IMPLEMENTANDO UM ALGORITMO GENÉTICO NA APROXIMAÇÃO DE ZEROS E PONTOS CRÍTICOS DE POLINÔMIOS

Fábio Santos de Goes*

Resumo

Este trabalho apresenta um pouco do know-how adquirido e dos resultados obtidos em um projeto final de curso desenvolvido pelo autor, no âmbito do curso de graduação em Informática e Tecnologia da Informação do IME e orientado pelo prof. Carlos A. de Moura. Apresentam-se aqui alguns aspectos práticos referentes à implementação de uma ferramenta numérica não determinística capaz de aproximar pontos críticos de polinômios, tais como representação numérica, métodos de sampling e operadores genéticos. Uma pequena parcela dos resultados obtidos é oferecida a título de ilustração.

1. Introdução

Neste texto pressupomos que o leitor esteja familiarizado com algoritmos genéticos. Estes constituem um ramo da área de Sistemas Complexos, e sua mecânica e suas aplicações têm sido bastante pesquisadas nos últimos anos. Discutimos brevemente uma implementação prática apresentada como projeto de graduação na Universidade do Estado do Rio de Janeiro, na forma de uma ferramenta numérica capaz de aproximar máximos, mínimos e zeros de funções polinomiais de coeficientes inteiros.

Este tipo de aplicação é discutido no trabalho em [Michalewicz, 1996], p18-22, e em maior detalhe no trabalho [Goldberg, 1989], cap 3. Nesta última referência, após dois capítulos de uma discussão introdutória, Goldberg apresenta o **SGA**, um arcabouço de rotinas em **PASCAL** que constituem o cerne de um algoritmo genético capaz de maximizar funções do tipo x^n , com n variando de 1 a 31. Neste texto é discutida a implementação de uma

Palavras-chave: sistemas complexos, algoritmos genéticos, métodos numéricos não-determinísticos
Departamento de Informática, Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro,
ordo.equester@gmail.com

ferramenta batizada de **ALGAS**, acrônimo para **AL**goritmos **Ge**néticos **Aplicado**S. O **ALGAS** é capaz de determinar máximos e mínimos locais em funções polinomiais da forma $a_m X^n + a_{m-1} X^{n-1} + a_{m-2} X^{n-2} + \dots + a_0 X^0$, onde os coeficientes a_i e n são inteiros.

A mecânica do programa consiste em analisar o espaço amostral correspondente a um trecho da imagem de uma função que corresponda a um domínio previamente definido. Seu resultado é o maior ou menor valor ali encontrado. Para que o usuário possa **perceber a diferença** entre um ponto de máximo local e o maior valor atingido pela função dentro do intervalo pesquisado, a ferramenta produz gráficos do polinômio investigado, em escala variável. Uma vez que o usuário perceba claramente a existência de um máximo ou mínimo local – o que pelo gráfico é trivial – basta delimitar *grosso modo* um intervalo do domínio que inclua esse ponto e o programa o encontrará **através da execução de um algoritmo genético**.

Adicionalmente, a ferramenta também é capaz de encontrar pelo menos uma raiz em um intervalo, caso esta exista. A estratégia utilizada não implica nenhuma alteração do algoritmo: as raízes são localizadas sempre a partir da minimização do **módulo** da função em um intervalo delimitado. Se a função cruzar o eixo das abscissas naquele intervalo, é evidente que este “mínimo” será uma raiz.

A Representação Numérica

A primeira questão referente à representação numérica interna do programa é como representar continuidade a partir de elementos discretos, como uma *string* de *bits*. A solução usual é dividir a unidade em pequenos segmentos que podem ser tornados tão pequenos quanto desejarmos. Conforme tais segmentos tendam a zero, tanto mais perfeita a continuidade obtida. Começamos com um exemplo simples: com um *bit* podemos representar dois intervalos de continuidade. O valor zero poderia representar um número entre 0 e 0,5 e o valor um poderia representar um número entre 0,5 e 1.

Seguindo esta lógica, com 2^{10} *bits* podemos particionar e representar 1024 intervalos da unidade, obtendo assim 3 casas de precisão decimal. Se precisarmos representar 20

unidades com 3 casas de precisão, deveremos lançar mão de **pelo menos** 20 x 1024 *bits*; se quisermos representar 3 unidades com quatro casas de precisão, 2^{15} *bits* $\approx 3 \times 2^{14}$ serão suficientes.

No programa foi utilizada uma representação binária em complemento de 2 com ponto fixo. É possível aumentar a precisão diminuindo o espaço de busca e vice-versa. A adoção de uma representação numérica potencialmente maior que os espaços de busca forçou a inclusão de penalidades (*constraints*) no algoritmo, para garantir que as soluções encontradas fiquem dentro do espaço de busca definido pelo usuário. A penalidade constitui somar ou subtrair o quadrado da imagem da função, conforme o objetivo do programa, e é uma estratégia usual para implementar restrições em problemas de programação linear [Goldberg, 1989].

Os principais tipos de dados presentes são o tipo polinômio e o tipo cromossomo. O polinômio foi de fato pensado como um tipo abstrato de dados com métodos correlatos (como adicionar termo, avaliar $f(x)$, destruir etc.), e a invocação de cada método deve incluir, necessariamente, a passagem de um ponteiro para a estrutura que será manipulada.

O tipo cromossomo armazena o genoma (a *string* de *bits* que representa os genes) de cada indivíduo. A população é uma variável global definida como um *array* de cromossomos; Junto da *string* de *bits* é armazenado também o valor em decimal dessa *string*, e a correspondente $f(x)$ para este valor em decimal. Calcular estes valores e armazená-los junto aos *bits* evita que se tenha que voltar a calcular estes valores cada vez que o programa precisar checar o valor decimal representado pelo genoma, o que pode ser deselegante do ponto de vista do algoritmo mas é computacionalmente importante.

A Modelagem da Seleção

A seleção dos elementos para a reprodução se dá de duas possíveis maneiras: através de um sistema de *ranking*, em que os melhores indivíduos têm maiores chances de obter um ou mais *crossovers*, desconsiderando o quanto esta *fitness* se sobressai em relação à *fitness* média. A outra possibilidade é através do mesmo tipo de “giro de roleta” do **SGA**, onde a possibilidade de seleção é perfeitamente proporcional à *fitness*.

. Há um parâmetro, “*crossover chance*”, que evita que indivíduos selecionados possam obter um *crossover* imediatamente. Conforme se aumenta o valor deste parâmetro, mais os melhores indivíduos tendem a monopolizar os *crossovers*. Se o parâmetro tende a zero a seleção é aleatória. Em geral foram usados valores em torno de 0,25, o que visava evitar uma seleção perfeitamente elitista, preservando-se a diversidade genética por mais tempo (o que em termos matemáticos significa investigar regiões distintas do espaço amostral ao invés de concentrar-se de imediato em uma única área que pareça mais promissora).

A seleção por *ranking* visa evitar o impacto que “super indivíduos” [Goldberg 1989] possam exercer sobre a população, monopolizando os *crossovers* e causando uma convergência prematura do algoritmo. Em alguns momentos este tipo de seleção apresentou resultados claramente melhores que a seleção pela roleta clássica, mas aconselha-se cautela nas conclusões visto que a **representação interna** dos números no **ALGAS** é diferente da adotada pelo **SGA**.

O Crossover

A operação de *Crossover* é definida exatamente segundo os moldes tradicionais, e sua implementação não possui detalhes especiais dignos de nota. O *Crossover* é feito invocando-se o procedimento *popMate* (f, g, h), onde f, g e h devem ser, respectivamente, os índices dos dois indivíduos no *array* da população, e h é o ponto de *crossover*. Em tese, é possível variar o ponto de *crossover* durante a execução do algoritmo, mas esta funcionalidade não foi utilizada.

A Mutação

A literatura tradicional [Goldberg 1989], [Michalewicz 1996] costuma atribuir uma pequena probabilidade de *crossover* a cada gene. Sem divorciar-nos deste modelo foi adotada uma estratégia computacionalmente mais interessante: ao invés de gerar um número aleatório para cada gene de cada cromossomo, e a partir deste sorteio decidir se haverá uma mutação ou não, define-se de antemão quantos genes deverão sofrer mutação, de acordo com uma taxa

percentual (em relação ao total da população) definida pelo usuário, e a partir daí são gerados dois números aleatórios para cada mutação: um, definindo o cromossomo afetado e o outro, definindo o gene que sofre um *NOT*.

Assim, as implementações tradicionais executariam n operações para decidir quais genes sofreriam mutações, sendo n número total de genes da população, ao passo em que o **ALGAS** executa $(n/100)*p*2$ operações, o que é vantajoso quando p é pequeno.

Conclusões

Várias conclusões foram obtidas a partir deste trabalho. Entre elas:

A distribuição inicial da população é importante para a eficiência do algoritmo

Em um dos testes foi rodado um problema de maximização com uma população inicial especialmente ruim e foram necessárias **8890 iterações** até que o algoritmo encontrasse a solução esperada. Por outro lado, com uma população uniformemente distribuída ao longo do intervalo de busca, o algoritmo costuma resolver o problema com uma aproximação de duas casas decimais em **menos de cem iterações**.

A presença de mutação é fundamental para explorar novas regiões do espaço de busca (i.e., schemata) não representados na população inicial

Quando o programa é executado sem possibilidade de mutações, o algoritmo não é capaz de extrapolar o espaço de busca delimitado pela distribuição inicial da população. Neste caso o programa acha resultados não muito melhores que os melhores indivíduos presentes em uma população inicial. Esta observação vem de encontro à proposição de que “*Reproduction alone does not promote exploration of new regions of the search space*” [Goldberg, 1989], p. 31.

Funções de penalização funcionam com eficiência

Surpreendentemente, não foi verificado nenhum aumento importante no número de iterações necessárias para o algoritmo obter uma solução adequada, mesmo na presença de funções de penalização. Supomos que as funções de penalização não somente **causem o descarte de resultados inválidos** mas inclusive **desencoragem a exploração de espaços não-permitidos** pelo algoritmo.

O funcionamento do algoritmo genético é não-determinístico

Um algoritmo genético, tal como implementado neste projeto, constitui um **método numérico** e não uma solução analítica. Autores como Michalewicz lembram que “não há garantia alguma” de que os melhores indivíduos da última iteração venham a ser as melhores soluções. Embora o mínimo de uma função determinada seja sempre o mesmo, o número de passos e os próprios passos que o algoritmo executa para descobrir esse valor variam. Quando a saída produzida é insatisfatória, o aumento do número de iterações permitidas ao algoritmo costuma mostrar-se suficiente para que o programa produza um resultado mais preciso.

Resultados

Para evitar exceder o espaço destinado a este texto serão mostrados apenas alguns poucos resultados para que se tenha uma idéia da capacidade do programa. Uma série de resultados mais interessante se encontra ao final do texto do projeto defendido, e há uma cópia disponível na Biblioteca do IME.

$-x^2-4x+8$

achar raízes do polinômio

1.46676599913917 $f(x) = 0.0184664927874092$, com 211 iterações

1.46410579547228 $f(x) = 0.0000289622246256$, após 27561 iterações

-5.46862792945467 $f(x) = 0.0313797129929867$, com 175 iterações

-5.46409600500192 $f(x) = 0.0000388681297351$, após 14031 iterações

maximizar o mesmo polinômio

-2.00000003028254 $f(x) = 12$, 276 iterações

achar uma raiz para x^3+x^2-x+1

-1.83827236837533 $f(x)=0.00554440714462512$ (ranking, 999 iterações)

-1.83927823706472 $f(x)=0.00004659696330611$ (ranking, 1.000.000 iterações)

-1.87499618530273 $f(x)=0.201149761744091$ (roleta, 999 iterações)

-1.84375 $f(x)=0.024505615234375$ (roleta, 1.000.000 iterações)

(notar que a ferramenta usou algoritmos diferentes para o *sampling*)

Referências Bibliográficas

1. **MICHALEWICZ, Zbigniew** *Genetic Algorithms + Data Structures = Evolution Programs*. New York, Springer-Verlag, 1996

2. **GOLDBERG, David Edwar** *Genetic Algorithms in Search Optimization and Machine Learning*. Redding, Addison-Wesley, 1989

4. **MANTOVANI, A. & FREIRE, J.** *Algoritmos Genéticos Conjugados a Heurísticas Clássicas para a Solução do PCV*. Rio de Janeiro, Universidade do Estado do Rio de Janeiro, Projeto Final de Graduação, IME, 1999

5. **HOLLAND, John H.** *Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, 1975

6. **DE JONG, K. A.** *An analysis of the Behavior of a Class of Genetic Adaptive Systems (Doctoral dissertation, University of Michigan)*. Abstract International 36(10), 514B, University microfilm # 76-9381. 1975

7. **DE JONG, K. A.** *Genetic Algorithms: A ten-year perspective in Grefenstette, JJ (editor). Proceedings of the First International Conference in Genetic Algorithms*, Lawrence Erlban Associates, Hillsdale, 1985