Algoritmos genéticos: aplicando a teoria a um estudo de caso

Genetic algorithms: applying theory to a case study

DOI:10.34117/bjdv7n3-016

Recebimento dos originais: 08/02/2021
Aceitação para publicação: 01/03/2021

Luis Borges Gouveia
Doutor em Ciências da Computação pela University of Lancaster (Reino Unido)
Instituição: Universidade Fernando Pessoa (Portugal)
Endereço: Praça 9 de Abril. Porto. Portugal
E-mail: lmbg@ufp.edu.pt

Andréa Cristina Marques de Araújo
Doutora em Ciência da Informação - Universidade Fernando Pessoa (Portugal)
Mestre em Ciência da Computação – UFSC
Instituição: Centro Universitário do Estado do Pará CESUPA
Endereço: Av. Gov. José Malcher n.1963 Belém-PA
E-mail: andreacristinamaraujo@gmail.com

Jacqueline de Fátima Teixeira
Mestre em Ciência da Computação – UFSC
Rua Rosemeri Silva Belto, 319 – Neves Ponta Grossa-PR
E-mail jackogut@hotmail.com

Guilherme Teixeira Santos
Graduando de Física – Bacharelado
Universidade Estadual de Ponta Grossa - PR
Campus Uvaranas – Av Gal Carlos Cavalcante, 4748 – Ponta Grossa PR
E-mail g.teixeira3399@gmail.com

RESUMO
Este artigo trata dos princípios de Algoritmos Genéticos aplicados a um estudo de caso, implementado em linguagem de programação Amzi! Prolog 3.3. Serão comentadas todas as etapas de desenvolvimento do referido programa, relacionando a fundamentação teórica à solução implementada.

Palavras chaves: Computação Evolucionária, Algoritmos Genéticos, Otimização, Operadores Genéticos, Programação Prolog.

ABSTRACT
This paper is about the principles of Genetic Algorithms applied to a case study, implemented in the programming language Amzi! Prolog 3.3 programming language. All the development stages of this program will be commented, relating the theoretical basis to the implemented solution.

Key words: Evolutionary Computation, Genetic Algorithms, Optimization, Genetic Operators, Prolog Programming.

1 INTRODUÇÃO

Certos tipos de problemas não podem ser resolvidos de forma computacional tradicional e determinística, através de métodos de resolução exatos, principalmente aqueles que envolvem processos de otimização global, síntese de um objeto (programa de computador, circuito eletrônico, etc) ou a busca de um modelo que reproduza o comportamento de determinado fenômeno (*machine learning*). Para vários desses problemas é freqüentemente possível encontrar um algoritmo que ofereça uma solução ótima ou aproximadamente ótima. Alguns desses algoritmos requerem informação auxiliar (derivadas no caso de técnicas de gradiente), que muitas vezes não estão disponíveis ou então são difíceis de serem obtidas.

Um dos métodos heurísticos para resolução dessa categoria de problemas é o foco da área de Inteligência Artificial, denominada Computação Evolucionária (CE) que dispensa essas informações auxiliares e oferece algoritmos gerais, dentre estes os Algoritmos Genéticos (AG), que são aplicados em problemas complexos, com grandes espaços de busca e difícil modelagem.

A Computação Evolucionária constitui-se portanto, como um ramo da ciência da computação que não exige o conhecimento de uma sistemática prévia de resolução, envolvendo tópicos de vida artificial, geometria fractal, sistemas complexos e inteligência computacional.

Os algoritmos genéticos propostos por Holland, fazem parte de um conjunto de métodos computacionais inspirados na teoria biológica da evolução de Darwin, definida por Borges (2000) como um conjunto de métodos de IA que simulam a evolução de estruturas individuais através de processos de seleção, mutação e reprodução, os quais dependem do desempenho de cada indivíduo no meio ambiente em que se encontra. Trata-se de algoritmos probabilísticos que fornecem métodos generalizados de otimização e busca paralela e adaptativa que simulam os processos naturais da evolução por vários ciclos.

Segundo Barreto (1999), a motivação para considerar a inspiração biológica da evolução das espécies, proposta por Darwin no século XIX, é o fato de acreditar-se que a Natureza utilizou mecanismos para resolver o problema da busca do ótimo. Considere que cada indivíduo de uma determinada população possui um certo grau de aptidão ao ambiente em que vive, como os indivíduos de maior aptidão tendem a se reproduzir com maior facilidade, através do uso de mecanismos para recombinação de suas próprias características, após um determinado número de gerações, é possível obter indivíduos

com altos valores de aptidão. A herança biológica transmitida através de gerações seria mantida nos genes de cada indivíduo, na sua cadeia de cromossomos.

Na Natureza, portanto, o processo de otimização busca a sobrevivência, utilizando-se de processos extremamente complexos para seleção dos mais aptos. Os algoritmos genéticos imitam esses princípios com os chamados operadores genéticos.

2 A LINGUAGEM DE PROGRAMAÇÃO UTILIZADA: PROLOG

A denominação dessa linguagem de programação vem de PROgramming in LOGic. Prolog é uma linguagem que utiliza um pequeno conjunto de mecanismos básicos para criar surpreendentes e poderosos programas. Estes mecanismos são máquinas padronizadas em *tree based data structuring* e *backtracking*

Programas Prolog são interpretados por um interpretador Prolog. A principal vantagem de ter um interpretador rodando um programa Prolog é que o interpretador atual pode ser substituído por uma versão mais atualizada e aperfeiçoada sem afetar o programa Prolog instalado.

Um programa Prolog consiste em fatos e regras. O código Prolog é escrito em forma declarativa. Genericamente falando, um programa Prolog descreve um problema ao invés da solução de um problema. Isto é, diferente de linguagens imperativas como C, C++ e Java em que um programa define uma série de passos que devem ser feitos. Linguagens de programação lógica como o Prolog são inerentemente de *high-level* porque elas focam a lógica computacional e não os mecanismos para implementá-la.

Segundo Barreto (1999), Prolog tem uma semântica declarativa que escreve e lê o programa constituindo-se como uma representação do que se conhece da definição do problema a resolver, e outra semântica operacional, que define os passos do Prolog para resolver o problema. Prolog é essencialmente um transformador de semânticas declarativas para operacionais.

Trata-se portanto, de uma linguagem poderosa que pode tratar problemas complexos em programas compactos. Programas escritos em Prolog, normalmente são pequenos e mais curtos comparados com aqueles escritos em outras linguagens, sendo portanto, usada para desenvolvimento de sistemas de planejamento, sistemas de conhecimento, sistemas especialistas e a muitos outros propósitos, dentre esses os Algoritmos Genéticos.

3 ALGORITMOS GENÉTICOS

3.1 FUNDAMENTAÇÃO

Até meados do século XIX, os naturalistas acreditavam que cada espécie havia sido criada separadamente por um Ser Supremo, ou através de geração espontânea. O trabalho do naturalista Carolus Linnaeus sobre a classificação biológica de organismos despertou o interesse pela similaridade entre certas espécies, levando a acreditar na existência de uma certa relação entre elas.

No entanto, somente depois de mais de vinte anos de observações e experimentos, Charles Darwin apresentou em 1858 sua teoria de evolução através de seleção natural simultaneamente com outro naturalista inglês Alfred Russel Wallace. Estes estudos demonstraram a existência de inúmeras variações apresentadas em indivíduos de cada espécie, as quais representam as características diferentes que determinam a condição de adaptação e sobrevivência do indivíduo no meio ambiente.

Por volta de 1900, o trabalho de Gregor Mendel, desenvolvido em 1865, sobre os princípios básicos de herança genética, foi redescoberto pelos cientistas e teve grande influência sobre os futuros trabalhos relacionados a evolução. A moderna teoria da evolução combina a genética e as idéias de Darwin e Wallace sobre a seleção natural, criando o princípio básico de Genética Populacional: a variabilidade entre indivíduos em uma população de organismos que se reproduzem sexualmente é produzida pela mutação e pela recombinação genética.

Todos os organismos vivos são formados por células. Cada organismo possui uma cadeia de cromossomos que representam strings de DNA e servem como modelo para seus descendentes. Um cromossomo é composto de genes sendo que cada gene possui sua posição especifica na cadeia do cromossomo, essa posição é denominada locus. Cada gene representa uma característica especifica do indivíduo e pode assumir um certo valor a um conjunto de valores específicos denominados alelos.

Relacionando essas definições ao AG, podemos dizer que o cromossomo equivale ao indivíduo, o qual é representado por uma string de comprimento finito. O gene corresponde ao bit e o locus a posição que o bit ocupa na string (indivíduo). O alelo refere-se ao conjunto de valores possíveis de serem atribuídos a um determinado bit (por exemplo, alfabeto binário).

Os AG foram inspirados na Teoria de Darwin sobre a evolução, sendo Jonh Holland responsável pelo desenvolvimento das primeiras pesquisas sobre simulações computacionais de sistemas genéticos. Em 1975, Holland publicou seu livro *Adaptation*

in Natural and Artificial Systems, hoje considerado a Bíblia dos AG. Desde então, estes algoritmos vêm sendo aplicados com sucesso nos mais diversos problemas de otimização e aprendizado de máquina.

3.2 PRESSUPOSTOS BÁSICOS

Os Algoritmos Genéticos diferem dos métodos tradicionais de busca e otimização, principalmente em quatro aspectos (BARRETO, 1999):

a) Trabalham com uma codificação do conjunto de parâmetros e não com os próprios parâmetros do problema;

b) Operam em uma população e não em pontos isolados, reduzindo o risco de busca;

c) Utilizam informações de custo ou recompensa (informações da função objetiva – *payoff*) e não derivadas ou outro conhecimento auxiliar;

d) Procedem a busca utilizando operadores escolásticos e não determinísticos.

O mecanismo é obtido a partir de uma população de indivíduos (soluções) representados por cromossomos (palavras binárias, vetores, matrizes, etc), cada um associado a uma aptidão (avaliação da solução no problema), que são submetidos a um processo de evolução (seleção, cruzamento e mutação) por várias gerações (ciclos), conforme ilustra a Figura 1.

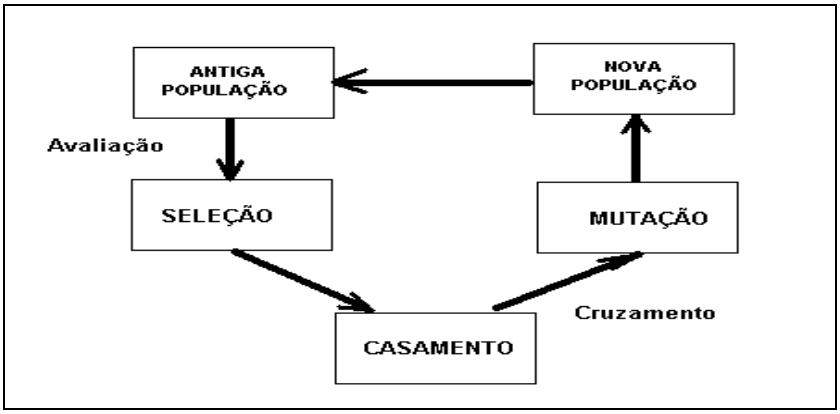


Figura1 – Mecanismo Básico de Funcionamento dos Algoritmos Genéticos.

Algoritmos Genéticos são muito eficientes para busca de soluções ótimas, ou aproximadamente ótimas (soluções boas), em uma grande variedade de problemas, pois não impõem muitas das limitações encontradas nos métodos de busca tradicionais. Um processo de otimização consiste em procurar melhorar a performance, com o objetivo de

alcançar um ou vários pontos ótimos. É desta forma que funcionam os Algoritmos Genéticos, combinando a sobrevivência do mais adaptado, com uma troca de informações ao mesmo tempo aleatória e estruturada.

Um AG pode convergir em busca de azar, porém sua utilização assegura que nenhum ponto do espaço de busca tem probabilidade zero de ser examinado. Toda tarefa de busca e otimização possui vários componentes, entre eles:

a) O espaço de busca onde são consideradas todas as probabilidades de solução de um determinado problema;

b) A função de avaliação (ou função de custo), que consiste numa maneira de avaliar os membros do espaço de busca.

As técnicas de busca e otimização tradicionais iniciam-se com um único candidato que, iterativamente, é manipulado utilizando algumas heurísticas (estáticas) diretamente associadas ao problema a ser solucionado. Por outro lado, as técnicas de computação evolucionária, entre elas a empregada nos algoritmos genéticos, operam sobre uma população de candidatos em paralelo. Assim, elas podem fazer a busca em diferentes áreas do espaço de solução, alocando um número de membros apropriado para a busca em várias regiões.

Segundo Obitko (1998), espaço de busca consiste no espaço onde estão todas as possíveis soluções de um problema. Quando nós estamos procurando resolver um problema, a idéia é achar a melhor solução dentro de um conjunto de possíveis soluções. Cada ponto no espaço de busca representa uma dessas possíveis soluções, a qual é atribuído valor (ou fitness) para o problema. Nós estamos procurando pela nossa solução, a qual é um ponto (ou mais) entre essas possíveis soluções. O processo de busca pela solução equivale à procura pelos extremos (mínimos ou máximos) no espaço de busca.

4 ETAPAS DE CONSTRUÇÃO DO AG DO ESTUDO DE CASO

O estudo de caso proposto refere-se a uma implementação utilizando a linguagem Prolog, para buscar elementos cujos cromossomos (codificados de forma binária) possuam o maior número de 1. Iremos então, conceituar e ilustrar os princípios fundamentais dos AG com trechos do programa implementado e resultados da experimentação realizada. No final deste trabalho apresentamos três anexos: um roteiro para auxiliar na execução do programa para experimentação (anexo 1), o código fonte do

programa ag11.pro (anexo 2) e finalmente o resultado comentado de duas experimentações realizadas (anexo 3).

Genericamente, um AG deve compor os seguintes passos:

Início
Repetir
Geração
Seleção
Crossover
Mutação
Sobrevivência
Até <número de iterações>
Fim

A sequência desses passos do modelo genérico está implementada da seguinte forma no estudo de caso:

```
% Procedimentos principais
geracoes(N,L,Ponto_Crossover,Pm,Gera,Max_gera,Lfitness,Fmedio,Melhores,Solucao):-
        sele_pares(N,Lfitness,Lpares),
        cruzamento(Ponto_Crossover,Lpares,Lcruza),
        mutacao(Pm,Lcruza,Lmuta),
        imprime(Gera,Lfitness,Fmedio,Melhores,Lpares,Lcruza,Lmuta),
        Gera1 is Gera+1,
        adapta(Lmuta,Nova_Lfitness),
        fitness_medio(N,Nova_Lfitness,Novo_Fmedio),
        melhores(Nova_Lfitness,fit(_,0),Melhores_gera,Melhores,Melhores_novos),
        elimina_repetidos(Melhores_novos,Mel),
        teste_convergencia1(L,Nova_Lfitness,Mel),!,
        teste_convergencia2(Gera1,Max_gera,Mel),!,
        geracoes(N,L,Ponto_Crossover,Pm,Gera1,Max_gera,Nova_Lfitness,Novo_Fmedio,Mel,Solucao).
```

O modelo genérico apresentado acima deve ser responsável pela realização das seguintes funções:

1. inicializa randomicamente a população de cromossomas (possíveis soluções do problema);
2. avalia cada cromossoma da população conforme função de fitness definida;
3. cria novos cromossomos a partir da população atual – *New Population* (aplicando os operadores genéticos mutação e cruzamento, substituindo os ascendentes pelos descendentes);

4. finaliza, se o critério de fim for alcançado, se não, reinicializa – *Replace/Test e Loop*.

Um maior detalhamento dessas funções pode ser observado no fluxograma a seguir:

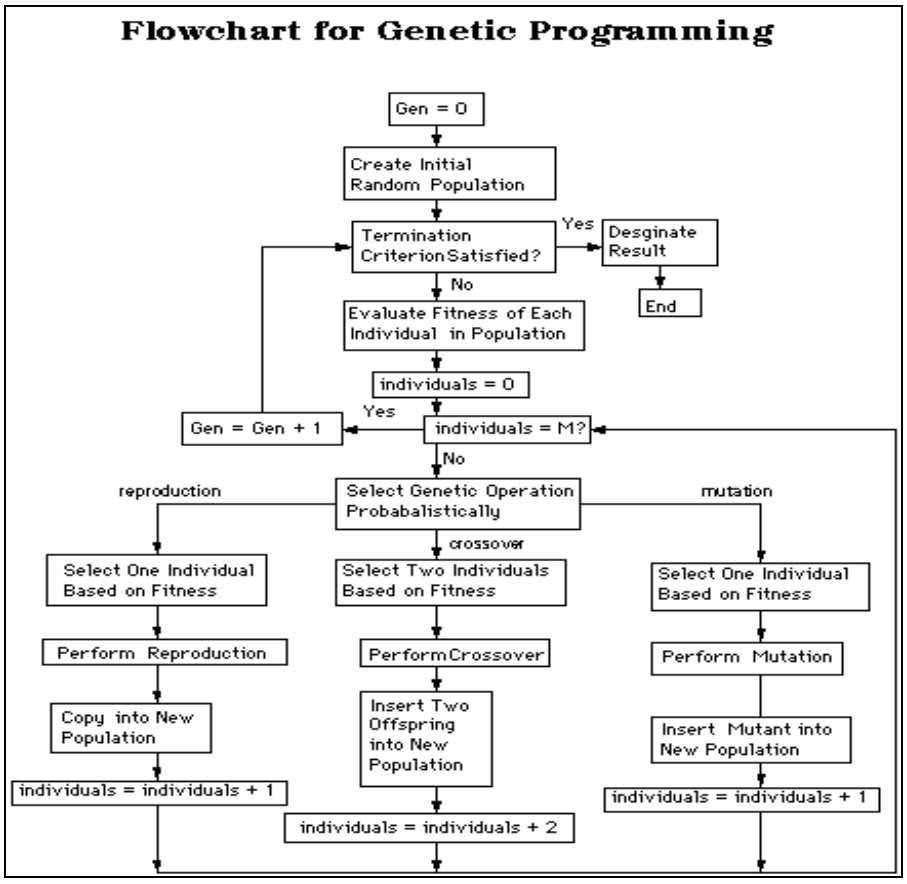


Figura.2- Fluxograma do Algoritmo Genético

4.1 ENCODING

A codificação do cromossomo é uma das grandes dificuldades encontradas quando estamos iniciando a solução do problema com algoritmos genéticos. Trata-se da representação destes problemas de maneira que os algoritmos genéticos possam trabalhar adequadamente sobre eles. A maioria das representações são genotípicas, pois utilizam vetores de tamanho finito em um alfabeto finito.

O cromossomo deve conter informações sobre a solução que ele deve representar, portanto a codificação varia de acordo com o problema em estudo.

O método de codificação mais utilizado é o código binário, mas é claro que existem muitas outras maneiras. A seguir comentaremos alguns desses métodos mais utilizados:

a) String Binário: conforme dito anteriormente, é o modo mais utilizado de codificação em AG, principalmente porque os primeiros estudos realizados neste campo foram feitos utilizando este método. Cada cromossomo tem a sua respectiva string binária (0 ou 1), conforme ilustrado na figura 3. Cada bit na string pode representar uma característica da solução ou o conjunto de string pode representar um número. Esse tipo de codificação nos fornece muitas possibilidades de cromossomos mesmo com um pequeno número de alelos. Em contraposição, constantemente não é comum a muitos tipos de problemas, podendo muitas vezes precisar de correções depois de realizadas as operações de crossover e/ou mutação.

Cromossomo A	10110010110010101110010 1
Cromossomo B	11111110000011000001111 1

Figura 3 – Cromossomos codificados na forma de String Binário.

A implementação do estudo de caso objeto deste trabalho, utilizou a codificação de string binária, como é apresentado destacado no trecho de programa abaixo:

```
%Geração da população inicial de N elementos com L bits cada
gera_pop_inic(1,L,[C1]):- gera(L,C1),!.
gera_pop_inic(N,L,[C1|Resto]):- gera(L,C1),
                                N1 is N-1,
                                gera_pop_inic(N1,L,Resto).
gera(1,[1]):- X is random,
              X>=0.5,!.
gera(1,[0]):- !.
gera(L,[X|Y]):- gera(1,[X]),
        L1 is L-1,
        gera(L1,Y).
```

b) Código de Permutação: este tipo de código é comumente usado em problemas de disposição e arrumação, como por exemplo, o problema do *traveling salesman* (caixeiro viajante), onde são definidas certas cidades e as distâncias entre elas, e o caixeiro deve visitar todas. O problema consiste em encontrar a melhor seqüência de visitas às cidades, percorrendo o menor percurso possível.

Na permutação, cada cromossomo corresponde a uma string de números onde cada número representa uma seqüência, conforme ilustrado na figura a seguir:

Cromossomo A	1 5 3 2 6 4 7 9 8
Cromossomo B	8 5 6 7 2 3 1 4 9

Figura 4 – Cromossomos codificados na forma de Permutação.

Mesmo para esses problemas, algumas correções no código também devem ser realizadas após as operações de mutação e crossover, visando deixar o cromossomo mais consistente, ou seja, com uma seqüência mais próxima da realidade.

c) Código de Valores: Valores diretos podem ser usados na codificação quando os problemas em questão trabalham com valores complicados, como números reais, pois a codificação binária nesses casos é um processo bastante complexo.

Neste método, cada cromossomo corresponde a uma string de valores característico do problema, por exemplo forma numérica, números reais, letras ou propriedades de objetos.

Cromossomo A	1.2324 5.3243 0.4556 2.3293 2.4545
Cromossomo B	ABDJEIFJDHDIERJFDLDFLFEGT
Cromossomo C	(back), (back), (right), (forward), (left)

Figura 5 – Cromossomos codificados na forma de Valores.

É importante ressaltar que para este tipo de codificação é necessário desenvolver operações de crossover e mutação específicas ao problema.

c) Código em Árvore: este código é utilizado principalmente em problemas que envolvem expressões e que englobam a programação genética. Cada cromossomo representa uma árvore de determinado objeto ou propósito equivalente às funções ou comandos da linguagem de programação. A figura a seguir representa claramente essa configuração.

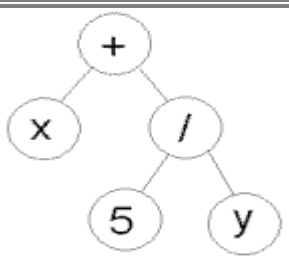
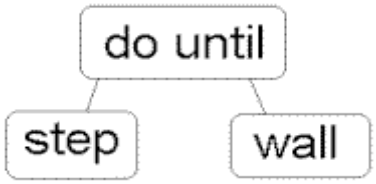
Cromossomo A	Cromossomo B
	
(+ x (/ 5 y))	(do_until step wall)

Figura 6 – Cromossomos codificados na forma de Árvore

Devido a sua forma de configuração em árvore, as operações de mutação e crossover podem ser efetuadas sem grandes complicações.

4.2 INÍCIO – ENTRADA DE PARÂMETROS

Nesta etapa de construção do algoritmo são definidos quais os valores das variáveis iniciais consideradas necessárias para a solução do problema, destacados no trecho a seguir:

```
% Procedimentos iniciais
ag(N,L,Ponto_Crossover,Pm,Max_gera):-
%onde:
  %N=tamanho da população inicial
  %L=tamanho do cromossomo em bits
  %Ponto_Crossover= ponto de corte do crossover
  %Pm=posição do bit de mutação
  %Max_gera=quantidade máxima de iterações
          N>=2,
          gera_pop_inic(N,L,Pop),
          adapta(Pop,Lfitness),
          fitness_medio(N,Lfitness,Fmedio),
          melhores(Lfitness,fit(_,0),_,[[0]],Melhores),
          elimina_repetidos(Melhores,Mel),
          tell('teste.pro'),
          teste_convergencia1(L,Lfitness,Mel),!,
                  geracoes(N,L,Ponto_Crossover,Pm,1,Max_gera,Lfitness,Fmedio,Melhores,Solucao).
```

4.3 GERAÇÃO DA POPULAÇÃO

4.3.1. População Inicial

De modo geral, a população inicial é gerada randomicamente obedecendo condições pré-estabelecidas pelo usuário, de acordo com conhecimento prévio do problema a ser otimizado. Essas condições devem ser as mais restringentes possíveis, pois irão diminuir o tempo de convergência, uma vez que os valores randômicos estarão mais próximo da solução desejada.

A seguir o trecho do programa correspondente a geração da população inicial:

```
%Geração da população inicial de N elementos com L bits cada
gera_pop_inic(1,L,[C1]):- gera(L,C1),!.
gera_pop_inic(N,L,[C1|Resto]):- gera(L,C1),
                                N1 is N-1,
                                gera_pop_inic(N1,L,Resto).
gera(1,[1]):- X is random,
              X>=0.5,!.
gera(1,[0]):- !.
gera(L,[X|Y]):- gera(1,[X]),
                L1 is L-1,
                gera(L1,Y).
```

4.3.2 População geradas

Ao finalizar a busca pelas melhores soluções a cada geração, o programa exibirá o resultado parcial obtido, de acordo com o trecho abaixo:

```
% Exibir resultados parciais de cada geracao
imprime(Geracao,Lfitness,Fmedio,Melhores,Lpares,Lcruza,Lpares_muta,Lmuta):-
        write('Geracao = '),write(Geracao),nl,
        write('Lfitness = '),write(Lfitness),nl,
        write('Fitness_medio = '),write(Fmedio),nl,
        write('Melhores Elementos Selecionados = '),write(Melhores),nl,
        write('Pares para Cruzamento - Lpares = '),write(Lpares),nl,
        write('Resultado do Cruzamento - Lcruza = '),write(Lcruza),nl,
        write('Elemento e Posicao a realizar mutacao - Lpares_muta = '),write(Lpares_muta),nl,
        write('Resultado da Mutacao - Lmuta = '),write(Lmuta),nl,nl.
```

4.4 FITNESS

Em conjunto com a codificação, a função fitness constitui-se como um dos passos mais difíceis e também mais importantes do desenvolvimento do AG, pois é ela que determina o quão bem um programa é capaz de resolver um problema específico.

Trata-se da função objetivo “f”, determinada na Biologia de função aptidão (fitness). Esta função “f” é portanto a medida de utilidade que se deseja maximizar.

Nas populações naturais a aptidão é determinada pela capacidade que os organismos têm de lutar pela sobrevivência, resistindo a predadores, doenças, escassez de alimentos e outros obstáculos. No meio artificial é a função de avaliação que decide quais os cromossomos (strings) que sobreviverão e quais morrerão.

No estudo de caso deste trabalho, o valor de fitness associado a cada cromossomo corresponde a quantidade de bits 1 no string, sendo que no problema analisado, é desejada a maximização da fitness (um maior número de bits 1). O trecho de programa abaixo ilustra esta situação:

```
% Calculo da fitness de cada cromossomo
fitness([0],0):- !.
fitness([1],1):- !.
fitness([1|RC],F):- fitness(RC,F1),
                    F is 1 + F1,!.
fitness([0|RC],F):- fitness(RC,F).


% Cálculo do fitness médio
fitness_medio(N,Lfitness,Fmedio):-
        somat_fit(Lfitness,Sfit),
        Fmedio is Sfit/N.

somat_fit([],0):- !.
somat_fit([fit(_,F)|Rfit],Sfit):-
        somat_fit(Rfit,S),
        Sfit is F+S.
```

4.5 OPERADORES GENÉTICOS

A base do funcionamento dos algoritmos genéticos são os chamados operadores genéticos. Os operadores genéticos transformam a população através de sucessivas gerações, estendendo a busca até chegar a um resultado satisfatório, ou seja, objetivando a seleção dos indivíduos mais aptos, necessários para que a população se diversifique e mantenha características de adaptação adquiridas durante as etapas de processos anteriores.

São três as possíveis operações realizadas nos AG:

4.5.1 Seleção:

Como já sabemos, cromossomos são selecionados da população para serem os pais no cruzamento. Segundo BARRETO (1999, p.171) a *seleção tem por objetivo fazer com que somente os elementos mais adaptados da geração anterior participem do processo que irá gerar a nova seleção.* O problema reside justamente em como proceder a escolha destes elementos.

De acordo com a Teoria de Darwin, os melhores deverão sobreviver e assim criar novas saídas. Existem muitos métodos de seleção para escolha dos melhores cromossomos, dentre estes citamos:

a) Método de Seleção pela Roleta Simples: Os pais são escolhidos de acordo com a sua fitness, onde os melhores cromossomos serão aqueles com maior valor de função. Imagine uma roda de roleta onde são colocados todos os cromossomos da

população, cada um ocupará um determinado espaço de acordo com sua respectiva fitness como mostra a figura abaixo.

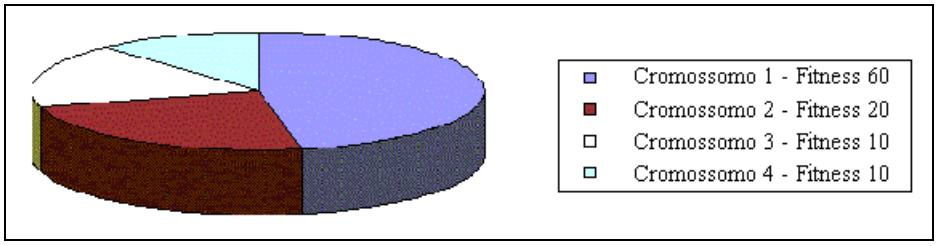


Figura 7 – Método de Seleção pela Roleta Simples.

Então a roleta é girada um determinado número de vezes, dependendo do tamanho da população, para escolha dos indivíduos que participarão da próxima geração. Como os cromossomos de maior fitness ocupam uma porção maior da roleta, a probabilidade de serem selecionados é maior do que os de aptidão mais baixa, dada a representatividade em porção relativamente menor na roleta.

b) Método de Seleção pela Roleta Ponderada (ou Método de Seleção por classificação): Quando os valores de fitness dos cromossomos são muito diferentes, a seleção pelo método da roleta simples, não é muito recomendada, pois as chances de escolha dos cromossomos com valores de fitness inferiores são bem irrelevantes em comparação aos cromossomos com fitness superiores.

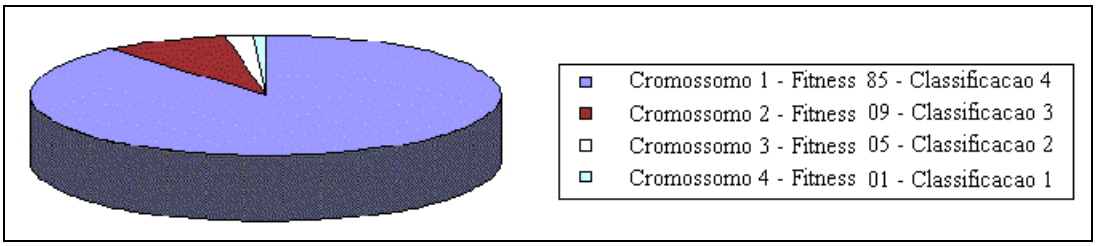


Figura 8 – Método de Seleção pela Roleta Ponderada: antes da classificação.

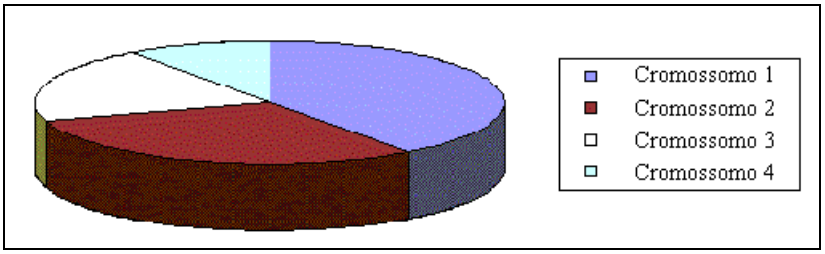


Figura 9 – Método de Seleção pela Roleta Ponderada: depois da classificação.

Este método primeiro classifica a população para que depois cada cromossomo receba seu valor de fitness de acordo com essa classificação. O pior cromossomo recebe fitness 1, o segundo pior, fitness 2, e o melhor receberá fitness N, onde N corresponde ao total de cromossomos da população. Na figura 9, podemos perceber que a probabilidade de seleção muda ao utilizarmos este método, pois todos os cromossomos passam a ter chances iguais de serem selecionados.

d) Método de Seleção Steady-State: Este tipo de método é um caso particular de seleção, pois a idéia principal é que uma parte (a melhor parte) dos cromossomos da população deve sobreviver para a próxima geração.

Em todas as gerações, os melhores cromossomos são escolhidos de acordo com seu valor de fitness para permanecer, e assim gerar a prole. Os piores, ou seja, os cromossomos com fitness inferior são removidos, e a prole gerada pelos cromossomos de maior fitness é colocada em seu lugar. Portanto, uma parte da população sobrevive na nova geração.

d) Método de Seleção por Elitismo: Quando uma nova população é criada pelo cruzamento e mutação, existe uma grande chance de que o melhor cromossomo da geração anterior seja perdido. Neste tipo de método, essa perda não acontece, pois o elitismo se preocupa em guardar os melhores cromossomos antes que a nova população seja gerada.

O Elitismo pode intensificar o desempenho do AG justamente porque ele evita a perda da melhor solução encontrada em gerações anteriores, por este motivo foi o método escolhido para ser implementado no estudo de caso em questão, conforme trecho do programa a seguir:

```
% Selecao dos elementos através do Método do Elitismo
melhores(Lfitness,fit(_,F),Melhores_gera,Melhores_ant,Melhores_novos):-
        melhor(Lfitness,fit(_,F),Melhor),
        fitness(Melhor,Fmel),
        selec_mel(Lfitness,Fmel,Melhores_gera),
        verifica_melhores(Melhores_gera,Melhores_ant,Melhores_novos).
```

```
melhor([fit(C,F)],fit(Cmelhor,Fitmelhor),C):- F > Fitmelhor,!.
melhor([fit(C,F)],fit(Cmelhor,Fitmelhor),Cmelhor):- !.
melhor([fit(C,F)|RLfitness],fit(Cmelhor,Fitmelhor),C1):-
        F > Fitmelhor,!,
        melhor(RLfitness,fit(C,F),C1).
melhor([fit(C,F)|RLfitness],fit(Cmelhor,Fitmelhor),C1):-
        melhor(RLfitness,fit(Cmelhor,Fitmelhor),C1).

elimina_repetidos([Melhor],[Melhor]):- !.
elimina_repetidos([Melhor|RMelhores],Mel):-
        pertence(Melhor,RMelhores),!,
        elimina_repetidos(RMelhores,Mel).
elimina_repetidos([Melhor|RMelhores],[Melhor|Mel]):-
        elimina_repetidos(RMelhores,Mel).

pertence(X,[X|Y]):-!.
pertence(X,[Y|Z]):-pertence(X,Z).

selec_mel([],_,[]):- !.
selec_mel([fit(C1,F1)|RLfitness],Fmel,[C1|RMelhores]):-
        F1=Fmel,!,
        selec_mel(RLfitness,Fmel,RMelhores).
selec_mel([fit(C1,F1)|RLfitness],Fmel,Melhores):-
        selec_mel(RLfitness,Fmel,Melhores).

verifica_melhores([X|Melhores_gera],[Y|Melhores_ant],Melhores_novos):-
        fitness(X,F1),
        fitness(Y,F2),
        ver(F1,F2,[X|Melhores_gera],[Y|Melhores_ant],Melhores_novos).

ver(F1,F2,Melhores_gera,Melhores_ant,Melhores_novos):-
        F1>F2,!,
        Melhores_novos = Melhores_gera.
ver(F1,F2,Melhores_gera,Melhores_ant,Melhores_novos):-
        F1<F2,!,
        Melhores_novos = Melhores_ant.
ver(F1,F2,Melhores_gera,Melhores_ant,Melhores_novos):-
        concatena(Melhores_gera,Melhores_ant,Melhores_novos).
```

4.5.2 Cruzamento

O *cruzamento* ou *crossover* é o operador responsável pela recombinação de características dos pais durante a reprodução, permitindo que as gerações herdem essas características. A operação consiste na troca entre si de parte dos cromossomos dos elementos pais, gerando elementos filhos que, apesar de serem cromossomos diferentes, ainda guardam influências dos cromossomos pais. Ele é considerado o operador genético predominante.

As formas de utilização desse operador são:

a) Ponto Simples: um ponto de cruzamento é escolhido e a partir deste ponto as informações genéticas dos pais são trocadas. As informações anteriores a este ponto em um dos pais são ligadas às informações posteriores no outro pai.

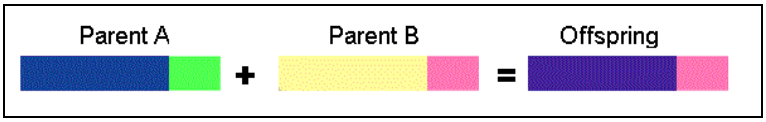


Figura 10 – Operador Crossover usando Ponto Simples.

Por exemplo, considerando os cromossomos a seguir **110**01011 e 101**11101**, com ponto de cruzamento 3, o cromossomos resultante seria **11011101**.

O trecho de programa abaixo implementa o operador genético crossover utilizando ponto simples, tendo em vista a simplicidade do problema estudado:

```
% Operação de Cruzamento na forma de utilização ponto simples

cruzamento(Lk,[par(C1,C2)],[Xc1,Xc2]):-
        localiza(Lk,C1,X),
        localiza(Lk,C2,Y),
        cruza(Lk,Y,C1,Xc1),
        cruza(Lk,X,C2,Xc2),!.
cruzamento(Lk,[par(C1,C2)|Rpares],[Xc1,Xc2|R]):-
        cruzamento(Lk,[par(C1,C2)],[Xc1,Xc2]),
        cruzamento(Lk,Rpares,R).

localiza(1,[X|Y],[X]):- !.
localiza(N,[X|Y],W):- N1 is N-1,
                      localiza(N1,Y,Z),
                      concatena([X],Z,W).

cruza(N,Z,X,Y):- deleta(N,X,W),
                 concatena(Z,W,Y).
deleta(1,[X|Y],Y):- !.
deleta(N,[X|Y],Z):- N>1,
                    N1 is N-1,
                    deleta(N1,Y,Z).
```

b) Multi-pontos: é uma generalização da idéia de troca de material genético através de pontos, onde dois ou mais pontos de cruzamento podem ser utilizados;

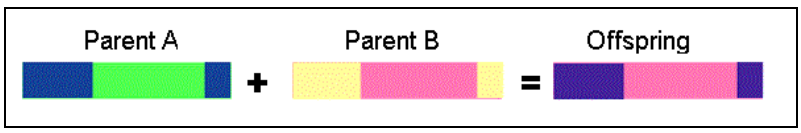


Figura 11 – Operador Crossover usando Multi-Pontos.

Por exemplo, considerando os cromossomos a seguir **11**0010**01** e 10**1111**01, com ponto de cruzamento 2 e 6, o cromossomo resultante da operação cruzamento seria **11111101**.

c) Uniforme: não utiliza pontos de cruzamento pré-definidos (a escolha é feita randomicamente), mas determina, através de um parâmetro global, qual a probabilidade de cada variável ser trocada entre os pais.

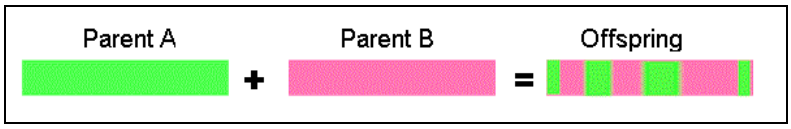


Figura 12 – Operador Crossover usando método uniforme.

d) Aritmético: algumas operações aritméticas são executadas para gerar uma nova prole.

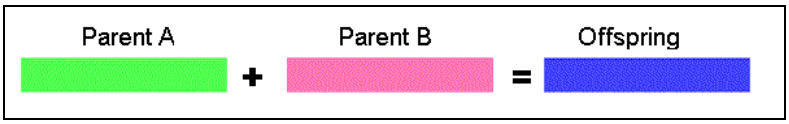


Figura 13 – Operador Crossover usando método aritmético.

Por exemplo, considerando os cromossomos 01001011 e 11011111, o cromossomo resultante utilizando a forma aritmética AND seria 00101010.

4.5.3 Mutação

Os operadores de mutação são necessários para a introdução e manutenção da diversidade genética da população, alterando arbitrariamente um ou mais componentes de uma estrutura escolhida, fornecendo meios para a introdução de novos elementos na população.

Trata-se de uma alteração aleatória e ocasional do valor de qualquer uma das posições do string. Estas alterações ocorrem de acordo com uma probabilidade pré-fixada, que juntamente com a probabilidade de cruzamento e o tamanho da população, controlam todo o processo de busca, influenciando diretamente a velocidade de convergência e evitando que aconteça a supremacia de uma determinada sub-população.

Desta forma, a mutação assegura que a probabilidade de se chegar a qualquer ponto do espaço de busca nunca será zero. Por ser considerado um operador genético secundário, geralmente a taxa de mutação Pm é pequena.

Algumas possíveis variações do operador mutação são apresentadas a seguir.

É importante ressaltar que alguns desses casos são específicos para codificação binária.

a) Bit Invertido: bits específicos do cromossomo são selecionados e invertidos. Como exemplo, pode ser verificado na *Figura 13* uma analogia do processo de mutação, onde é alterada a cor dos *strings* nas posições "2", "4" e "7", da situação "A" para a situação "A".

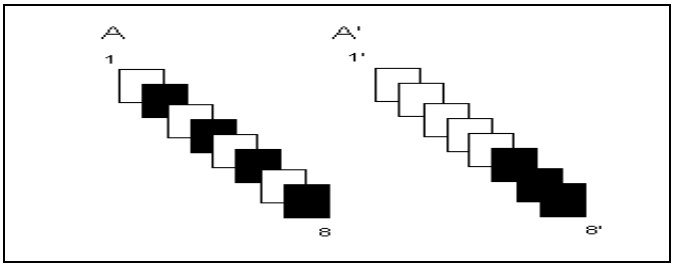


Figura 14 – Operador Mutação usando Bit Invertido.

O trecho de programa a seguir implementa o operador mutação, numa quantidade tal de bits de acordo com o valor atribuído inicialmente a Pm :

```
%Operação de Mutação por inversão de bits
% Sendo Pm o percentual desejado de bits a serem mutados, tem-se
% N*L*Pm/100 = Num_bits que serão invertidos.
% Assim, a mutação é realizada gerando-se aleatoriamente Num_bits
mutacao(N,L,Pm,Lcruza,Lmuta,Lpares_muta):-
        Vezes is integer(N*L*Pm/100) +1,
        gera_pares_muta(N,L,Vezes,Lpares_muta),
        muta1(Lpares_muta,Lcruza,Lmuta).

gera_pares_muta(N,L,1,[par(Npop,Lbit)]):- Npop is integer(random*N)+1,
                                          Lbit is integer(random*L)+1,!.
gera_pares_muta(N,L,Vezes,[par(Npop,Lbit)|R]):-
        gera_pares_muta(N,L,1,[par(Npop,Lbit)]),
        V is Vezes-1,
        gera_pares_muta(N,L,V,R).

muta1([Par1],Lcruza,Lmuta):-
        muta2(Par1,Lcruza,Lmuta),!.
muta1([Par1|R],Lcruza,Lmuta):-
        muta2(Par1,Lcruza,Lmuta1),
        muta1(R,Lmuta1,Lmuta).

muta2(par(1,Lbit),[C1|RC],[Cmuta|RC]):-
        muta3(Lbit,C1,Cmuta),!.
muta2(par(Npop,Lbit),[C1|RC],[C1|RCmuta]):-
        Npo1 is Npop-1,
        muta2(par(Npo1,Lbit),RC,RCmuta).

muta3(1,[0|Y],[1|Y]):- !.
muta3(1,[1|Y],[0|Y]):- !.
muta3(N,[X|Y],[X|Z]):-
```

```
        N1 is N-1,
        muta3(N1,Y,Z).
```

b) Mudança de posições: Dois bits do cromossomo são selecionados e a posição é trocada.

Por exemplo, o cromossomo (1 **2** 3 4 5 6 **8** 9 7) depois de realizado esta forma de operador resultaria no cromossomo (1 **8** 3 4 5 6 **2** 9 7).

c) Adding:. Este tipo de mutação é utilizado quando os valores de codificação do cromossomo são valores reais. Escolhido um desses valores das posições do cromossomo é subtraído (ou adicionado) um valor qualquer.

Por exemplo, o cromossomo inicial (1.29 5.68 **2.86 4.11** 5.55), resultaria no cromossomo (1.29 5.68 **2.73 4.22** 5.55),

4.6 TESTE DE CONVERGÊNCIA

Segundo BARRETO (1999), o teste de convergência é responsável pela análise do desempenho desenvolvido pela população tendo em vista o objetivo pretendido.

No nosso estudo de caso utilizamos dois testes de convergência: o primeiro quando é encontrada uma solução ótima ou próxima do aceitável, e o segundo quando a quantidade de proles geradas atingir o valor Max_gera, recebido como parâmetro de entrada.

Em ambas as situações, o algoritmo finaliza o processo de busca, apresentado um resultado comentado.

```
% Teste de Convergência 1 - encontrou solução
teste_convergencia1(L,_,[Melhor|RMelhores]):-
        fitness(Melhor,Fmelhor), Fmelhor<L,!.
teste_convergencia1(_,Lfitness,Melhores):-
    write('Solucao encontrada antes de extrapolar a quantidade
      maxima de geracoes !!!'),nl,nl,
    write('A lista de fitness da populacao eh:'),nl,
    write('Lfitness = '),
    write(Lfitness),nl,nl,
    write('E a lista de solucoes eh:'),nl,
    write('ListaSolucoes = '),
    write(Melhores),nl,
    told,
    fail.

% Teste de Convergência 2 - não encontrou a melhor solução,
    mas extrapolou o número máximo de gerações pré-estabelecida
teste_convergencia2(Gera1, Max_gera,_):- Gera1 =< Max_gera,!.
teste_convergencia2(Gera1, Max_gera,Melhores):-
    write('Extrapolou a quantidade maxima de geracoes
```

```
        definidas ...'),nl,nl,
    write('A melhor lista de solucoes ate a geracao '),
write(Max_gera),
write(' eh:'),nl,
    write('Solucoes = '),
    write(Melhores),nl,
    told,
    fail.
```

5 PARÂMETROS DO AG

Considerando as necessidades do problema e dos recursos disponíveis, os parâmetros principais a serem definidos são:

5.1 TAMANHO DA POPULAÇÃO

Esse parâmetro afeta o desempenho global e a eficiência dos AG. Uma população bem pequena oferece uma menor cobertura do espaço de busca, causando uma queda no desempenho. Uma população suficientemente grande fornece uma maior cobertura do domínio do problema e previne a convergência prematura para soluções locais. Entretanto, com uma população grande em excesso tornam-se necessários recursos computacionais maiores, ou um tempo maior de processamento do problema. Deve-se então buscar um ponto de equilíbrio a respeito do tamanho escolhido para a população.

5.2 TAXA DE CRUZAMENTO

Quanto maior for essa taxa, mais rapidamente novas estruturas serão introduzidas na população. Entretanto, isto pode gerar um efeito indesejado pois uma maior parte da população será substituída, ocorrendo assim perda de variedade genética e de estruturas de alta aptidão, convergindo a uma população com indivíduos extremamente parecidos, indivíduos estes de solução boa ou não. No entanto, com um valor muito baixo o algoritmo pode levar mais tempo para oferecer uma resposta aceitável.

5.3 TAXA DE MUTAÇÃO

Mesmo uma baixa taxa de mutação previne que uma dada posição fique estagnada em um valor, além de possibilitar que se chegue em qualquer ponto do espaço de busca. Deve ser considerado que uma taxa de mutação muito alta, tende a tornar a busca essencialmente aleatória.

6 APLICAÇÕES DOS AG

Os AG podem ser aplicados nos mais diversos ramos do conhecimento, objetivando uma otimização multidimensional em que se busca uma solução global. Dentre essas aplicações podemos citar:

1) Multiprocessor Scheduling: objetiva diminuir o custo derivado da comunicação entre processos em um ordenador paralelo de memória distribuída;
2) Biologia Molecular e Fisicoquímica: em problemas relacionados a estrutura molecular existem experimentos que dão indícios sobre a validade de hipóteses de trabalho que sustentam a utilização de técnicas baseadas em populações devido a estrutura da função objetivo;
3) Engenharia em Construções: objetiva otimização discreta de estruturas;
4) Busca em Base de Dados: objetiva otimizar o processo de busca tradicional;
5) Geofísica: problemas como o *Seismic Waveform Inversion*, onde se combina a informação a priori do modelo estatístico com os dados obtidos, e também no problema *Inversion for Seismic Anisotropy*;
6) Redes Neurais (RN): os estudos de AG relacionados as RN vêm sendo realizados no aspecto de otimização relacionados com a busca de funções lineares discriminantes em problemas de classificação.
7) Compressão de dados: objetiva identificar um método para encontrar um sistema de funções locais iteradas (LIFS) de codificação de imagens, cujo resultados tenham qualidade similar a utilização do método convencional de compressão fractal, com tempo relativamente menor.

7 RECOMENDAÇÕES

Alguns pontos devem ser levados em consideração na implementação de algoritmos genéticos. Essas considerações são bem genéricas, tendo em vista que na maioria das vezes é necessário desenvolver AG específicos para cada caso estudado, pois não existe um modelo geral o qual descreveria os parâmetros do AG para qualquer problema. Mas, resolvemos colocar as recomendações abaixo relacionadas, pois foram os resultados de estudos empíricos sobre os AG que obtivemos em nosso estudo.

1. Taxa de Crossover: A taxa de cruzamento deve ser alta. o resultado de experimentos em outros trabalhos demonstra que valores entre 80% e 95% são os mais utilizados, no entanto em nosso trabalho utilizamos taxa de 100%, que

resultou numa boa performance, tendo em vista a simplicidade do problema abordado;

2. Taxa de Mutação: Ao contrário da taxa de crossover, a taxa de mutação deve ser baixa, oscilando entre 0,5% e 1%. Como em nossa implementação a taxa de mutação é uma entrada parametrizada (Pm), evidenciamos essa recomendação;
3. Tamanho da População: Um tamanho grande de população também não é muito recomendável, pois prejudica o aspecto velocidade na busca pela solução do problema. Um tamanho considerado razoável fica entre 20 e 30 indivíduos; no entanto, em alguns problemas que encontramos em nossa pesquisa bibliográfica, tamanhos de 50 a 100 demonstraram melhores resultados. Utilizamos o tamanho de população 10 (também uma entrada parametrizada), porque conseguia realizar com sucesso a experiência;
4. Quanto a seleção dos indivíduos, a estratégia recomendada é o método do elitismo, tendo em vista que pelo menos uma das melhores soluções é mantida na nova geração, não existindo a perda do melhor cromossomo das gerações anteriores. Foi esse o método de seleção adotado na implementação;
5. A codificação dos cromossomos depende principalmente do problema, mas também do tamanho da instância do problema;
6. Crossover e tipo de mutação: a escolha dos operadores está estritamente ligada ao problema e a codificação do cromossomo.

Uma das grandes vantagens que podemos citar refere ao uso de AG está em seu paralelismo, ou seja, a execução simultânea de cada procedimento, que também é uma característica da linguagem Prolog.

Também é relativamente fácil de ser implementado, pois uma vez que se consegue montar um AG, uma outra implementação poderá ser definida para resolver problemas diferentes, codificando um novo cromossomo (um objeto apenas) ou, com a mesma codificação do cromossomo, apenas mudando a função de fitness. No entanto, em certos casos, mudar a codificação do cromossomo e a função de fitness pode ser muito difícil.

A desvantagem do AG reside no tempo de operação, pois dependendo do problema, pode ser muito lento em comparação com outros métodos disponíveis.

Os AG, ao usarem ao mesmo tempo operações randômicas e estruturadas, operam de forma criativa e singular a troca de informações entre os *strings*, o que emula

de certa maneira a forma de descobrir e pesquisar do ser humano. Até recentemente, precisava-se de pesados recursos computacionais e muita matemática e estatística.

Segundo os estudiosos, os Algoritmos Genéticos simplificam a solução, porque não há a necessidade de se trabalhar sobre todos os dados do problema; basta que se conheça o que deve ser maximizado e quais são as variáveis que devem ser julgadas. Essa característica dos AG também permite que se interrompa o processamento para ver as soluções já atingidas, possibilitando que mudanças possam ser realizadas para otimizar ainda mais o processo.

BIBLIOGRAFIA

BARRETO, Jorge Muniz. **Inteligência Artificial no Limiar do Século XXI**. Florianópolis: J.M.Barreto, 1999.

BORGES, Paulo Sérgio S. **Notas e transparências de sala de aula**. Belém: CESUPA, Curso de Pós Graduação em Inteligência Computacional - Parte 1 Computação Evolucionária, 2000.

DOLAN, Ariel. **Artificial Life and others experiments.** Disponível em: http://www.arieldolan.com. Data de acesso: 2 jun 2020.

DOLAN, Ariel. **Articial Life and Java.** Disponível em: http://www.aridolan.com Data de acesso 2 jun 2020.

FERNANDEZ, Jaime J., LENAERTS, Tom. **The Genetic Programming Tutorial Notebook.** Disponível em: http://geneticprogramming.com/Tutorial/tutorial.html Houston
Texas Data de acesso.2 jun 2020.

HAUPT, Randy L., HAUPT, Sue Ellen. **Pratical Genetic Algorithms**. New York: Wiley & Sons, 1998.

MITCHELL, Melanie. **An Introduction to Genetic Algorithms**. A Bradford Book, Massachusetts, 1996.

OBITKO, Marek. **Introduction to Genetic Algorithms**. 1998. Disponível em: http://cs.felk.cvut.cz/~xobitko/ ga/menu.html Data de acesso 26 mai 2020.

MENDES FILHO, Elson Felix. **Inteligência Artificial:** Algoritmos Genéticos. Disponível em: http://www.icmsc.sc.usp.br/~prico/gene1.html. Botucatu SP Data de acesso 2 jun 2020.