,2⟩

Função de *fitness* - soma dos custos de movimentação entre cada par de cidades.

Operadores

- Crossover OX:

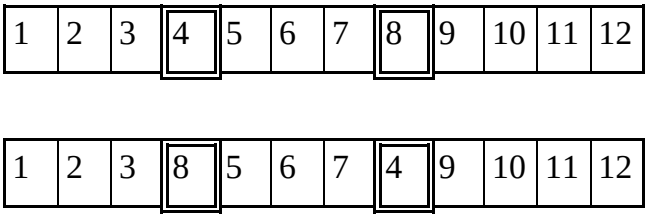
1	2	3	4	5	6	7	8	9	10	11	12

7	3	1	11	4	12	5	2	10	9	6	8

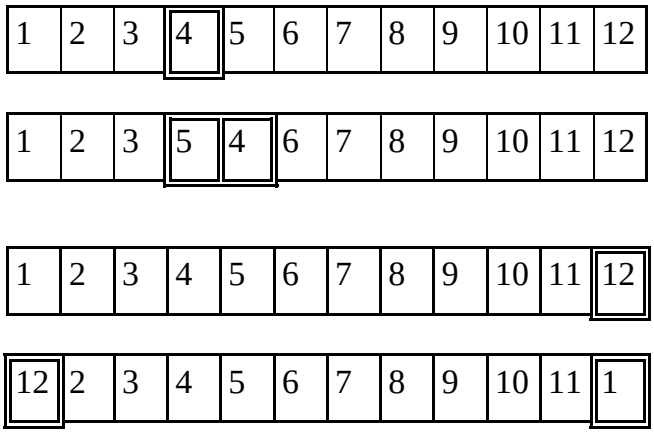
			4	5	6	7					

1	11	12	4	5	6	7	2	10	9	8	3

- Mutação Inversiva



- Variação de Mutação Inversiva



EC6 – Implementar o algoritmo genético modificado para o problema do caixeiro viajante. A posição das cidades será indicada por meio de um arquivo contendo linhas do seguinte tipo:

N – número de cidades no arquivo

(x,y) – coordenadas da cidade no mapa

O custo de deslocamento entre duas cidades é dado pela distância entre as duas cidades. Os operadores e mecanismos de seleção a serem utilizados podem ser quaisquer.

11 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.
- 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 é 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. Numa população de tamanho *n*, entre 2^m e $n \cdot 2^n$ diferentes esquemas podem ser representados.
- 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*. Assuma 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 conseqüência imediata deste teorema é que os algoritmos genéticos tendem a explorar o espaço por meio de esquemas curtos e de baixa ordem que, subseqüentemente, 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 ε é a probabilidade de um esquema ser rompido por *crossover*.
- note entretanto que, em alguns problemas, alguns blocos construtivos (esquemas curtos, de ordem baixa) podem direcionar erroneamente o algoritmo, levando-o a convergir a pontos sub-ótimos. Este fenômeno é conhecido como **decepção**. Assim, a hipótese dos blocos construtivos não fornece uma explicação definitiva do porquê os algoritmos genéticos funcionam. Ela é apenas uma indicação do porquê os algoritmos genéticos funcionam para uma certa classe de problemas.

11.1 Deception Problem

- alguns blocos construtivos podem direcionar erroneamante o GA, levando a uma convergência a pontos sub-ótimos:
  - $\langle 1\ 1\ 1\ *\ *\ *\ *\ *\ *\ *\rangle$ – fitness *acima* da média
  - $\langle *\ *\ *\ *\ *\ *\ *\ *\ 1\ 1\rangle$ – fitness *acima* da média
  - $\langle 1\ 1\ 1\ *\ *\ *\ *\ *\ 1\ 1\rangle$ – fitness *muito menor* que $\langle 0\ 0\ 0\ *\ *\ *\ *\ *\ 0\ 0\rangle$
  - solução ótima – $\langle 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\rangle$
  - tendência a convergir para pontos como $\langle 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\rangle$

- algumas alternativas foram propostas para combater o problema da decepção (GOLDBERG, 1989; MICHALEWICZ, 1996). A primeira assume 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 é a utilização de um operador de *inversão*: selecionam-se 2 pontos em um cromossomo e inverte-se a ordem dos bits entre os pontos selecionados (alterando-se a codificação em conjunto com a operação).
- a terceira opção é utilizar algoritmos genéticos *messy*, que diferem do algoritmo genético clássico de várias maneiras: codificação, operadores, presença de cromossomos de tamanho distinto e fases evolutivas.

12 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.

BERTONI, A. & DORIGO, M. "Implicit Parallelism in Genetic Algorithms", *Artificial Intelligence*, 61(2): 307-314, 1993.

ESHELMAN, L. J., CARUANA, R. A. & SCHAFFER, J. D. "Biases in the Crossover Landscape", *in* Schaffer, J. (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, 10-19, 1989.

FOGEL, D. B. "An Introduction to Simulated Evolutionary Computation", *IEEE Transactions on Neural Networks*, 5(1): 3-14, 1994.

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", University of Michigan Press, 1975.

HOLLAND, J.H. "Adaptation in Natural and Artificial Systems", 2nd edition, The MIT Press, 1992.

IYODA, E.M. "Inteligência Computacional no Projeto Automático de Redes Neurais Híbridas e Redes Neurofuzzy Heterogêneas", *Tese de Mestrado*, Faculdade de Engenharia Elétrica e de Computação (Unicamp), 2000.

MICHALEWICZ, Z. "Genetic Algorithms + Data Structures = Evolution Programs", 3rd edition, Springer, 1996.

MICHALEWICZ, Z. & SCHOENAUER, M. "Evolutionary Algorithms for Constrained Parameter Optimization Problems", *Evolutionary Computation*, 4(1): 1-32, 1996.

SYSWERDA, G. "Uniform Crossover in Genetic Algorithms", *in* Schaffer, J.D. (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pp. 2-9, 1989.