1. Computação Evolutiva

Prof. Renato Tinós

Programa de Pós-Graduação Em Computação Aplicada

Depto. de Computação e Matemática (FFCLRP/USP)

1.2. Algoritmos Genéticos

1.2.5. Implementação Computacional

```
// Baseada em [Goldberg, 1989]
...
/* Definição de constantes */
defina     maxpop = 100                          // tamanho maximo da populacao
defina     tamcrom = 30                          // tamanho do cromossomo

/* Definição dos tipos de dados */
tipo_de_dado alelo : booleana                    // tipo de dado que os alelos
                                                 //   podem assumir (poderia ser
                                                 //   outros tipos como real e
                                                 //   inteiro)

tipo_de_dado cromossomo : alelo[tamcrom]
tipo_de_dado individuo :       {
                               crom :      cromossomo        // cromossomo
                               fitness :  real               // funcao de avaliacao
                               pai1 , pai2 , jcross : inteiro // indice dos pais e ponto de
                                                             //  crossover
                               }
tipo de dado populacao :       individuo[maxpop]
...
```

1.2.5. Implementação Computacional

```
...

/* Definição da variáveis globais */
declare   popvelha , popnova : população
declare   poptam, lcrom, gen , maxgen : inteiro
declare   pcross , pmutacao , somafitness : real
declare   nmutacao , ncross : inteiro          // tipos usados nas estatisticas
declare   media , max , min : real             // tipos usados nas estatisticas

...
```

1.2.5. Implementação Computacional

```
/* Programa principal */
procedimento ag(  )

        gen ← 0
        inicialização ( )                       // procedimento para inicialização
                                                //  das variáveis e da população
        faça
                gen ← gen + 1                   // número de gerações
                geração ( )
                estatística( poptam , max , media , min , somafitness , popnova)
                impressão ( )                   // proc. para impressão na tela
                popvelha ← popnova
        enquanto ( gen ≤ maxgen )
```

1.2.5. Implementação Computacional

```
procedimento geração( )

        declare j , pai1, pai2 , jcross : inteiro
         j ← 1
        faça
                // Escolha dos Pais
                pai1 ← seleção( poptam , somafitness , popvelha )
                pai2 ← seleção( poptam , somafitness , popvelha )
                // Reproducao
                crossover( popvelha[ pai1 ].crom , popvelha[ pai2 ].crom ,
                                popnova[ j ].crom , popnova[ j+1 ].crom,
                                lcrom , ncross , nmutacao , &jcross , pcross , pmutacao )
                // Filho j
                popnova[ j ].fitness ← calcfitness( popnova[ j ].crom , lcrom )
                popnova[ j ].pai1 ← pai1                        // pai 1 do filho j
                popnova[ j ].pai2 ← pai2                        // pai 2 do filho j
                popnova[ j ].jcross ← jcross                    // ponto de crossover do filho j
                // Filho j + 1
                popnova[ j+1 ].fitness ← calcfitness( popnova[ j+1 ].crom , lcrom )
                popnova[ j+1 ].pai1 ← pai1                      // pai 1 do filho j
                popnova[ j+1 ].pai2 ← pai2                      // pai 2 do filho j
                popnova[ j+1 ].jcross ← jcross                  // ponto de crossover do filho j
                 j ← j + 2                              // incremento do índice da populacao
        enquanto ( j ≤ poptam)
```

1.2.5. Implementação Computacional

```
função seleção( poptam : inteiro ; somafitness : real ; pop : população   ) : inteiro

        declare randomico , somaparcial : real
        declare j : inteiro
        somaparcial  ← 0
        j ← 0
        randomico ← rand(0,1) * somafitness      // gera o ponto de parada da roleta
                                                 //  rand(l,r) gera um variável aleatória real
                                                 //     entre [ l , r ] com  distribuição uniforme
        faça
                    j ← j + 1
                    somaparcial ← somaparcial + pop[ j ].fitness
        enquanto (somaparcial < randomico  e  j < poptam)
        retorne ( j )
```

1.2.5. Implementação Computacional

```
procedimento crossover(  pai1 , pai2 , filho1 , filho2 : cromossomo ;
                         lcrom , ncross , nmutacao , jcross : inteiro;
                         pcross , pmutacao : real)

    declare j : inteiro
    se ( rand(0,1) ≤ pcross )
        jcross ← randint( 1 , lcrom-1 )      // gera o ponto de crossover
                                             // randint(l,r) gera um variável aleatória
                                             //  inteira entre [ l , r ] com distrib. uniforme
        ncross ← ncross + 1                  // contador de crossovers
    senão
        jcross ← lcrom
    fim se
    para j ← 1 até j ← jcross
        filho1[ j ] ← mutacao( pai1[ j ] , pmutacao , nmutacao )
        filho2[ j ] ← mutacao( pai2[ j ] , pmutacao , nmutacao )
    fim para
    se ( jcross < lcrom )
        para j ← jcross+1 até j ← lcrom
            filho1[ j ] ← mutacao( pai2[ j ] , pmutacao , nmutacao )
            filho2[ j ] ← mutacao( pai1[ j ] , pmutacao , nmutacao )
        fim para
    fim se
```

1.2.5. Implementação Computacional

```
função mutacao( alelovalor : alelo ; pmutacao : real ; nmutacao : inteiro ) : inteiro

        se ( rand(0,1) ≤ pmutacao )
                nmutacao ← nmutacao + 1     // contador de mutações
                retorne ( !alelovalor )     // not alelovalor
                                            // se os alelos assumissem valores
                                            // inteiros, deveria
                                            // retornar um numero inteiro aleatorio
        senão
                retorne ( alelovalor )
        fim se
```

1.2.5. Implementação Computacional

```
/* Calcula estatísticas da população */
procedimento estatística(  poptam : inteiro ; max , media , min , somafitness : real ;
                                        pop : populacao)

        declare j : inteiro
        somafitness ← pop[ 1 ].fitness      // soma do fitness da população
        min ← pop[ 1 ].fitness              // mínimo valor de fitness da população
        max ← pop[ 1 ].fitness              // máximo valor de fitness da população

        para j ← 2 até j ← poptam
                somafitness ← somafitness + pop[ j ].fitness
                se (pop[ j ].fitness > max )
                        max ← pop[ j ].fitness
                fim se
                se (pop[ j ].fitness < min )
                        min ← pop[ j ].fitness
                fim se
        fim para

        media ← somafitness / poptam      // média de fitness da população
```

1.2.5. Implementação Computacional

Exemplo 2.4. Achar o máximo da função

$$f(z) = (z)^{10}$$

Sendo z um número inteiro positivo entre 0 e $(2^{30}-1)$

[Goldberg, 1989]

- Observações
  - Se um esquema enumerativo é utilizado, deve-se analisar 2^{30} possíveis soluções
    - se o tempo de análise de cada solução for de 1 ms, então serão necessários cerca de 2 anos para a análise de todas as soluções
  - Se a função for normalizada entre 0 e 1, então apenas 1,05 % das possíveis soluções resultam em valores da função maiores que 0,9

1.2.5. Implementação Computacional

Exemplo 2.4.

- Algoritmo Genético
  - Função de avaliação
    - $fitness\ (z_i) = (z_i / z_{max})^{10}$
      - Varia entre 0 e 1
  - Decodificação
    - Vetor de bits $\mathbf{x}_i$ com dimensão 30
      - Exemplo: $\mathbf{x}_i = [1\ 0\ 1\ ...\ 0]^T$
    - O vetor de bits é decodificado para números inteiros
      - $z_i = x_i(1)\ 2^0 + x_i(2)\ 2^1 + x_i(3)\ 2^2 + ... + x_i(30)\ 2^{29}$

1.2.5. Implementação Computacional

Exemplo 2.4.

- AG: função de avaliação

```
/* Calculo da função de avaliação f (z) = (z/z_max)^10 */
função calcfitness( crom : cromossomo ; lcrom : inteiro ) : real

        declare zmax , z : real
        zmax ← 1073741823.0                 // fator de normalização
                                            //  o máximo valor de z é 2^30 – 1 (ver próx. slide)
        z = decodificação( crom , lcrom )   // decodifica o vetor de bits em um valor int
                                            //  fenótipo (pode assumir outros tipos)

        retorne (  potência( z / zmax ,10 )  )
```

1.2.5. Implementação Computacional

Exemplo 2.4.

- AG: decodificação

```
/* Decodifica um vetor de bits em um número inteiro */
função decodificação( vetor : cromossomo ; lbits : inteiro ) : inteiro

        declare j , valor , potdedois : inteiro
        potdedois ← 1
        valor ← 0
        para j ← 1 até j ← lbits
                se ( vetor[ j ] )
                        valor ← valor + potdedois
                fim se
                potdedois ← 2*potdedois
        fim para
        retorne ( valor )
```

1.2.5. Implementação Computacional

Exemplo 2.4.

- Ver [Goldberg, 1989]
- AG: parâmetros
  - poptam = 30
  - pcross = 0,6
  - pmutação = 0,0333

Population Report

Generation 0				Generation 1				
#	string	x	fitness	#	parents	xsite	string	x	fitness
1)	11100001100110000010111111010 0	9.4621E+08	0.2824	1)	(1,19)	30	11100001100110000010110111010 0	9.4621E+08	0.2824
2)	11001100101101011100000010000 1	8.5862E+08	0.1069	2)	(1,19)	30	11010101111100011100100110001 01	8.9712E+08	0.1658
3)	01010111100101111000001000 1101	3.6739E+08	0.0000	3)	(23,19)	19	11010101111000000110101100000 01	8.9655E+08	0.1647
4)	01100111100001110110111101 1010	4.3423E+08	0.0001	4)	(23,19)	19	11111100100000000100100110001 01	1.0591E+09	0.8715
5)	01111111101001010110101101 10110	5.3539E+08	0.0009	5)	(19, 1)	11	11100001100110000011001100010 1	9.4621E+08	0.2824
6)	10110111100100001100010110 1101	7.6993E+08	0.0359	6)	(19, 1)	11	11010101111100000100011111010 0	8.9707E+08	0.1657
7)	00011010111110000100101111 1000	1.1312E+08	0.0000	7)	(16, 1)	6	11100001100110000010111110010 0	9.4621E+08	0.2824
8)	01010011101011101000111110 0010	3.5099E+08	0.0000	8)	(16, 1)	6	11001101010110110001100111010 0	8.6132E+08	0.1103
9)	01101100111000101001111000 1011	4.5670E+08	0.0002	9)	(23,17)	30	10111100100001001111011001000 1	7.9071E+08	0.0469
10)	01001101011010100000110101 1011	3.2470E+08	0.0000	10)	(23,17)	30	11001110100100010001110110001 1	8.6640E+08	0.1170
11)	01011101101010101100110100 0010	3.9287E+08	0.0000	11)	(6,26)	30	10110111000100001100010110110 1	7.6783E+08	0.0350
12)	01001101001100101011001000 1110	3.2379E+08	0.0000	12)	(6,26)	30	11001000010101010011011001111 0	8.4026E+08	0.0861
13)	11000010010111011101100001 00111	8.1520E+08	0.0636	13)	(2,19)	10	11010101111010001001000010000 1	8.9720E+08	0.1659
14)	01011000100111111010001001 10001	3.7171E+08	0.0000	14)	(2,19)	10	11001101101101011100001100010 1	8.6281E+08	0.1122
15)	01011011111000001011011100 11010	3.8538E+08	0.0000	15)	(17,17)	26	11001111100100010001110010001 1	8.7060E+08	0.1228
16)	11001101010110010001100110 0000	8.6129E+08	0.1103	16)	(17,17)	26	11001011110100010001110010001 1	8.5487E+08	0.1023
17)	11001111110100010001110010 00011	8.7165E+08	0.1243	17)	(19,17)	30	11010101111000001001001100011 1	8.9707E+08	0.1657
18)	10000000010001011110010101 11100	5.3802E+08	0.0010	18)	(19,17)	30	11001111110100000001110100001 1	8.7163E+08	0.1243
19)	11010101111000001001100010 0101	8.9707E+08	0.1657	19)	(19,19)	24	11010101111000001001001100010 1	8.9707E+08	0.1657
20)	01001111100001000100001101 1101	3.3352E+08	0.0000	20)	(19,19)	24	11010101111000001001001100010 1	8.9707E+08	0.1657
21)	00110101110011011101001000 0011	2.2567E+08	0.0000	21)	(26,17)	30	11001000010101010011011101110 0	8.4026E+08	0.0861
22)	00011001111100000110010011 0110	1.0880E+08	0.0000	22)	(26,17)	30	11000011110100010001110010001 1	8.2132E+08	0.0686
23)	10111100100000001111011001 0001	7.9064E+08	0.0469	23)	(23, 1)	3	11100001100110000000111111000 1	9.4621E+08	0.2824
24)	01110110000110001010010110 0111	4.9533E+08	0.0004	24)	(23, 1)	3	10111100100000001111010001010 0	7.9064E+08	0.0469
25)	01011011101000110101000101 0010	3.8436E+08	0.0000	25)	(27, 1)	10	11000001100110000010100111001 1	8.1199E+08	0.0612
26)	11001000010101010011011001 1110	8.4026E+08	0.0861	26)	(27, 1)	10	10100110000101010100110111010 0	6.9660E+08	0.0132
27)	10100110010111010100100110 0011	6.9778E+08	0.0134	27)	(1,17)	25	11001000100110000010111111010 0	8.4135E+08	0.0873
28)	10001111011101000010000011 0010	6.0169E+08	0.0031	28)	(1,17)	25	11100111110100010001110010001 1	9.7231E+08	0.3707
29)	01001010001000010000110110 0011	3.1092E+08	0.0000	29)	(1,19)	23	11010101000110000010111111010 0	8.9378E+08	0.1597
30)	00101100111100111001010110 0011	1.8854E+08	0.0000	30)	(1,19)	23	11100001111000000001001100010 1	9.4739E+08	0.2859

Note: Generation 1 & Accumulated Statistics: max=0.8715, min=0.0132, avg=0.1732, sum=5.1967, nmutation=35, ncross= 10

FIGURE 3.16 SGA run, generation report $t = 0$–1.

Population Report

Generation 6				Generation 7					
#	string	x	fitness	#	parents	xsite	string	x	fitness
1)	11110001101001001010111111 10001	1.0135E+09	0.5615	1)	(8, 6)	7	11111100101100000010111100 0100	1.0599E+09	0.8779
2)	11111100100010001000111111 10011	1.0592E+09	0.8726	2)	(8, 6)	7	11111100100000001001101110 1100	1.0591E+09	0.8715
3)	11111100100000001001111111 10100	1.0591E+09	0.8715	3)	(9,24)	9	11111100100000001001101010 0111	1.0591E+09	0.8715
4)	11111001111000000010111111 00100	1.0481E+09	0.7849	4)	(9,24)	9	11111110100000001000011111 10111	1.0675E+09	0.9430
5)	11111100110110000110111110 10110	1.0605E+09	0.8834	5)	(6,18)	3	11111101100000111010111111 00100	1.0633E+09	0.9070
6)	11111100101100000010111111 01100	1.0599E+09	0.8779	6)	(6,18)	3	11111000101100000011111111 01100	1.0431E+09	0.7484
7)	11111100100110000010111111 01100	1.0595E+09	0.8747	7)	(10,22)	22	10111001110000001001111111 10111	7.7910E+08	0.0405
8)	11111100100000001001011000 0100	1.0591E+09	0.8715	8)	(10,22)	22	11111100111101000010111110 0100	1.0610E+09	0.8872
9)	11111110100000001000001110 0111	1.0675E+09	0.9430	9)	(15,15)	30	11111001101000001001111111 10001	1.0470E+09	0.7772
10)	11111100110000001001111111 10111	1.0601E+09	0.8801	10)	(15,15)	30	11111001101000001001111111 1001	1.0470E+09	0.7772
11)	11111100100000001001111111 10111	1.0591E+09	0.8715	11)	(12,12)	5	11111111100000000001111101 0001	1.0716E+09	0.9807
12)	11111110100000001001111111 10001	1.0675E+09	0.9430	12)	(12,12)	5	11111110100000001001111111 0001	1.0675E+09	0.9430
13)	11101101100000001000001100 0101	9.9616E+08	0.4724	13)	(26,16)	15	11111100101000000010110111 0111	1.0596E+09	0.8757
14)	11111100100111000010111111 10000	1.0595E+09	0.8752	14)	(26,16)	15	11111001011000001010111111 00100	1.0460E+09	0.7694
15)	11111001101000001001111111 10001	1.0470E+09	0.7772	15)	(20, 1)	30	11111100100000001001001100 0101	1.0591E+09	0.8715
16)	11111100101000001010111111 00100	1.0596E+09	0.8758	16)	(20, 1)	30	11110001101001001010111111 0001	1.0135E+09	0.5615
17)	11111001111000001011001110 1111	1.0481E+09	0.7850	17)	(9,10)	30	11111110100000001001101110 0111	1.0675E+09	0.9430
18)	11111101100000111010111111 00100	1.0633E+09	0.9070	18)	(9,10)	30	11111100110000001001111111 0111	1.0601E+09	0.8801
19)	11111100100000001001111111 10000	1.0591E+09	0.8715	19)	(3, 8)	14	11111101100000001001111111 0100	1.0633E+09	0.9066
20)	11111100100000001001001100 0101	1.0591E+09	0.8715	20)	(3, 8)	14	11111100100000001001101100 0100	1.0591E+09	0.8715
21)	11111001111000001010111111 10000	1.0481E+09	0.7850	21)	(1,26)	8	11111001011000000000111011 0001	1.0460E+09	0.7694
22)	10111001111001001010111111 00100	7.7969E+08	0.0408	22)	(1,26)	8	11110001101001001010111111 1111	1.0135E+09	0.5615
23)	11111110100000001001111111 10001	1.0675E+09	0.9430	23)	(18,20)	30	11101101100000111010111111 10100	9.9621E+08	0.4726
24)	11111100100000001001111111 10111	1.0591E+09	0.8715	24)	(18,20)	30	11111100100000001001001000 0101	1.0591E+09	0.8715
25)	11111111100000001001111111 00100	1.0717E+09	0.9807	25)	(14,30)	3	11111100100110000010111110 0000	1.0595E+09	0.8747
26)	11111001011000000010111111 10111	1.0460E+09	0.7694	26)	(14,30)	3	11111100100111000010111101 0100	1.0595E+09	0.8752
27)	11111100100111000110111111 10000	1.0595E+09	0.8752	27)	(23, 3)	18	11111100100100001001111111 10001	1.0593E+09	0.8736
28)	01111001100001001000001100 0101	5.0968E+08	0.0006	28)	(23, 3)	18	11111110110000001001111111 10100	1.0685E+09	0.9523
29)	11111100110000001001101111 10001	1.0601E+09	0.8801	29)	(24, 2)	30	11111100100001001000011111 10111	1.0591E+09	0.8720
30)	11111100100110000010111111 00100	1.0595E+09	0.8747	30)	(24, 2)	30	11111100100010001001111111 10011	1.0592E+09	0.8726

Note: Generation 7 & Accumulated Statistics: max=0.9807, min=0.0405, avg=0.8100, sum=24.2997, nmutation=201, ncross= 71

FIGURE 3.18 SGA run, generation report $t = 6$–7.

Comentários

- Referências
  - Goldberg, D. E. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Pub. Co., 1989
    - Capítulo 3