Introdução ao Algoritmo Genético

Sadao Massago

Agosto de 2013

1 Introdução

O algoritmo genético é um método de otimização bio insperado, desenvolvida por John Henry Holland em 1975. Segundo a teoria evolucionária de Charles Darwin, os seres sofre mutações e os mais aptos (melhores) vão sobreviver. Então investigar como ocorre tal seleção natural e efetuar simulação permite gerar dados com características desejadas. Neste texto será abordado sobre algoritmos genéticos, usando como exemplo, casos de cromossomos codificados pelos números reais.

2 Modelo simplificado da evolução genética e otimização.

As características físicas dos seres vivos são codificados geneticamente pelos cromossomos formados pelas palavras (sequencias de alfabetos) de 4 tipo de proteínas (alfabetos). A sequencia destes códigos caracteriza a espécie e suas características físicas de um individuo.

Estes cromossomos são consequências de cruzamento dos cromossomos dos pais, acrescidas de uma mutação.

Segundo a teoria evolutiva de Charles Darwin, os indivíduos que apresentar características mais aptas para o ambiente tem mais chance de sobreviver e deixar seus herdeiros. Assim, pelo decorrer das gerações, a população de uma espécie evolui para melhor adaptar ao ambiente onde reside.

Podemos escrever a simulação deste processo evolutivo em termos de algoritmos como segue.

```
repete
  gera novo individuo
```

```
  avalia a aptidao do individuo
  escolhe os mais aptos
fim repete
```

O ponto mais importante é como simplificar e adaptar o fenômeno evolutivo sem perder a sua eficiência evolutiva. É importante lembrar que estamos tentando obter um algoritmo inspirado na teoria evolutiva e não, um simulador de evolução de um ser vivo. O processo precisa ser simples e o resultado deve ser eficiente.

Antes de mais nada, é necessário saber qual problema o sistema evolutivo resolve. Como sobrevive os mais aptos (melhor), ele é associado ao problema de otimização (obter maior ou menor valor da função). No entanto, ele costuma funcionar independente do tipo de função. Assim, algoritmo genético e denominado de método meta-heurístico para otimização e é amplamente empregado nos casos em que a implementação dos métodos de otimização tradicional não é apropriada ou complicada.

Para este texto, assumiremos que `drand` é uma função que retorna valores reais aleatórios entre 0 e 1 e `rand(N)` retorna inteiro aleatório entre 1 e N. Além disso, o vetor de tamanho N é assumido que está indexado de 1 a N.

3 Codificação do problema

Inicialmente, o problema precisa ser codificada. O problema apropriada para aplicar o algoritmo genético é o problema de otimização, isto é, obter o valor mínimo (ou máximo) sujeito a alguma condição específica.

Analisando o problema, determina-se os parâmetros numéricos e suas condições que caracteriza uma solução factível (solução que satisfaz a condição do problema). Esta sequência de números é denominado de cromossomos. Inicialmente, somente os cromossomos formados pelos números inteiros foram usados, mas com o tempo, os números reais também começam a ser usados com frequência.

Para servir de exemplo, vamos supor que queremos obter o mínimo da função $f(x) = \sum x_i^2$. Em geral, dado o problema de encontrar o mínimo de $f : X \to Y$, $x \in X$ é o cromossomo. O cromossomo costuma ser sequências de números, sendo armazenados no vetor dos números. Um individuo codificado armazena o cromossomo e as informações relevantes obtidos a partir dele. No caso de $f : X \to Y$, o individuo codifica $x \in X$ e $z = f(x)$.

Por exemplo, no nosso exemplo, um individuo pode armazenar o ponto x e o valor $z = f(x)$. Vamos codificar o cromossomo no vetor x e seu valor na variável z.

```
individuo
  x : vetor de tamanho N dos reais
```

```
  z : real
fim individuo
```

4 população

Uma população é o conjunto dos indivíduos. Cada individuo é representado pelos seus cromossomos. Em geral, podemos representar uma população como sendo vetor de indivíduos.

```
populacao : vetor de tamanho POP_SIZE de individuo
```

Em cada iteração do algoritmo genético, uma nova população será gerada a partir do atual, através do chamado operador genético. A forma como gera a nova população varia de caso por caso, devendo escolher a forma mais apropriada para cada problema. No entanto, existem duas formas opostas importantes: *steady-state* e *generacional*.

No steady-state, uma nova população é obtido pela população antiga, trocando com o novo individuo gerado somente quando ele for melhor que pior da população. Em termo operacional, gera os novos indivíduos e troca com o pior da população, caso ele for melhor. Caso não supere o pior, novo individuo será descartado. A ideia é juntar os novos indivíduos na população, ordenar pela aptidão e truncar pelo tamanho da população.

No generacional, exceto uma seleta de indivíduos denominados elites, todos serão trocados pelo novo individuo. Se não tiver os elites, seria próximo ao que acontece na natureza.

5 Inicialização, evaluação e ordenação

Inicialmente, gera os indivíduos iniciais da população aleatoriamente. Para nosso exemplo, vamos supor que o tamanho da população é POP_SIZE. Então gera POP_SIZE de n-uplas de números sorteados entre A e B. Para cada par de pontos, calcula o valor de f.

A população é inicializado pela função chamado novo_individuo

```
novo_individuo
  para i=1..N
    individuo.x[i] = A+(B–A)*drand
  fim para
```

```
  individuo.z = f(individuo.x)
  retorna individuo
fim novo_individuo
```

Agora ordenamos o individuo em termos dos valores de f. Como queremos o menor valor de f, o melhor individuo fica no começo da lista.

```
inicializa
  para k=1..POP_SIZE
    populacao[k] = novo_individuo
  fim para
  ordena populacao
fim inicializa
```

6 Operador genético: gerando novos indivíduos

Existem duas formas básicas de gerar nova população, mas eles usam o mesmo operador básico.

Elitismo (clone dos melhores): Alguns indivíduos do começo da lista entra para nova população. O elitismo é essencial para o caso generacional a fim de evitar que a população piore. No caso de steady-state, isto ocorre naturalmente e não haverá elitismo explicito.

Cruzamento (recombinação): Escolhe-se dois indivíduos na população, denominado de pai. O novo individuo é obtido, combinando os cromossomo de cada um deles. Existem várias formas de combinar os cromossomos, resultando diversos operadores de cruzamento. Para nosso exemplo, vamos supor que cada gene é a média ponderada dos genes correspondentes dos pais, denominado de "Blend" ([2]).

```
cruzamento(pai1, pai2)
  para x=1..N
    t = drand
    individuo.x[i] = t*pai1.x[i] + (1-t)*pai2.x[i]
  fim para
  individuo.z = f(individuo.x)
  retorna individuo
fim cruzamento
```

Para cruzamento, é desejável que evite que pai1 seja diferente de pai2. Poderá fazer sem muita dificuldade e melhorar a eficiência do cruzamento, mas não será

obrigatória (pior dos casos, perderá uma parcela dos individuos obtidos por cruzamentos).

```
seleciona_pais(i1, i2)
  i1 = rand(POP_SIZE)
  i2 = rand(POP_SIZE-1)
  se i2 <= i1 entao
    i2 = i2 + 1
  fim se
fim seleciona_pais
```

Mutação (clone modificado): copia o individuo da população anterior e modifica um pouco o cromossomo. Como existe várias formas de modificar o cromossomo, haverá diversos operadores de mutação. Para nosso exemplo, vamos supor que a mutação ocorre em um dos genes do cromossomo, com pequena perturbação, com máximo de L para cima ou para baixo.

```
mutacao(pai)
  individuo = pai
  i = rand(N)
  individuo.x[i] = individuo.x[i] + 2*L*(drand-0.5)
  individuo.z = f(individuo.x)
  retorna individuo
fim mutacao
```

Novos (opcional): A fim de introduzir indivíduos variados, poderá também colocar indivíduos gerado independente da população anterior. Isto melhora a busca inicial, principalmente quanto o problema apresentar mais de um mínimo local. Este operador já foi apresentado para gerar a população inicial

7 Gerando uma nova população

No caso de steady-state ([4]), o individuo gerado entra na população somente quando for melhor que o anterior. Então, gera o individuo, avalia e insere se for melhor que o pior da população.

```
steady_state
  para k=1..NUM_MUTACAO
    i = rand(POP_SIZE)
    item = mutacao(populacao[i])
    se (item.z < populacao[POP_SIZE].z) entao
```

```
      trocar ultimo por item e reordenar
    fim se
  fim para
  para k=1..NUM_CRUZAMENTO
    seleciona_pais(i1, i2)
    item = cruzamento(populacao[i1], populacao[i2])
    se (item.z < populacao[POP_SIZE].z) entao
      trocar ultimo por item e reordenar
    fim se
  fim para
  para k=1..NUM_NOVOS
    item = novo_individuo
    se (item.z < populacao[POP_SIZE].z) entao
      trocar ultimo por item e reordenar
    fim se
  fim para
fim steady_state
```

No generacional ([4]), toda população será trocada. Após a troca, efetuará a ordenação da população de acordo com a aptidão (que no caso do exemplo, será valor de f).

É assumido que NUM_ELITISMO + NUM_MUTACAO + NUM_CRUZAMENTO + NUM_NOVOS = POP_SIZE.

```
generacional
  contador = 1
  para k=1..NUM_ELITISMO
    nova_populacao[k] = populacao[k]
    contador = contador + 1
  fim para
  para k=1..NUM_MUTACAO
    i = rand(POP_SIZE)
    nova_populacao[contador] = mutacao(populacao[i])
    contador = contador + 1
  fim para
  para k=1..NUM_CRUZAMENTO
    seleciona_pais(i1, i2)
    nova_populacao[contador]
      = cruzamento(populacao[i1], populacao[i2])
```

```
    contador = contador + 1
  fim para
  para k=1..NUM_NOVOS
    nova_populacao[contador] = novo_individuo
  fim para
  ordena nova_populacao
fim generacional
```

8 Critério de parada e limpeza

Durante a iteração, poderá gerar um individuo existente, ou muito próximo do existente. Neste caso, os indivíduos da população tende a ser tudo igual ou parecidos, reduzindo a sua eficiência. Para evitar a redução de indivíduos variados na população, é bastante recomendável que implemente a rotina de limpeza na qual substitui os indivíduos iguais ou muito parecidos por um novo qualquer, eliminando do senário. Isto reduz o problema da convergência para o mínimo local em vez do mínimo global. Não esquecer de ordenar após limpar.

```
limpeza
  para k=1..POP_SIZE
    para i=(k+1)..POP_SIZE
      se populacao[k] = populacao[i] entao
        populacao[i] = novo_individuo
      fim se
    fim para
  fim para
  ordena populacao
fim limpeza
```

A rotina de limpeza costuma ser chamado de tempo em tempo e não a cada iteração.

A avaliação para parada pode ser feita por número de iterações, resultado da aptidão, desvio padrão da parte de população, etc. É importante que escolha a forma adequada de avalização do critério de parada.

9 Como tudo isso funciona

Vamos exemplificar o funcionamento do algoritmo genético para nosso problema que é minimizar a função $f(x) = \sum x_i^2$ para $N = 2$. Assim, $f(x) = x_i^2 + x_2^2$

Código do individuo será:

cromossomo: x_i, $i = 1, 2$

aptidão: $z = f(x)$

Vamos usar 1 elitismo, 1 mutação, 1 cruzamento e 1 novos, com POP_SIZE=4.

População inicial:

Sorteando 4 indivíduos e calculando seu valor, temos

x_1	x_2	z
0.5	0.2	0.29
-1.5	-0.2	2.29
1.2	-0.6	1.8
1	1.5	3.25

Ordenando por z, temos

x_1	x_2	z
0.5	0.2	0.29
1.2	-0.6	1.8
-1.5	-0.2	2.29
1	21.5	3.25

Uma iteração do steady-sate

Suponha selecionado o terceiro individuo e perturbando, tendo

$x_1 = -2$, $x_2 = -0.5$. Então $z = 4.25$.

Como ele é pior que o último individuo, ele não vai entrar na população. Assim, a população continua sendo o que era antes.

Agora suponha que o primeiro e segundo foi escolhido como pai1 e pai2 para o cruzamento. Sorteando valores entre seus genes, suponha que obtemos

$x_1 = 1$, $x_2 = 0.1$. Então $z = 1.01$, sendo melhor do que o último elemento.

Então substituindo o último e reordenando, teremos

x_1	x_2	z
0.5	0.2	0.29
1	0.1	1.01
1.2	-0.6	1.8
-1.5	-0.2	2.29

Agora suponha que foi gerado um individuo novo $x_1 = 1.6$, $x_2 = 1.3$. Então $z = 4.25$, pior do que o último elemento. Então a população será mantida.

Note que, se o elemento gerado não for melhor que o último, a população anterior será mantida. Se for gerado o melhor que o último individuo, será trocado por último

e reordenado.

Uma geração do generacional.

Vamos supor mesma população inicial

Como tem um elitismo (sempre deve ter no mínimo 1), após o elitismo, a nova população ficaria

x_1	x_2	z
0.5	0.2	0.29

Agora, vamos supor que foi sorteado terceiro elemento para mutar e sofreu mesma mutação que o stady-state.

$x_1 = -2$, $x_2 = -0.5$. Então $z = 4.25$.

Apesar dele ser pior do que o último individuo, ele entra na nova população.

x_1	x_2	z
0.5	0.2	0.29
-2	-0.5	4.25

No sorteio de cruzamento, suponha sorteado o primeiro e segundo como no caso do steady-state. O pai1 e pai2 será escolhido da população anterior e não da população nova. Então serão cruzamentos de $x_1 = 0.5$, $x_2 = 0.2$ com $x_1 = 1.2$, $x_2 = -0.6$. Assim, vamos supor que foi gerado o individuo $x_1 = 1$, $x_2 = -0.2$ com $z = 1.04$. A nova população ficaria

x_1	x_2	z
0.5	0.2	0.29
-2	-0.5	4.25
1	-0.2	1.04

Gerando o individuo restante como sendo novo individuo

x_1	x_2	z
0.5	0.2	0.29
-2	-0.5	4.25
1	-0.2	1.04
0.2	1.2	1.48

Ordenando, teremos

x_1	x_2	z
0.5	0.2	0.29
1	-0.2	1.04
0.2	1.2	1.48
-2	-0.5	4.25

Observação: No steady-state, a população é atualizada para cada individuo gerado, mas no generacional, população é atualizado somente quando tudo estiver gerados. Estes dois métodos de geração são extremos opostos e existem varias variações intermediarias.

10 Considerações finais

Para o algoritmo genético, é recomendado ter um bom gerador de números aleatórios e é importante saber lidar com distribuição que não é necessariamente uniforme.

A seleção dos indivíduos costuma ser feita usando uma roleta que prevaleça os indivíduos de menor custo (ou maior aptidão). Em geral, os pesos na roleta são custos ou a posição, favorecendo o menor valor. Devido a facilidade de implementação, poderá sortear os indivíduos usando números aleatórios com a distribuição exponencial em termos de posição, em vez da roleta.

Para a mutação, costuma usar distribuição normal em vez de sortear uniformemente o valor num intervalo fixo.

Para melhorar a eficiência computacional no cálculo da distribuição exponencial e normal, poderá substituir pela distribuição dada pelas funções bias e gain descrito em [5].

Referências

[1] Coelho, Leandro dos S., "Fundamentos, Potencialidades e Aplicações de Algoritmos Evolutivos", SBMAC, 2003.

[2] Haupt, Randy L., "Pratical Genetic Algoritmos", John Wiley & Sons, Inc, 2004.

[3] Banzhaf, Wolfgang et. al., "Genetic Programming (an introduction)", Morgan Kaufmann Publishers, Inc, 1998.

[4] Barbosa, Helio J. C., "Introdução aos Algoritmos Genéticos (XX CNMAC)", SBMAC, 1997.

[5] Schlick, Christophe, "Fast Alternatives to Perlin's Bias and Gain Functions", Graphics Gems IV, Academic Press, pp. 401-403, 1994.