Melhor carga algoritmia genético aplicado a estocagem de produtos

Introdução

O problema da estocagem de produtos que considerando sua área e valor selecionando a melhor combinação de itens. O objetivo principal é preencher a uma área com o maior valor possível, não ultrapassando a área máximo permitida, ou seja, organizar os objetos dentro de uma área pré-definida considerando seu valor utilizando a otimização combinatória.

Problema – Trabalho Pratico!

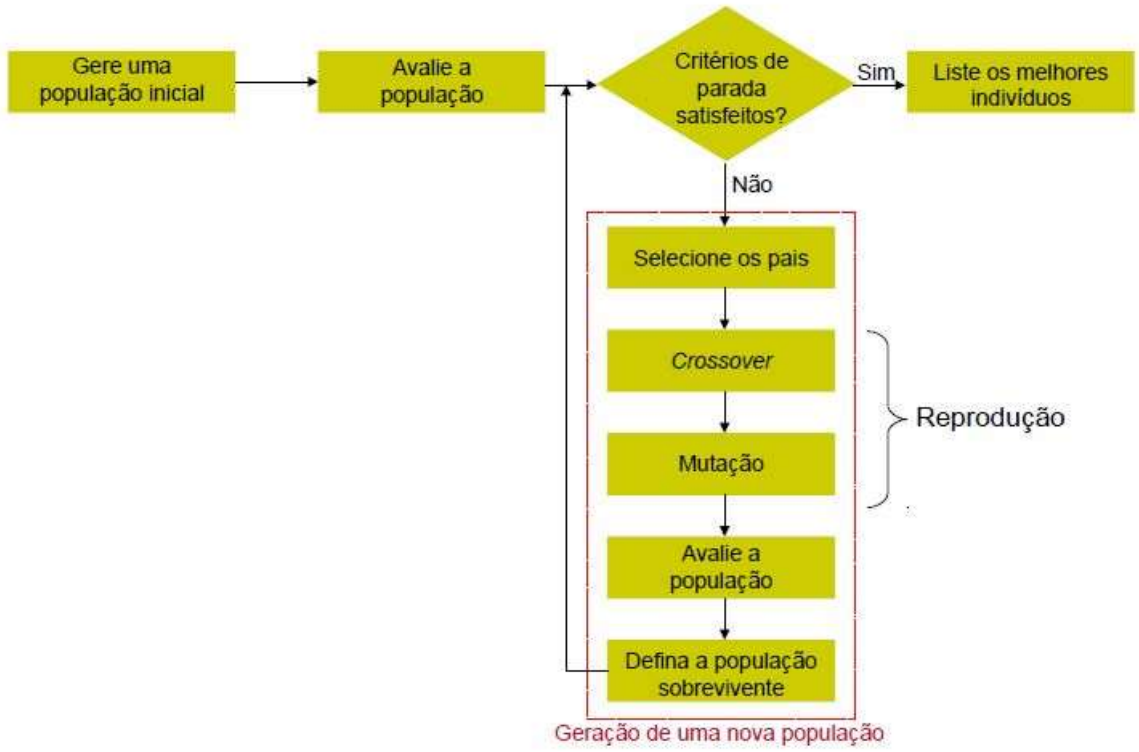
Desenvolver um programa que aplique Algoritmos Genéticos para resolver o problema de selecionar a melhor carga considerando um conjunto de itens, cada qual com um determinado área e valor, determinar quais itens devem ser inclusos no espaço em m² predefinido, de maneira que se obtenha o maior valor e que a soma da área desses itens não ultrapasse a capacidade máxima permitida.

Para a instância do problema que deverá ser resolvido neste trabalho, considere os seguintes dados:

Objeto	1	2	3	4	5	6	7	8
Valores	3,00	3,00	2,00	4,00	2,00	3,00	5,00	2,00
Área m²	5	4	7	8	4	4	6	8

Área máxima permitida: 25 m²

Estrutura do algoritmo genética:



Desenvolvimento.

Para representar um cromossomo (individuo) ou melhor um possível problema, vamos representar uma possível solução para o problema. Representado por uma cadeia de bits, nesse caso, sempre 8 (quantidade de objetos). Bit 1 representa que o objeto deve ser incluso, bit 0, não deve ser incluso. Exemplo

Cromossomo 1 = 11001110 → Deve-se incluir os objetos 1, 2, 5, 6 e 7. Efetuando o cálculo do área.

ArrayList e Vetores: Utilizaremos sempre ArrayList para representa a população, temos um Array de Vetor! que ficara salvo o cromossomo + sua aptidão ou qualquer outro dado, servirá como um índice para os cálculos.

```
ArrayList<String[]> idvs = new ArrayList<>();
idvs.add(new String[]{"11111001", 36});
idvs.add(new String[]{"11001110", 25});
idvs.add(new String[]{"11100001", 16});
```

Classe AplicacaoAG

1º Passo – Criado o projeto, vamos utilizar de uma .class (classe) chamada AplicacaoAG onde serão programados a maioria dos métodos necessários para a execução do algoritmo.

Criamos 3 variáveis globais, 2 vetores de inteiros que representarão o valor e o área dos objetos respectivamente, e um ArrayList de Strings por ser dinâmico representará a população de indivíduos, por fim um construtor que receberá os valores como parâmetro.

2º Passo – Criaremos 2 métodos um para exibir a população de indivíduos e o outro para fazer o calculo de aptidão.

A principio essas funções são bem simples, vejamos: No método exibePopulacao(ArrayList<String> populacao) recebemos como parâmetro um ArrayList e passamos como valor pra variável global _POPULAÇÂO.

```
    public void exibePopulacao(ArrayList<String> populacao) {
        this._POPULACAO = populacao;
        console.append("População >> " + populacao + "\n");
    }
```

No metodo calculaAptidao(String _DNA) recebemos uma String como parâmetro na qual calcularemos o valor de cada individuo levando em conta a posição de seu valor e área nos vetores. Dizendo que: Se na posição da String for “1” somaremos o valor de acordo com a posição do vetor, caso contrario “0” não será somado, no fim e retornado um inteiro do valor que aquele individuo tem ou seja sua Aptidão!

```
public int calculaAptidao(String _DNA) {

```

```
    /*
    utiliza o Vetor de valorObj para calcular a aptidão retorna o valor
    da aptidão
    ou retorna 0 caso o mesmo ultrapasse a área de 25 m², calculado
    pela função validaCromossomo
    */
            int aptdao = -1;

            if (validaCromossomo(_DNA)) {
                for (int i = 0; i < _DNA.length(); i++) {
                    String aux = String.valueOf(_DNA.charAt(i));
                    if (aux.equals("1")) {
                        aptdao += valorObj[i];
                    }
                }
            }

            return aptdao;
        }
```

validaCromossomo()

É um método que retorna verdadeiro caso o indivíduo tenha aptidão <= a 25m² que é o limite da mochila ou falso se ultrapassar!

```
        public boolean validaCromossomo(String _DNA) {

            /*
            Utilizando o vetor global de areaObj para calcula o área.
            retorna um valor Booleano no caso do cromossomo ter o
            área menor ou igual a 25.
             */
            int area = 0;

            for (int i = 0; i < _DNA.length(); i++) {
                String aux = String.valueOf(_DNA.charAt(i));
                if (aux.equals("1")) {
                    area += areaObj[i];
                }
            }
            return area <= 25;
        }
```

3 º Passo – Listar os melhores indivíduos, para escolher os melhores indivíduos utilizaremos a seleção por roleta onde na lista de indivíduos serão escolhidos os com probabilidade de escolha % maior, para isso a expressão matemática será, x = (y1 + y2 …) / x, será a probabilidade de cada um a soma de todos dividido por ele mesmo

Para essa etapa utilizaremos 2 métodos, um para selecionar apenas a Aptidões validas, ou seja, menor ou igual a 25, e o outro para fazer a seleção na roleta, ordenando do maior para o menor, ficando apenas com metade da lista para que seja feito o Crossover posteriormente.

selecaoPorAptdao()

```
    public ArrayList<String[]> selecaoPorAptdao() {

        //Descarta os individuos com aptidão -1 ou seja INVALIDO
        ArrayList<String[]> idvs = new ArrayList<>();
        int cont = 0;

        console.append("Calculo de Aptidões >> ");
        for (String idv : _POPULACAO) {
            int apt = calculaAptidao(idv); //Calcula a aptdao e
adiciona no array aptds
            if (apt > -1) {
                idvs.add(new String[]{idv, String.valueOf(apt)});
                console.append("[" + idv + ", " + apt + "] ");
            } else {
                cont++;
            }
        }
        console.append("\nDescartados >> " + cont + " Individuos
com o valor acima de 25m²\n");
        return idvs;
    }
```

selecaoPorRoleta()

```
public ArrayList<String> selecaoPorRoleta(ArrayList<String[]> idvs)
{

        float soma = 0;
        for (String[] apt : idvs) {
            soma += Float.parseFloat(apt[1]);
        }

        //Calcula o % de probabilidade de um idv ser escolhido e
imprimi no console
        console.append("Probabilidade de Escolha >>\n");
        for (String[] apt : idvs) {
            apt[1] = String.valueOf(soma /
Float.parseFloat(apt[1]));
            console.append("[" + apt[0] + ", " + apt[1] + "%]\n");
        }
        //Ordena o melhores IDVS de acordo com a probabilidade de
ser escolhido
       for (int i = 0; i < idvs.size(); i++) {
            String[] a = idvs.get(i);
            double prob1 = Double.parseDouble(a[1]);

            for (int j = i; j < idvs.size(); j++) {
                String[] b = idvs.get(j);
```

```
                double prob2 = Double.parseDouble(b[1]);

                if (prob2 > prob1) {
                    String[] c = idvs.get(j);
                    idvs.remove(j);
                    idvs.add(i, c);
                }
            }
        }
        ArrayList<String> melhores = new ArrayList();
        int cont = 0;
        for (String[] aux : idvs) {
            if (cont <= _POPULACAO.size() / 2) {
                melhores.add(aux[0]);
                cont++;
            }
        }
        console.append("\nIndividuos Selecionados >> " + melhores +
"\n");

        return melhores;
    }
```

4º Passo – Agora já com o indivíduos selecionados vamos gerar os "filhos" através do método de CrossOver, o método consiste em pegar a 1ª metade de um Indivíduo e junta com a 2ª metade de outro para formar um individuo e depois a outras metade formando outro individuo, formando assim 2 filhos veja:

IDV 1 = 1000 0001
IDV 2 = 1111 0000
RESULTADO 1 = 1000 0000
RESULTADO 2 = 1111 0001
Os métodos é bastante simples, recebe dois indivíduos e retorna um novo individuo.

Crossover()

```
    public String Crossover(String c1, String c2) {

        /*
        Função de crossover de cromossomos, pega a metade (1 parte)
do cromossomo c1
        mais a metade (2 parte) do cromossomo c2, depois o mesmo
com c2 (1 parte) + c1 (2 parte)
         */
        String filho = c1.substring(0, c1.length() / 2) +
c2.substring(c2.length() / 2);

        return filho;
    }
```

5º Passo, apos gerar 4 filhos (indivíduos) a partir de 4 pais (indivíduos) é preciso mais um passo antes de formar a nova população que é a Mutação, para manter assim sempre a diversidade dos indivíduos, para esse caso simples utilizei uma classe de números aleatórios para sortear posição que será alterada:

EX: IDV 1 = 10001111 —> Mutação (random em uma posicao) —> IDV 1 = 10101111

Mutacao()

```
public String Mutacao(String cromossomo) {

   /*  Sorteia uma posição da String e altera o valor e
retorna outra String */
        Random rd = new Random();
        int pos = rd.nextInt(cromossomo.length());
        char[] aux = cromossomo.toCharArray();

        for (int i = 0; i < aux.length; i++) {
            if (i == pos) {
                if (aux[pos] == '1') {
                    aux[pos] = '0';
                } else {
                    aux[pos] = '1';
                }
            }
        }
        return String.valueOf(aux);
    }
```

6º Passo, após passar por todas a etapas de preparação dos indivíduos, vamos gerar uma nova população afim de passar por todo o processo novamente; para isso foi elaborado uma função que retorna um ArrayList<String> aplicando os métodos aprendidos até aqui veja:

GeraNovaPopulacao()

```
public ArrayList<String> GeraNovaPopulacao() {
        ArrayList<String[]> selecao = selecaoPorAptdao();
        ArrayList<String> pais = selecaoPorRoleta(selecao);
        ArrayList filhos = aplicaCrossover(pais);
        ArrayList mutacao = aplicaMutacao(filhos);
        ArrayList nova_populacao = new ArrayList();
        //Cria a nova populacao de inviduos
        nova_populacao.addAll(pais);
        nova_populacao.addAll(mutacao);
        System.out.println("Nova Populacao >>
"+nova_populacao);
        return nova_populacao;
    }
```