UNIVERSIDADE ESTADUAL DE MATO GROSSO DO SUL
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Relatório de desenvolvimento

Schwefel Function e Caixeiro Viajante

Trabalho acadêmico apresentado à disciplina de Programação Paralela e Distribuída, do curso de Bacharelado em Ciência da Computação como requisito parcial para obtenção da nota do terceiro bimestre.

Prof. Dr. Rubens Barbosa Filho

Gabriel de Biasi
Rodolpho Pivetta Sabino

Dourados - MS
Novembro de 2015

1. Introdução

1.1. Algoritmos Genéticos

Algoritmos Genéticos (AG) são técnicas de buscas utilizadas para encontrar soluções aproximadas em problemas de otimização e de buscas irregulares. Esses algoritmos são implementados de modo que, em uma população de representações abstratas de soluções é selecionada uma, em busca de soluções melhores. Um AG deve deve reconhecer caminhos que levam à soluções ótimas, combinando os princípios de sobrevivência baseados nos indivíduos melhores adaptados.

1.2. Schwefel Function

A função Schwefel é uma função complexa, com vários mínimos locais. A função ao ser plotada no gráfico, sua forma mostra que criar um algoritmo para solucionar uma busca pelo mínimo global é uma tarefa muito complicada. Na figura 1, é apresentado a função em três dimensões.

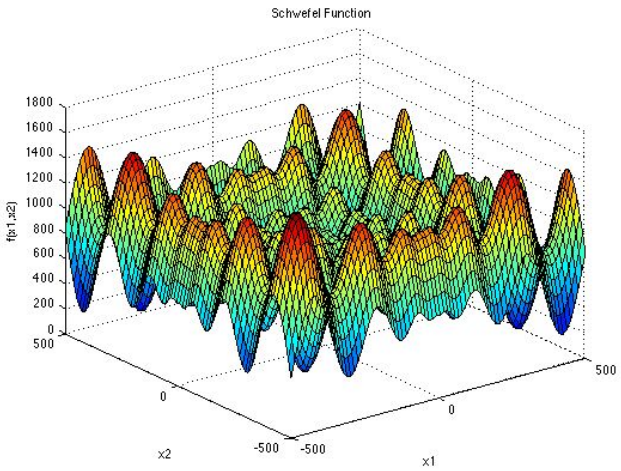

Figura 1 - Gráfico da Schwefel Function em três dimensões

A escrita da função é relativamente simples, comparado com outras funções de otimização. A figura 2 mostra a definição da função.

$$f(\mathbf{x}) = 418.9829d - \sum_{i=1}^{d} x_i \sin(\sqrt{|x_i|})$$

Figura 2 - Definição da função Schwefel

O domínio desta função é definido em um hipercubo restrito, onde $x_i \in [-500, 500]$, para i = 1, …, d. A variável d é o número de dimensões. A saída esperada do melhor indivíduo é f(x) = (0, 0, ..., 0) onde x é igual à (420.9687, 420.9687, …, 420.9687).

1.3. Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) é conhecido pela sua complexidade. Em um contexto simples, existe um caixeiro viajante que deseja visitar um conjunto de cidades no menor percurso possível para que cada cidade seja visitada apenas uma única vez. Este problema tem inúmeras aplicações práticas, como por exemplo a implementação de sistemas digitais, minimização de rotas de veículos, sequenciamento de atividades, entre outros.

Este problema, em termos computacionais, é um exemplo de problema de otimização combinatória. A formulação desse problema se dá por encontrar o número `R(n)` de rotas para o caso de `n` cidades, utilizando um raciocínio combinatório simples. Em um exemplo de `n = 4` cidades, a primeira e a última posição são fixas; a segunda posição pode assumir qualquer uma das 3 cidades restantes; a terceira posição pode assumir qualquer uma das 2 cidades restantes; e por fim a quarta posição assume a cidade restante (SILVEIRA, 2002).

O número de rotas pode ser calculado como `3 * 2 * 1 = 6`. Desse modo, como a primeira posição é fixa, esse problema pode ser descrito utilizando a notação fatorial, o que resulta em:

```
R(n) = (n - 1)!
```

A execução computacional desse problema em termos gerais e a fim de comparação, pode ser visualizada na tabela abaixo.

n	Rotas por segundo	(n - 1)!	Tempo
5	250 milhões	24	insignificante

10	110 milhões	362880	0.003 seg
15	71 milhões	87 bilhões	20 min
20	53 milhões	$1.2*10^{17}$	73 anos
25	42 milhões	$6.2*10^{23}$	470 milhões de anos

Fonte: Adaptada de (SILVEIRA, 2002)

Como pode ser visto, até mesmo para um número muito pequeno de cidades, o problema se torna extremamente grande, impossibilitando sua solução. Por essa razão o problema do caixeiro viajante é categorizado como NP-Completo, ou seja, faz parte de uma classe de problemas para os quais não existem algoritmos polinomiais em séries temporais determinísticos.

1.4. Objetivos

Este trabalho tem como objetivos implementar os algoritmos Schwefel Function e Caixeiro Viajante utilizando algoritmos evolutivos e realizar os testes de convergência das soluções.

2. Implementação

2.1. Materiais

Para a implementação dos algoritmos foi utilizada a linguagem *Python*, na versão 2.7. A conexão entre o *Python* e a *Message Passing Interface* (MPI) foi realizado utilizando o pacote *mpi4py*.

2.2. Middleware

O middleware pede que 6 (seis) métodos sejam implementados para que um problema possa ser encontrado utilizando o algoritmo. Sendo elas:

- **new_individual** - Precisa retornar uma lista de caracteres, '0' ou '1', restritamente. Esta lista deve representar um indivíduo do problema em representação binária.
- **get_fitness(individual)** - Precisa retornar um valor inteiro que represente o quão perto está este indivíduo da resposta desejada. Quanto mais próximo do **zero**, mais correta está a resposta.
- **is_finished(individual)** - Precisa retornar se o indivíduo passado por parâmetro é aceitável ou não como resposta do problema.
- **validate_individual(individual)** - Precisa retornar um valor booleano, dizendo se um indivíduo é válido para a confecção do problema ou não.
- **num_bits()** - Precisa retornar o número de *bits* utilizado para representar um indivíduo.
- **show(individual)** - Precisa retornar uma representação em texto do indivíduo.

Ao obter estes métodos, o algoritmo está pronto para ser executado pelo algoritmo evolutivo. Nas seções à frente, é apresentado um resumo de como estes métodos foram implementados para cada problema.

2.3. Schwefel Function

O indivíduo utilizado para este problema foi caracterizado como uma lista de 30 bits, onde cada grupo de 10 bits representam um número inteiro, variando seu valor de 0 à 1024. O algoritmo tem a função de aleatorizar e aproximar o valor destes três números binários para o valor inteiro mais próximo do resultado, o número 421. Como o algoritmo não trabalha com

números negativos, a função de *fitness* subtrai 500 do valor de cada inteiro presente no indivíduo para que ele represente a faixa de valores -500 à + 500. Abaixo está a representação de um indivíduo:

Este indivíduo possui 30 bits, separado em grupos de 10 bits. Neste exemplo, os três números inteiros são: 229, 220 e 691. Repassando a correção, os valores corretos são: -271, -280 e 191. O fitness deste indivíduo possui o seguinte valor: 650.5660173959305.

2.4. Caixeiro Viajante

O indivíduo utilizado para este problema foi caracterizado como uma lista de 24 bits, onde cada grupo de 3 bits representam um ID de uma cidade, variando entre 0 à 7. O algoritmo utiliza a distância entre dois pontos para calcular a distância total de um percurso.

Como as cidades são geradas aleatoriamente na inicialização do problema, a distância varia de acordo com as execuções. Para o fator de aceitação, é necessário que a distância do menor percurso seja repetida 10 vezes para cada *slave*.

Abaixo está a representação de um indivíduo:

Este indivíduo possui 24 bits, separado em grupos de 3 bits. Neste exemplo, os IDs das cidades são: 1, 2, 0, 3, 5, 6, 4, 7.

3. Resultados

3.1. Ambiente de Testes

Os testes foram executados no Laboratório de Informática I do Curso de Ciência da computação, utilizando 5 computadores sendo um *master* e quatro *slaves*. A configuração das máquinas é descrita abaixo:

Processador: Intel i5-240 3.5GHz 6MB Cache

Memória RAM: 4GB DDR3

Armazenamento: 500GB

Sistema Operacional: Ubuntu 14.04 (64 bits)

3.2. Schwefel Function

Quantidade de Modelos Probabilísticos Gerados			**População:** 900 indivíduos	
Execução	**25%**	**50%**	**75%**	**100%**
Nº 1	4	505	899	900
Nº 2	4	502	900	900
Nº 3	6	526	899	900
Nº 4	5	535	898	900
Nº 5	5	515	895	900
Nº 6	4	515	898	900

3.3. Caixeiro Viajante

Quantidade de Modelos Probabilísticos Gerados			**População:** 576 indivíduos	
Execução	**25%**	**50%**	**75%**	**100%**
Nº 1	22	389	574	576
Nº 2	12	367	576	576
Nº 3	12	392	574	576

Nº 4	13	385	573	576
Nº 5	18	381	576	576
Nº 6	23	379	575	576

REFERENCIAIS TEÓRICOS

SILVEIRA, J. F. Porto da. Problema do Caixeiro viajante. 2002. Disponível em: http://www.mat.ufrgs.br/~portosil/caixeiro.html. Acessado em 18 de nov 2015.