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

Relatório sobre o desenvolvimento:

Rastrigin Function e Radio Network Optimization

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

Profª. Drª. Rubens Barbosa Filho

Felipe Lima Morais
Robson Takashi Kawakita

Dourados - MS
Novembro de 2015

1. Introdução

1.1. Algoritmos Genéticos

Algoritmos Genéticos são inspirados no princípio Darwiniano da evolução das espécies e na genética . São algoritmos probabilísticos que fornecem um mecanismo de busca paralela e adaptativa baseado no princípio de sobrevivência dos mais aptos e na reprodução (GOLBERG, 2015).

1.2. Rastrigin Function

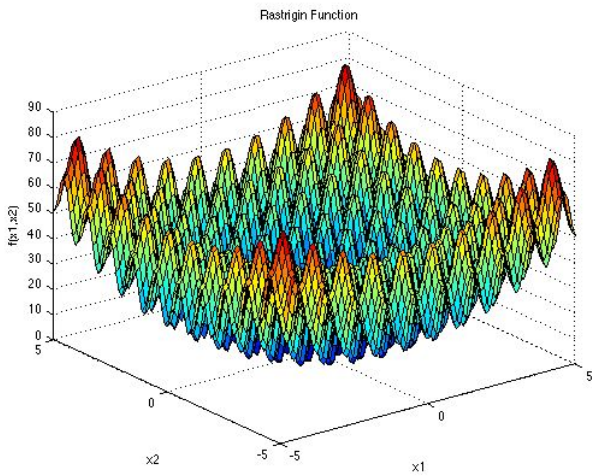
A função Rastrigin é um algoritmo utilizado para executar testes de otimização de algoritmos. A função Rastrigin tem vários mínimos locais, a localizações desses são distribuídos regularmente, possui a complexidade de O(n ln(n)), onde n é a dimensão do problema. A Figura 1 mostra a definição da função, e a Figura 2 mostra a representação gráfica na sua forma tridimensional.

Figura 1: Definição da função Rastrigin

$$f(\mathbf{x}) = 10d + \sum_{i=1}^{d} \left[x_i^2 - 10\cos(2\pi x_i) \right]$$

Fonte: http://www.sfu.ca/~ssurjano/rastr2.png

Figura 2: Representação gráfica da função Rastrigin

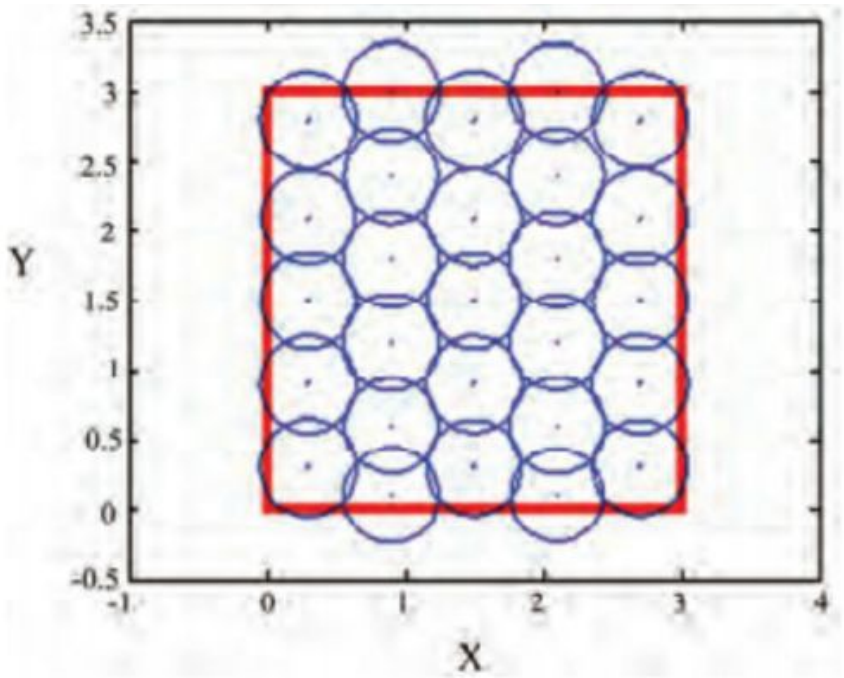


Fonte: http://www.sfu.ca/~ssurjano/rastr.png

1.3. Radio Network Optimization

O Radio Network Optimization é um problema de otimização de cobertura de sinal. ele se propõe a descobri: dada uma determinada área, qual o melhor posicionamento das antenas afim de cobrir a maior área possível. Existem várias casos em que este problema pode ser aplicado, não apenas a distribuição de sinal. São características consideradas são a área de cobertura de cada antena, os pontos onde elas estão instaladas, o desperdício de cobertura fora da área programada, a sobreposição de cobertura entre antenas e entre outros. A Figura 3 expressa graficamente em 2D um exemplo de distribuição de antenas. Os pontos são as antenas, os círculos são a área de cobertura das antenas e o quadrado é a área que deseja ser coberto pelo sinal das antenas.

Figura 3: Exemplo de distribuição de antenas.



Fonte: http://www.sfu.ca/~ssurjano/rastr.png

1.4. Objetivos

O objetivo deste trabalho é implementar os algoritmos Rastrigin Function e o Radio Network Optimization utilizando algoritmos evolutivos, relatar seu desenvolvimento e descrever os resultados.

2. Implementação

2.1. Materiais

A linguagem *Python*, na versão 2.7, foi utilizada para o desenvolvimento dos algoritmos de do *middleware*. O *Message Passing Interface* (MPI) foi utilizado para distribuir das tarefas de processamento da execução entre as máquinas, e a biblioteca *mpi4py* na comunicação dos algoritmos e o MPI.

2.2. Middleware

Os algoritmos foram modelados como classes, compostas por 6 métodos cada que são utilizados pelo *middleware*:

- **new_individual()** - Retorna uma lista de '0' ou '1' aleatório como forma do novo individuo em representação binária.
- **get_fitness(individual)** - Retorna um valor que representa quão próximo esta da resposta, quanto mais perto, retorna uma resposta próxima de **zero**.
- **is_finished(individual)** - Verifica se o indivíduo é uma resposta aceitável do problema do problema.
- **validate_individual(individual)** - Verifica se um indivíduo é válido para o domínio do problema.
- **num_bits()** - Retorna o numero de bits que representa um individuo.
- **show(individual)** - Imprime um individuo na tela.

2.3. Rastrigim Function

O método new_individual() da classe Rastrigin gera um indivíduo de acordo com variáveis globais para facilitar a utilização, no testes os indivíduos foram compostos por 30 *bits* que representaram 3 números inteiros de 10 bits variando de 0 à 1024. Este indivíduo representa uma coordenada [x1, y1, z1] do domínio da função. Para trabalhar com o domínio da função de uma maneira mais fácil, foi utilizados alguns artifícios. O domínio da função Rastrigin é de -5,12 a 5,12, logo para não utilizar decimais e números negativos. Multiplicamos por 100 e somamos 512, dessa maneira a nossa representação poderia ser feita por 10 bits por número e 30 bits a coordenada. O método validate_individual() resposavel por verifica se os números gerados estão no intervalo de 0 a 1024, que representam o domínio

da função -5,12 a 5,12. A função get_fitness() retorna a proximidade da resposta, conforme mostra a implementação na Figura 4.

Figura 4: Método get_fitness da classe Rastringin

```
def get_fitness(self, individual):
    v = []
    for i in xrange(AMOUNT_NUM):
        number = int(''.join(individual[i*NUM_BITS_IN_NUM:(i+1)*NUM_BITS_IN_NUM]), 2)
        v.append((number - 512)/100.0)

    alpha = 10
    fitness = 0
    for i in range(AMOUNT_NUM):
        fitness += v[i]**2 - alpha*math.cos(2*math.pi*v[i])
    return float(fitness) + alpha*AMOUNT_NUM
```

Fonte: Elaborada pelos autores

Na linha 5 o numero que estava representado em binário é subtraído 512 e dividido por 100 para voltar ao seu valor original.

1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0

Este indivíduo possui 30 bits, representando três números de 10 bits. Neste exemplo, os três números inteiros são: 514 , 512 e 514. Repassando a correção, os valores corretos são: 0.02 , 0.00 , 0.02. O fitness deste indivíduo possui o seguinte valor: 0.15850597371.

2.4. Radio Network Optimization

Para este problema definimos as constates antes de iniciarmos a resolução: Número de bits de x1 “(x1, y1)”; Porcentagem da área de deseja cobrir; Raio de alcance de uma antena. Pois, a partir dessas informações definimos as características para resolvermos o problema. Por exemplo.

- Número de bits de x1 “(x1, y1)”: 5
- Porcentagem da área de deseja cobrir: 0.8
- Raio de alcance de uma antena: 10

A partir dessas informações definimos a área total para ser coberto.

Área Limite: *2 ^ número de bits = 2^5 = 32*.

Área Total: *32 * 32 = 1024*

A área limite é similar ao problema anterior, consideramos uma eixo de 0 a 32, mas isso representa um eixo -16 a 16. Dessa forma não trabalhamos com valores negativos.

O número de antenas necessárias para cobrir a área desejada é obtida pela formula:

$$Teto(\frac{(Área\ Total \times Porcentagem\ de\ área\ para\ cobrir)}{Area\ coberta\ por\ uma\ antena})$$

Exemplo: $Teto(\frac{(1024 \times 0.8)}{(pi \times (10^2))}) = 3$

O indivíduo utilizado para este problema é de uma lista de **Número de Antenas * Número de bits de x1 * 2** bits, neste exemplo uma lista de 3*5*2 bits. Representa todas as coordenadas das antenas [x1, y1, x2, y2 ...], onde (x1, y1) é a coordenada de uma antena. Os indivíduos são gerados aleatoriamente pela função new_individual().

A função validate_individual(individual) valida os indivíduos para que estejam dentro da área válida. As antenas também são validadas para que toda a área de cobertura esteja dentro da área valida. No exemplo a nossa área válida está até 32.

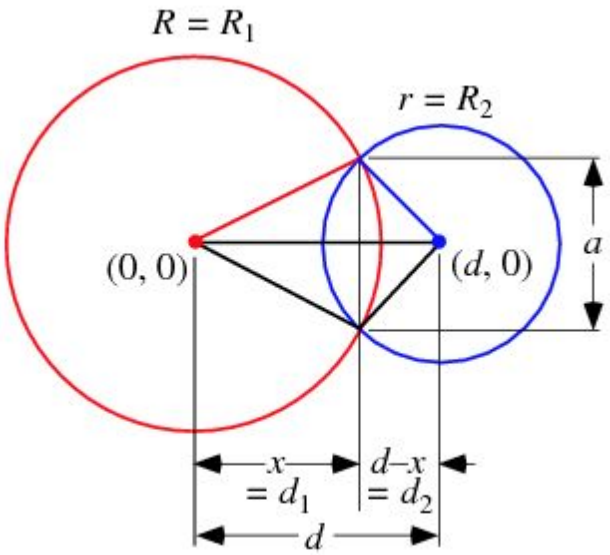
O get_fitness(individual) verifica o quanto da área foi coberta, levando em consideração a área de intersecção das antenas, subtraindo da área coberta. Quanto maior é a área coberta, quanto mais próximo estiver da área coberta desejada, mais proximo o fitness está do zero. Para o valores sejam representado corretamente o número que está em bits é subtraido pela *Área Limite*/2. Dessa forma, recuperamos a representação da coordenada, como o exemplo das coordenadas em bits, onde os números são (30,10), mas para o algoritmo significa (14,-6).

1	1	1	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	1	1

Este indivíduo acima possui 30 bits, representando três antenas. Neste exemplo, as antenas estão representados nas coordenadas (28,2) , (9, 0) , (14, 3). Repassando a correção, as coordenadas são: (12, -14) , (-7, -16) , (-2, -13).

Uma dificuldade está no calculo de sobreposição de círculos (Área de cobertura das antenas) no calculo do get_fitness(individual). Para resolver o problema, foi utilizado a formula apresentada na Figura 6, tornando possível o calculo das sobreposições, sendo possível analisar na Figura 5:

Figura 5: Exemplo de sobreposição



Fonte: http://mathworld.wolfram.com/Circle-CircleIntersection.html

Figura 6: Formula de sobreposição de círculos.

$$A = A(R, d_1) + A(r, d_2)$$
$$= r^2 \cos^{-1}\left(\frac{d^2 + r^2 - R^2}{2\,d\,r}\right) + R^2 \cos^{-1}\left(\frac{d^2 + R^2 - r^2}{2\,d\,R}\right) -$$
$$\frac{1}{2}\sqrt{(-d + r + R)(d + r - R)(d - r + R)(d + r + R)}\,.$$

Fonte: http://mathworld.wolfram.com/Circle-CircleIntersection.html

3. Resultados

3.1. Ambiente de Testes

O ambiente de testes foi Laboratório de Informática I, foram utilizados 5 computadores, suas especificações são 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. Rastrigim Function

Modelos Probabilísticos Gerados

Tabela 1: Execução do Rastrim Function com uma população de 900 indivíduos

Teste	25%	50%	75%	100%
1º	3	522	897	900
2º	5	519	900	900
3º	4	492	900	900
4º	4	496	900	900
5º	3	517	898	900
6º	6	510	900	900

Fonte: Elaborada pelos autores

O Rastrigin Function chegou ao resultado. De todos os testes realizados no Rastrigin algumas chegaram ao final, mas, apenas tenho o número de iterações de um teste. No teste 6, o resultado chegou para a Slave 1 com 4819 iterações e 6 modelos probabilísticos, para a Slave 2 com 7 iterações e 510 modelos, para Slave 3 com 2 iterações e 900 modelos e para Slave 4 com 1 iteração com 900 modelos. Todos as Slaves convergiram ao fitness = 0, que no nosso problema se dava na coordenada [0.0 , 0.0 , 0.0].

Nos outros testes apenas a Slave 1 demorava para convergir, enquanto as outras Slaves convergiam em menos de 10 iterações.

3.3. Radio Network Optimization

Infelizmente os dados gerados pelo Radio Network Optmization não foram suficientes para construção de resultados. O tempo para a convergência desse algoritmo não possibilitou a conclusão das execuções e analise dos dados em tempo hábil para a construção do relatório.

4. Referencias

GOLBERG, David E. Genetic algorithms in search, optimization, and machine learning. Addion wesley, 1989.

SURJANOVIC, Sonja; BINGHAM, Derek. Optimization Test Problems Rastingin Function. <http://www.sfu.ca/~ssurjano/rastr.html> Acessado em 18/11/2015.

Weisstein, Eric W. "Circle-Circle Intersection." From *MathWorld*--A Wolfram Web Resource. <http://mathworld.wolfram.com/Circle-CircleIntersection.html> Acessado em 18/11/2015