Biblioteca modelo C++ para método de otimização estocástica NSGA-II parametrizado, com aplicação em função bimodal multiobjetivo

Hamilton José Brumatto[1]

[1]Universidade Estadual de Santa Cruz - UESC

hjbrumatto@uesc.br

Abstract. *NSGA II is an algorithm that allows finding a Pareto front that represents an optimum of a multiobjective function. In this work we proposed to create a C++ template library to facilitate the user to apply this algorithm in a function. We show in this work the results when varying the parameters and applying the algorithm in a bimodal function. As a result we have a functional library and an analysis of the variation of parameters in the NSGA II algorithm.*

Resumo. *O NSGA II é um algoritmo que permite encontrar uma frente de pareto que representa um ótimo de uma função multiobjetivo. Neste trabalho nos propusemos a criar uma biblioteca C++ usando modelos para tipos de dados para facilitar ao usuário aplicar este algoritmo em uma função. Mostramos neste trabalho os resultados ao variar os parâmetros aplicando o algoritmo em uma função bimodal.*

1. Introdução

O NSGA (*nondominated sort genetic algorithm*) em sua versão II[Deb et al. 2002] é um algoritmo genético multiobjetivo evolucionário ou seja, um algoritmo que cria uma população de proposta de solução para as funções e por um critério de seleção (ordenação por dominância) separa as melhores soluções para que estas gerem nova população. Neste modelo evolucionário espera-se que após algumas gerações a solução venha a atingir um conjunto ótimo para as funções. Em funções multimodais, estas soluções podem ser ótimas locais ou globais.

Os algoritmos genéticos buscam mimetizar a teoria da evolução, ou seja, são criadas populações através de parâmetros evolutivos, ao longo do tempo novas gerações convergem, por critérios de aptidão à sobrevivência, para uma situação que otimize os objetivos iniciais. A população deve convergir para uma frente de pareto ótima, que pode ser global, ou local.[Barbosa et al. 2010].

Aqui vamos dar destaque ao algoritmo NSGA II, embora existam outros algoritmos genéticos. A escolha deste algoritmo é estritamente pessoal, se queremos construir uma biblioteca, levando à esta vários algoritmos, precisamos começar com um, o NSGA II foi a escolha da vez. Sendo um algoritmo, uma receita de procedimentos computacionais, a sua aplicação não deve depender das funções. A linguagem C++ através de "modelos" [Vandevoorde and Nicolai 2010] permite a construção de algoritmos genéricos que independem das funções objetivos. O usuário não tem de se preocupar em reescrever o algoritmo, apenas indicar a função multiobjetivo escolhida e os parâmetros desejados.

Neste trabalho propomos um algoritmo genérico baseado em *templates* C++ específico para o método NSGA-II aplicável a funções multiobjetivo multimodais. O trabalho de Shen et all[Shen et al. 2015] propõe uma biblioteca para otimização em C++ que seja aplicada a funções multiobjetivos, porém não descreve formas de parametrização.

Na seção 2 descrevemos o algoritmo NSGA-II, na seção 3 o conceito de "bibliotecas modelos" e algoritmos genéricos, bem como a construção do NSGA-II, na seção 4 aplicamos esta biblioteca a uma função multiobjetivo bimodal onde buscamos, a partir dos parâmetros usados, demonstrar a convergência do algoritmo às soluções ótimas e, finalmente na seção 5 apresentamos as conclusões deste trabalho.

2. NSGA-II

Não há intenção aqui de fazer uma vasta comparação dos vários algoritmos genéticos, nosso interesse é realmente a implementação do NSGA-II. Nas notas de aula do prof. Sadao Massago[Massago 2013] podemos encontrar uma rápida descrição do algoritmo genético:

```
repete
  gera nova população
  avalia a aptidão da população
  escolhe os mais aptos
fim repete
```

Dentro desta proposta, uma população para o problema multiobjetivo é um conjunto de soluções, não necessariamente ótima. A avaliação desta população seria baseada em um ordenamento das soluções de forma que as soluções melhores precedem as soluções com piores resultados. A escolha dos mais aptos vem de um corte desta população para um conjunto de melhor precedência neste ordenamento. Por fim a geração da população seria a geração de novos pontos que se assemelham aos pontos que já possuem uma solução melhor.

Espera-se que ao longo de gerações, a população se restrinja a uma frente de pareto que otimize a função. Vamos aos passos do algoritmo genético.

2.1. Gera nova população

Vimos que a geração de uma população depende de uma escolha de uma população anterior com melhor qualidade. Mas deve existir uma primeira geração, e esta primeira geração pode ser construída a partir de uma distribuição uniforme dentro das restrições de domínio das funções multiobjetivos. Um dos parâmetros definidos para o algoritmo é o tamanho da população.

Os algoritmos genéticos são inspirados nos mecanismos de evolução natural e recombinação genética, buscando através do princípio Darwiniano a sobrevivência dos mais aptos. Desta forma os descendentes provém de dois mecanismos possíveis de criação: cruzamento (recombinação genética) e mutação (evolução natural). A ideia no cruzamento é carregar as características dos genitores, sendo estes aptos o que vier gerado por estes também tem chances de ser. Na mutação busca-se com pequena alterações na característica do indivíduo gerar um novo com melhor aptidão.

Neste trabalho há a proposição de representação da posição de cada indivíduo no diagrama de fase da função como característica genética. As novas gerações podem ser construídas a partir do intervalo entre dois indivíduos (cruzamento) ou mesmo da geração de um indivíduo em uma posição deslocada do indivíduo original (mutação).

2.2. Avalia a aptidão da população

Dada a população (original + novos indivíduos), é feita uma escolha por uma ordem parcial dos pontos. Uma ordem em questão de dominância. Desta ordem começam a surgir as frentes de pareto. Dizemos que um ponto é dominado por um ou mais pontos se houver pontos que oferecem um resultado melhor que este. Os pontos em uma frente de pareto não têm dominância entre si. O primeiro conjunto, ou a primeira frente, são os pontos nos quais ninguém domina. A segunda frente são pontos que somente possui um único outro ponto dominando, e assim sucessivamente. Sendo que os indivíduos da primeira frente são mais aptos que os da segunda e assim sucessivamente.

2.3. Escolhe os mais aptos

Definidas as frentes, precisamos escolher o conjunto de indivíduos mais aptos a partir destas frentes. Neste caso, o tamanho da população é um parâmetro importante. Os indivíduos estão em ordem parcial, e dentro de uma mesma frente, a ordem prevê a escolha de indivíduos afastados uns dos outros para maior diversificação. Restringimos agora a população aos mais aptos que passarão a gerar nova população.

2.4. Convergência do método

Este processo é iterativo, a cada geração espera-se que as frentes convirjam para os ótimos das funções objetivos. Se as funções forem multimodais, podemos até mesmo encontrar indivíduos em frentes que representam ótimos locais. Se não conhecemos de antemão as soluções determinísticas fica difícil saber se estamos encontrando um ótimo local ou um ótimo global. A solução é rodar várias vezes alterando os parâmetros do algoritmo. Um fato é importante cada vez que rodamos o algoritmo temos um resultado diferente dada a aleatoridade da distribuição e da geração de novos indivíduos. Os parâmetros para o algoritmo são:

- *Número máximo de iterações*
- *Tamanho da população*
- *Número de novos indivíduos*
- *Cruzamento*
- *Mutação*

O sucesso do método depende de um refinamento dos parâmetros. A solução ótima de um problema multiobjetivo acaba sendo uma linha (superfície ou hipersuperfície) (convexa) no diagrama $f_1 \times f_2 (\times f_3 \times f_4 ...)$[Coello et al. 2014] ou seja, mesmo se não temos uma solução determinística para a função, os pontos devem convergir na forma de uma hiper-linha. Se houver outros mínimos locais podem surgir outras linhas. Quando os pontos formam uma linha bem definida, podemos entender que houve uma convergência para o problema.

3. Biblioteca modelo do C++ e a implementação do algoritmo

A linguagem de programação C++ permite que trabalhemos com algoritmos genéricos e bibliotecas modelos. Tanto a linguagem C++ quanto a linguagem C aceitam como parâmetro uma referência a uma função. Desta forma o algoritmo NSGA-II pode ser aplicável a qualquer função multiobjetivo. O uso de modelos permite funções que operam sobre tipos variados. Um ponto é um vetor da dimensão da entrada da função multiobjetivo. Nesta implementação a biblioteca não será responsável por uma saída gráfica, e sim em gerar os pontos que representam a solução ótima. A restrição é que o tipo modelo, aceite o *operador[]* (at) retornando um *double* para uma determinada coordenada do ponto. Também é preciso aceitar o *operator*$<$ para comparação de desigualdade entre os valores dos objetivos.

3.1. O corpo principal do NSGA-II

Como vimos na seção 2 o corpo principal do NSGA-II nada mais é que uma cópia do algoritmo genético. Na listagem 1 vemos o corpo das iterações por geração da função *nsga_ii*. Em especial, *pop* é a população *pop_c* e *pop_m* a população dos indivíduos gerados por cruzamento e mutação. A função multiobjetivo é referênciada pelo ponteiro *fmobj*.

Listagem 1 Corpo de iteração por gerações da função *nsga_ii*

```
while(ger--) {
// Geração de novos indivíduos
    vector<individuo<T>>pop_c(n_crossover);
    pop_c = crossover<T>(pop, n_crossover);  // geração por crossover
    for(int i=0; i<(int)pop_c.size(); i++)
        pop_c[i].setCost(fmobj(pop_c[i].pos()));
    vector<individuo<T>>pop_m(n_mutation);
    pop_m = mutation<T>(pop, min, max, n_mutation, sigma, n_mu); // geração por mutação
    for(int i=0; i<(int)pop_m.size(); i++)
        pop_m[i].setCost(fmobj(pop_m[i].pos()));
    pop.insert(pop.end(),pop_c.begin(),pop_c.end()); // agregação das novas
    pop.insert(pop.end(),pop_m.begin(),pop_m.end()); // populações

// Avaliação da aptidão
    dominancy(pop); // classificação das frentes de pareto
    dist_aglomeracao(pop); // classificação por distância de aglomeração

// Escolha dos mais aptos
    sort(pop.begin(), pop.end(), globalSort<T>); // Ordena pela aptidão
    pop.erase(pop.begin()+n_pop,pop.end()); // restringe a população a n_pop
}
```

4. Função multiobjetivo bi-modal e resultados

Vamos aplicar o NSGA-II em uma função multiobjetivo (2 objetivos) e bimodal, possui dois ótimos um global e um local(1):

$$\min \begin{cases} f_1(x_1, x_2) \equiv x_1 \\ f_2(x_1, x_2) \equiv \dfrac{g(x_2)}{x_1} \end{cases} \qquad (1)$$
$$\text{onde: } x_1 > 0, 0 \leqslant x_2 \leqslant 1$$

E a função $g(x_2)$ é dada pela expressão em (2)

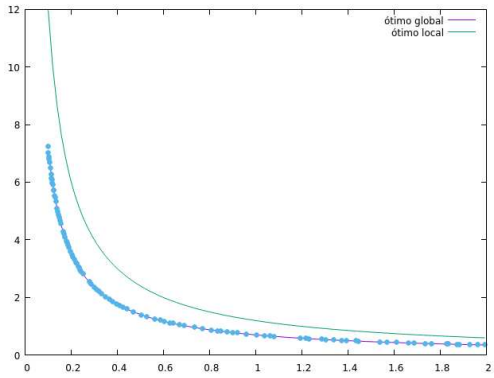

Figura 1. Convergência para o ótimo global do algoritmo

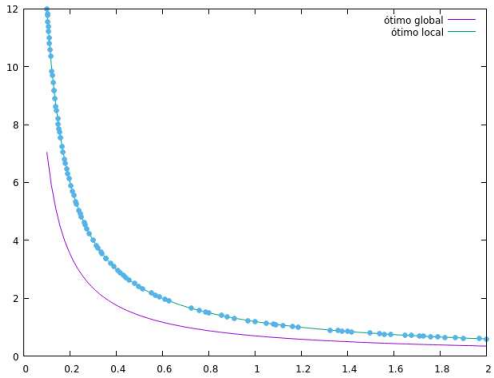

Figura 2. Convergência para o ótimo local do algoritmo

$$g(x_2) = 2 - exp\left\{-\left(\frac{x_2 - 0.2}{0.004}\right)^2\right\} - 0.8exp\left\{-\left(\frac{x_2 - 0.6}{0.4}\right)^2\right\} \quad (2)$$

A função $g(x_2)$ é multimodal com dois mínimos, um global $\overline{x_2} = 0.2$ com $g(\overline{x_2}) \approx 0.7057$, e um local $\underline{x_2} = 0.6$ com $g(\underline{x_2}) \approx 1.2$.

4.1. Resultados do algoritmo NSGA-II

Vimos que o resultado depende da parametrização e da aleatoriedade. Para esta função existem duas frentes importantes, do ótimo global e do local. Se houver pontos no ótimo global, nas novas gerações teremos mais pontos associados a este ótimo e o resultado deve convergir para o ótimo global. A Figura 1 mostra a convergência para o ótimo global, a Figura 2 para um ótimo local. Temos até mesmo uma convergência mista, na Figura 3, decorrente de uma escolha não ideal dos parâmetros para o algoritmo.

5. Conclusão

A proposta deste trabalho foi de criar uma biblioteca parametrizável para o algoritmo NSGA II, neste aspecto tivemos um ótimo resultado. O usuário da biblioteca somente precisa indicar a função multiobjetivo que irá trabalhar, bem como a restrição nas variáveis da função. A biblioteca também permite que as variáveis ou resultado da função seja de

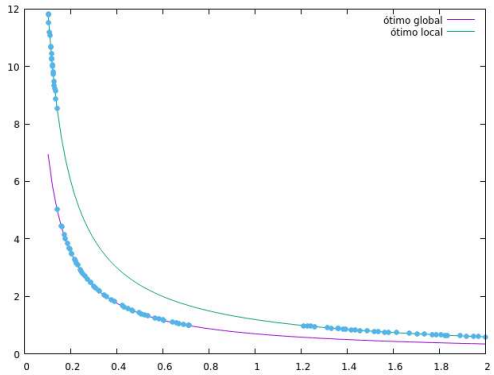

Figura 3. Convergência dupla do algoritmo

um tipo definido pelo próprio usuário, desde que este tipo implemente o operador *operator[]* e o operador *operator*< para comparar a ordem dos objetivos. Por exemplo podemos usar um *deque* de valores complexos tanto nas coordenadas como no resultado da função multiobjetivo.

Uma proposta futura seria criar esta biblioteca utilizado apenas a linguagem C, dentro das restrições da linguagem tal como existe no conjunto de bibliotecas científicas do GSL[Team] (*GNU Scientific Library)*) esta é uma biblioteca que não está disponível naquele acervo.

Referências

Barbosa, A. M., Ribeiro, L. d. C., and Arantes, J. M. d. O. (2010). Algoritmo Genético Multiobjetivo: Sistema Adaptativo com Elitismo. In *9th Brazilian Conference on Dynamics Control and their Applications*, pages 940–945, Serra Negra - SP.

Coello, C. C., Lamont, G. B., and Veldhuizen, D. A. V. (2014). *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2nd 2007 ed. edição edition.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.*, 6:182–197.

Massago, S. (2013). Introdução ao Algoritmo Genético. Publisher: Depto Matemática - UFSCar.

Shen, R., Zheng, J., and Li, M. (2015). A hybrid development platform for evolutionary multi-objective optimization. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 1885–1892.

Team, G. GNU Scientific Library Documentation.

Vandevoorde, D. and Nicolai, J. (2010). *C++ templates: the complete guide*. Addison-Wesley, 12th edition.