INTELIGÊNCIA ARTIFICIAL APLICADA

AULA 6

Prof. Luciano Frontino de Medeiros

UNINTER
CENTRO UNIVERSITÁRIO INTERNACIONAL

CONVERSA INICIAL

Nesta aula você estudará sobre os Algoritmos Genéticos, os conceitos, características, etapas do algoritmo, bem como o exemplo de aplicação de AG, mostrando em detalhes a sua dinâmica, e a aplicação desse exemplo utilizando linguagem de programação.

CONTEXTUALIZANDO

A teoria da evolução de Darwin foi um dos conceitos que revolucionaram a ciência no século XIX. Em meados do século XX, a ideia de evolução permitiu a concepção de sistemas que se modificam de acordo com um conjunto de operações específicas, na busca da solução de um problema. Em vez de solucionar um problema de busca por meio de procedimentos analíticos, pode-se começar com uma solução parcial e, ao longo de várias iterações, fazer evoluir essa solução até chegar a uma boa o bastante. Um dos representantes da linha de pesquisa evolucionária são os Algoritmos Genéticos, que permitem a abordagem de um problema, considerando a evolução de soluções parciais. Geralmente são aplicados em problemas dos quais não se tem muito, ou mesmo nenhum conhecimento. Estudaremos agora alguns dos conceitos importantes dessa técnica relevante de IA.

TEMA 1: INTRODUÇÃO A ALGORITMOS GENÉTICOS

As pesquisas de IA, dentro da linha evolucionária, baseiam-se na observação de mecanismos evolutivos da natureza, incluindo auto-organização e comportamento adaptativo. Os modelos mais conhecidos de algoritmos evolucionários são os algoritmos genéticos, a programação genética e as estratégias evolucionárias.

Um algoritmo genético (AG) faz parte da classe de algoritmos de busca. O algoritmo procura uma solução dentro de um espaço para um problema de **otimização**. Assim, os algoritmos genéticos podem ser uma boa opção para

efetuar a busca em problemas considerados intratáveis. A aplicação de algoritmos genéticos se faz em várias áreas, tais como a arquitetura de circuitos eletrônicos, planejamento e roteirização, programação de games, previsão do tempo e ainda descoberta de identidades matemáticas (RUSSEL; NORVIG, 2004).

Podemos afirmar que um AG é considerado um algoritmo de busca em feixe estocástica, em que os estados sucessores são criados a partir da combinação de dois (ou mais) estados “pais”, em vez de serem criados a partir da variação de um único estado. Temos assim uma analogia com a seleção natural proveniente da biologia. A busca em feixe se dá pelo uso de uma população inicial gerada aleatoriamente, que evolui ao longo das “gerações” (iterações do algoritmo). A produção de uma nova população é avaliada por uma “**função objetivo”**, ou “**função de *fitness*”**. Essa função retorna à nova população, priorizando os estados melhores (RUSSEL; NORVIG, 2004, p. 115).

Dos principais fatores que têm feito dos AG uma técnica de busca bem-sucedida e utilizada, estão (AZEVEDO, 2000, p. 35):

- Possui operacionalização simples;
- Baixa dificuldade de implementação;
- Apresenta alta eficácia com relação ao processo de busca na região onde provavelmente se encontra o valor máximo global;
- Útil quando se tem pouco ou mesmo nenhum conhecimento do problema, ou ainda quando tal conhecimento é impreciso.

Pelas suas características, os AGs fazem parte dos métodos probabilísticos de busca e otimização. Os AGs utilizam o conceito de probabilidade, mas não são considerados simples buscas aleatórias. Ao contrário, eles direcionam a busca para regiões onde é mais provável

encontrar uma solução ótima. Diferente das técnicas de busca convencionais, as principais diferenças residem em (AZEVEDO, 2000, p. 37):

- "A busca da melhor solução é feita sobre uma população de pontos e não sobre um único ponto (busca em feixe)";
- "Os AGs perfazem uma busca cega, sendo a única exigência o conhecimento da função objetivo de cada indivíduo, não sendo necessária nenhuma informação adicional";
- "Os AGs usam operadores estocásticos ou probabilísticos e não regras determinísticas na direção de uma busca altamente exploratória e estruturada; as informações das gerações anteriores são acumuladas, auxiliando a direcionar estas buscas".

De forma geral, os passos para a execução de um AG estão descritos no Quadro 1.

Quadro 1: Passos genéricos para o Algoritmo Genético

1. Gera-se aleatoriamente uma população de candidatos a solução do problema. 2. Enquanto não satisfaz as condições de terminação (cada iteração e geração): (a) Gera-se uma nova população inicialmente vazia. (b) Enquanto a nova população não estiver completa: i. Executa a **seleção** de dois indivíduos aleatoriamente da população atual, dando preferência na seleção àqueles indivíduos que possuem maior ***fitness***. ii. Executa o **crossover** dos dois indivíduos, conforme um ponto de corte, para obter dois novos indivíduos.

(c) Concede a cada membro da nova população a chance da operação de **mutação**.

(d) Substitui a população atual com a nova população.

3. Escolhe o indivíduo da população com a melhor avaliação de ***fitness*** para ser a solução do problema.

Fonte: adaptado de GOMEZ (2005).

População é o conjunto de candidatos a soluções do problema em questão. Por meio das sucessivas gerações, novos candidatos são gerados na população, ao passo que outros desaparecem, ou seja, são descartados da população. Uma solução na população é denominada **indivíduo**. A adaptação ou adequação (***fitness***), relacionada a um indivíduo, significa a qualidade dessa medida representada por esse indivíduo. “Quanto melhor a solução, maior será a sua adaptação. É claro que isso depende das características do problema a ser resolvido” (GOMEZ, 2005).

Os indivíduos que fazem parte de um AG precisam de uma codificação que o simbolize no espaço possível de indivíduos que caracterizam o problema. Essa codificação é feita utilizando-se **cromossomos**. Um cromossomo é a tradução das características do indivíduo no **alfabeto** utilizado pelo AG. Podemos concluir que um cromossomo é uma sequência de bits, sendo constituídos por “**genes**” que se referem a cada bit (LINDEN, 2012, p. 65). Os algoritmos geralmente utilizam como alfabeto a codificação binária (0’s e 1’s) para sua representação. Um cromossomo terá também um comprimento fixo de bits ou genes.

Os operadores mais comuns utilizados nos AG para criar os novos indivíduos através das gerações são:

- Seleção;

- Cruzamento ou *crossover*;
- Mutação.

O operador de **seleção** é comparável ao que encontramos na natureza sobre a lei da seleção natural, relativo à sobrevivência daqueles que melhor se adaptam ao ambiente. Os indivíduos mais adaptados (ou seja, aqueles que apresentam melhor *fitness*) são selecionados para gerarem a descendência. O operador de seleção utiliza um critério de maior probabilidade de cruzamento, dando prioridade para aqueles que possuem maior *fitness*. (GOMEZ, 2005).

Um dos métodos mais utilizados para perfazer a seleção é o **método da roleta viciada** ou **roleta ponderada**. Esse método dá mais probabilidade para que um indivíduo de maior *fitness* seja escolhido em contrapartida a outro elemento que tenha menor *fitness*. Isso não quer dizer que os indivíduos que possuam menor *fitness* não passem para a próxima geração, mas que a probabilidade de fazerem parte dessa nova geração é reduzida.

O ***crossover*** ou cruzamento ocorre pela mistura de duas soluções ou **indivíduos**, com o objetivo de criar dois novos indivíduos. Esse cruzamento tende a formar novos indivíduos, que possuem características dos “pais”, e que têm a possibilidade de atender melhor o *fitness*. O ***crossover*** atua considerando dois indivíduos pais, assumindo um ponto de corte em seus cromossomos e intercalando as partes, resultando novos indivíduos. AGs podem ter pontos de corte fixos ou aleatórios ao longo das gerações.

Durante cada geração, existe uma pequena chance de que um indivíduo dentro da população sofra uma **mutação**, que vai mudar a característica do indivíduo levemente. A mutação embute no AG uma característica totalmente aleatória. Conforme uma taxa especificada no algoritmo, alguns genes dentro dos cromossomos são escolhidos de forma randômica e alterados para outros valores permitidos pelo alfabeto do cromossomo.

Uma variável relevante para o funcionamento do AG é o **tamanho da população**. O aumento dos indivíduos em uma população permitirá o alcance de um número maior de soluções possíveis, implicando assim na variabilidade

de alternativas oferecidas pelo AG. Isso permitirá que as melhores soluções, que estejam próximas de uma solução ótima (se esta existir) sejam criadas. Portanto, o tamanho da população deve ser grande, levando-se em conta as restrições físicas (GOMEZ, 2005). O que limita o tamanho da população é o **tempo** que vai demorar o AG na sua execução, e a quantidade de **memória** para guardar a população.

Qual a melhor forma de terminar a execução do AG? A abordagem mais simples é rodar o algoritmo de busca por um **número fixo de gerações** – quanto mais gerações, melhor. Outra abordagem é encerrar o algoritmo se, depois de passadas algumas gerações, não se obtiver uma melhora na adaptação do melhor indivíduo da população. Ou seja, a variação dos melhores indivíduos obtidos para os que são criados novamente é mínima. Dessa forma, adotar um **limiar de variação** pode reduzir o número de gerações na execução de um AG.

TEMA 2: EXEMPLO DE APLICAÇÃO DE AG

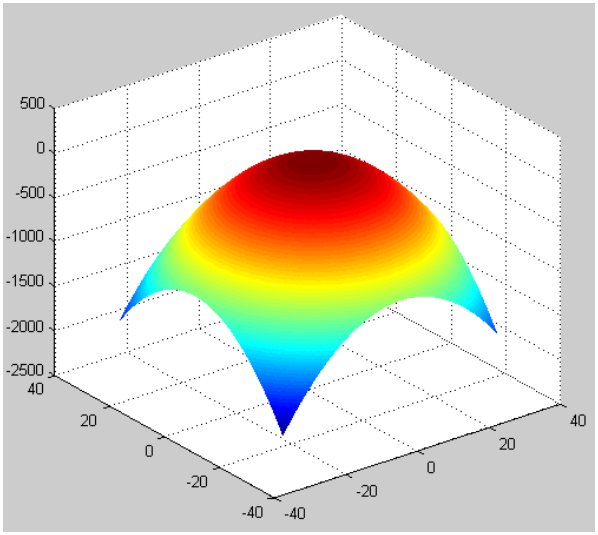
Bons exemplos para entender a execução de um AG são aqueles nos quais se conhece qual a solução ótima ou ideal. Problemas matemáticos que possuem métodos determinísticos de resolução com poucas variáveis são perfeitos para o entendimento de um AG. Entretanto, é importante ressaltar que a maioria dos problemas reais contém uma diversidade muito grande de variáveis e são intratáveis, cenário no qual o uso de AG se faz realmente robusto.

O exemplo de otimização que utilizaremos será encontrar o máximo de uma equação quadrática de duas variáveis. A solução para este problema pode ser encontrada facilmente pelo cálculo diferencial da matemática. Vamos utilizar a seguinte equação:

$$f(x) = 2 - (x-3)^2 - (y-2)^2$$

Sabemos de antemão que o valor máximo dessa equação se encontra no ponto *x=3* e *y=2*, com *f(x)=2*. Podemos ver na Figura 1 o gráfico dessa equação, mostrando visualmente o ponto de máximo que se pode obter.

Figura 1: Representação no plano espacial da equação $f(x) = 2 - (x-3)^2 - (y-2)^2$, com o ponto de máximo em *x=3* e *y=2*, com *f(x)=2*.



Detalharemos então o exemplo nos passos seguintes:

a) Criaremos um AG que será capaz de obter esse ponto de máximo como solução, sendo a expressão em f(x) a própria **função objetivo** ou ***fitness***.

b) Precisamos definir o cromossomo com o qual iremos gerar os indivíduos para o AG. Para definir este cromossomo, devemos determinar qual a faixa de valores que as variáveis x e y irão assumir. Vamos supor que as variáveis x e y devam ficar no intervalo **[0,7]**. Ou seja, o valor mínimo que elas podem assumir é 0 e o máximo será 7. Podemos utilizar a codificação binária para representar os cromossomos. Lembrando a conversão de números decimais para binários:

Decimal	Binário
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Notamos que, naturalmente, a conversão de decimal para binário nos dá o que precisaremos. Vamos assumir agora que, para cada variável, teremos um comprimento de 3 **genes**. Com as duas variáveis, teremos, portanto, um **cromossomo**, composto de dois alelos com um total de 6 **genes**. Por exemplo, o cromossomo para x=1 (001b conforme a tabela) e y=5 (101b) será:

001101

c) Após a definição do cromossomo, vamos supor uma população inicial de 10 indivíduos. Assim, cada indivíduo é uma combinação de dois valores: x e y. Podemos começar com a seguinte população escolhida aleatoriamente:

x	y	Cromossomo
3	6	011110
6	4	110100
3	5	011101
7	0	111000
3	7	011111
1	6	001110
1	2	001010
5	4	101100
6	1	110001

4	2	100010

d) Avaliaremos os indivíduos gerados aleatoriamente a partir da função objetivo. Adicionaremos mais uma coluna na tabela anterior, indicando o valor obtido pela função objetivo, substituindo os valores de x e y:

x	*y*	*Cromossomo*	*(x,y)*
3	6	011110	8
6	4	110100	15
3	5	011101	3
7	0	111000	32
3	7	011111	5
1	6	001110	8
1	2	001010	0
5	4	101100	8
6	1	110001	18
4	2	100010	3

e) O operador de seleção deve ser executado. Assim, organizaremos em ordem descendente os indivíduos, de acordo com a função de *fitness*. Notamos também que nessa primeira população, o indivíduo com *fitness* 0 (zero) é o mais próximo da solução para o problema:

x	y	Cromossomo	(x,y)
1	2	001010	0
3	5	011101	3
4	2	100010	3
3	6	011110	8
1	6	001110	8
5	4	101100	8
6	4	110100	15
3	7	011111	15

6	1	110001	18
7	0	111000	32

f) Escolhemos agora um ponto de corte da população. Adotaremos como critério a seleção dos primeiros quatro indivíduos para a próxima população a ser gerada.

x	y	Cromossomo	(x,y)
1	2	001010	0
3	5	011101	3
4	2	100010	3
3	6	011110	8
1	6	001110	8
5	4	101100	8
6	4	110100	15
3	7	011111	15
6	1	110001	18
7	0	111000	32

g) Simularemos agora o método da roleta viciada, fazendo com que o primeiro indivíduo da lista seja escolhido para gerar 4 indivíduos para a nova população; o segundo 3 indivíduos; o terceiro 2 indivíduos e o quarto 1 indivíduo (veja a coluna “ponderação”). Assim, a nova população terá também 10 indivíduos. No passo de cruzamento (*crossover*), escolhemos aleatoriamente um ponto de corte (no caso, o quarto gene). Os cromossomos ficam, então, divididos para a execução do *crossover* (com os genes ilustrado em duas cores):

x	y	Cromossomo	(x,y)	Ponderação
1	2	001010	0	4
3	5	011101	3	3
4	2	100010	3	2

3	6	011110	8	1

h) O próximo passo gera a nova população, sendo cada novo indivíduo combinado a partir de dois indivíduos escolhidos aleatoriamente. Note que só preencheremos a coluna referente aos cromossomos:

x	y	Cromossomo	(x,y)
		001001	
		001001	
		001010	
		001010	
		011110	
		011110	
		011101	
		100010	
		100010	
		011110	

i) Agora, executaremos o operador **mutação**. Vamos estipular uma taxa de mutação de 1 gene para cada 12 genes do total da população. Como temos 60 genes ao total, podemos modificar aleatoriamente 5 genes nos indivíduos da população gerada pelo *crossover* (genes alterados na cor verde). Assim:

x	y	Cromossomo	(x,y)
		001011	
		001001	
		011010	
		001010	
		011110	
		011111	
		011101	
		101010	

		100010	
		011010	

j) O próximo passo é decodificar o cromossomo na representação binária para os valores das variáveis novamente:

x	y	Cromossomo	(x,y)
1	3	001011	
1	1	001001	
3	2	011010	
1	2	001010	
3	6	011110	
3	7	011111	
3	5	011101	
5	2	101010	
4	2	100010	
3	2	011010	

k) Finalmente, avaliamos a função objetivo para estes novos valores obtidos:

x	y	Cromossomo	(x,y)
1	3	001011	1
1	1	001001	3
3	2	011010	0
1	2	001010	0
3	6	011110	8
3	7	011111	15
3	5	011101	3
5	2	101010	8
4	2	100010	3
3	2	011010	0

Temos assim uma nova população. Podemos notar que nesta geração, o valor máximo obtido foi **1**, relativo ao ponto *x=1* e *y=3*. Para as próximas gerações, o algoritmo genético deve ser executado a partir do passo e), continuando até obter o valor máximo da função objetivo considerada.

Esse exemplo demonstra a dinâmica do AG de uma forma simplificada, porém permitindo visualizar a aplicação dos operadores de seleção, cruzamento e mutação. A faixa de valores foi tratada com números inteiros, mas poderíamos adotar números reais. Por exemplo, um cromossomo com tamanho total 30 (15 bits para cada variável), com o alfabeto binário, os pontos tomados ao acaso *x=3,07* e *y=2,13*, teriam a seguinte representação codificada em binário para o cromossomo:

100010000111010110001001100100

Os 15 primeiros bits se referem ao valor de x e os 15 últimos ao valor de y. Ao avaliarmos pela função de *fitness* esses valores de x e y, obtemos o valor de *f(x) = 1,9769*.

TEMA 3: AG EM JAVA

Para a abordagem desse mesmo problema, utilizando linguagem de programação, utilizaremos a biblioteca Java API for Genetic Algorithms (JAGA). Essa biblioteca consiste em uma série de classes para o uso em diversos tipos de problemas. Para exemplos que lidem com aspectos comuns dos AG, podemos criar uma classe de exemplo com o problema em questão. Outra classe deve ser criada para definir a função objetivo.

Para saber mais... A biblioteca JAGA está disponível para download neste endereço: <http://jaga-java-api-for-genetic-algorithms.soft112.com/download.html>, ou por meio do QR Code ao lado.	

Na Figura 2, a seguir, temos a classe Example1.java, contendo o exemplo da maximização da função objetivo anterior. O código está comentado, facilitando o entendimento. O tamanho da população foi definido em 40 indivíduos, a taxa de *crossover* em 90%, a taxa de mutação em 2%, executando 10000 gerações. Os indivíduos utilizam duas variáveis, com precisão de duas casas decimais, operando na faixa [-6,6], codificados com 30 bits (15 para cada variável).

Figura 2: Código da classe Example1.java, mostrando a configuração do problema de maximização da função objetivo do exemplo anterior

```
public class Example1 {

    public Example1() {
    }

    public void exec() {
        // Define os parâmetros para o AG
        GAParameterSet params = new
DefaultParameterSet();
        params.setPopulationSize(40);
        params.setFitnessEvaluationAlgorithm(new
Example1Fitness());
```

```
                    // Inclui a reprodução da nova população com
crossover e mutação
                    CombinedReproductionAlgorithm repAlg = new
CombinedReproductionAlgorithm();
                    repAlg.insertReproductionAlgorithm(0, new
SimpleBinaryXOver(0.9));
                    repAlg.insertReproductionAlgorithm(1, new
SimpleBinaryMutation(0.02));
                    params.setReproductionAlgorithm(repAlg);

                    // Define o método da roleta viciada
                    params.setSelectionAlgorithm(new
RouletteWheelSelection(-10E10));
                    // Número máximo de gerações
                    params.setMaxGenerationNumber(10000);
                    // Define as variáveis por indivíduo, a precisão decimal
e o tamanho do cromossomo
                    NDecimalsIndividualSimpleFactory fact = new
NDecimalsIndividualSimpleFactory(2, 2, 15);
                    fact.setConstraint(0, new RangeConstraint(-6, 6));
                    fact.setConstraint(1, new RangeConstraint(-6, 6));
                    params.setIndividualsFactory(fact);

                    // Constrói o AG
                    ReusableSimpleGA ga = new
ReusableSimpleGA(params);
                    // Associa o analisador
                    AnalysisHook hook = new AnalysisHook(System.out,
false);
                    ga.addHook(hook);
```

```
                final int attempts = 1;

                // Executa o AG
                GAResult [] allResults = new GAResult[attempts];
                for (int i = 0; i < attempts; i++) {
                        hook.reset();
                        GAResult result = ((ReusableSimpleGA)
ga).exec();
                        allResults[i] = result;
                }
                System.out.println("\nALL DONE.\n");
                for (int i = 0; i < attempts; i++) {
                        System.out.println("Result " + i + " is: " +
allResults[i]);
                }

        }

        public static void main(String[] unusedArgs) {
                // Chamada da classe para execução do AG
                Example1 demo = new Example1();
                demo.exec();
        }
}
```

Figura 3: Classe Example1*Fitness*.java, que contém o método responsável pela avaliação da função objetivo

```
public class Example1Fitness implements FitnessEvaluationAlgorithm {

    public Example1Fitness() {}

    public Class getApplicableClass() {
        return NDecimalsIndividual.class;
    }

    public Fitness evaluateFitness(Individual individual, int age, Population population, GAParameterSet params) {
        NDecimalsIndividual indiv = (NDecimalsIndividual) individual;
        double x = indiv.getDoubleValue(0);
        double y = indiv.getDoubleValue(1);
        double f = 2 - (x-3)*(x-3) - (y-2)*(y-2);
        Fitness fit = new AbsoluteFitness(f);
        return fit;
    }
}
```

A classe que contém o método que avalia pela função objetivo (*evaluateFitness*) está descrita na Figura 3. Na Figura 4 temos a tela de saída de uma execução da classe Example1.java. Pode-se notar que após 10000 gerações, foi encontrado o melhor resultado para x=2.97 e y=1.94, resultando na função objetivo f(x) = 1.9955. Esse melhor resultado foi encontrado na geração 3779. Na Figura 5 estão listadas algumas execuções deste AG, mostrando as diferentes soluções alcançadas. Diferentes parâmetros podem ser testados para

verificar se o AG pode encontrar uma solução mais próxima da calculada deterministicamente.

Figura 4: Tela de saída de uma execução da classe Example1.java, mostrando que, após 10000 gerações, foi encontrado o melhor resultado para os valores x=2.97 e y=1.94, resultando a função objetivo f(x) = 1.9955

```
*** Computation completed.
    Generations: 10000
    Fitness evaluations: 400040
    Run time: 00:04.4099
    Best result: {size=2; repr=00000011011110100000010100011;
vals=(2.97, 1.94); fit=1.9955}
*** Best fitness (1.9955) was discovered in generation 3779.

ALL DONE.

Result 0 is: {size=2; repr=00000011011110100000010100011;
vals=(2.97, 1.94); fit=1.9955}
```

Figura 5: Dez execuções do AG do exemplo, mostrando as melhores soluções alcançadas em cada uma delas

No.	*x*	*Y*	*f(x,y)*	*Geração*
1	3.00	1.99	1.9999	2475
2	2.99	2.01	1.9998	9816
3	3.14	2.06	1.9768	4019
4	3.00	1.98	1.9996	6431
5	3.01	2.00	1.9999	306
6	3.00	1.99	1.9999	1438

7	2.97	2.04	1.9975	1259
8	3.00	2.01	1.9999	2102
9	2.99	2.00	1.9999	9165
10	2.07	2.01	1.9990	2341

SÍNTESE

Nesta aula foi estudada uma introdução aos Algoritmos Genéticos, que fazem parte da linha de pesquisa evolucionária. Um AG faz parte da classe de algoritmos de busca, indo atrás de uma solução dentro de um espaço para um problema específico de otimização.

Os algoritmos genéticos podem ser uma boa opção para efetuar a busca em problemas considerados intratáveis. A busca em feixe estocástico se deve ao uso de uma população inicial gerada aleatoriamente, que evolui ao longo das iterações do algoritmo. A produção de uma nova população é avaliada por uma função objetivo ou de *fitness*. AG é uma técnica bem-sucedida, simples de operar, de fácil implementação, eficaz na busca da região onde provavelmente se encontra o máximo global e aplicável em situações em que se tem pouco ou nenhum conhecimento do modelo ou, ainda, esse é impreciso. Um algoritmo genético possui como principais operadores a seleção, o cruzamento ou *crossover* e a mutação. Um indivíduo que é uma solução em um AG é codificado por meio de um cromossomo, conforme o alfabeto utilizado pelo AG.

REFERÊNCIAS

RUSSEL, S.; NORVIG, P. **Inteligência Artificial**. Tradução da 2. ed. Rio de Janeiro: Campus, 2004.

AZEVEDO, F. M.; BRASIL, L. M.; OLIVEIRA, R. C. L. **Redes Neurais com Aplicações em Controle e em Sistemas Especialistas**. Florianópolis: Visualbooks, 2000.

LINDEN, R. **Algoritmos Genéticos**. Rio de Janeiro: Ciência Moderna, 2012.

HAYKIN, S. **Redes Neurais**: Princípios e prática. Porto Alegre, Bookman, 2001.

KOZA, J. **Genetic Programming**. Boston-MA: MIT Press, 1992.

GOMEZ, L. A. **Uma Introdução aos Algoritmos Genéticos**. Computação Evolutiva e Engenharia do Conhecimento, 2005. Disponível em: <http://www.labeee.ufsc.br/~luis/ga/GA-INTRO.pdf>. Acesso em: 01/04/2017.