UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

PEDRO GABRIEL DE SOUZA VEREZA MEDEIROS

# M-KNAPSACK E ALGORITMOS GENÉTICOS

Porto Alegre
2015

# 1 INTRODUÇÃO

Este relatório traz resultados e comparações entre o uso de método simplex aplicado através do software GLPK e do uso de algoritmos genéticos para solução do problema Multidimensional Knapsack.

A seção 2 apresenta uma breve descrição do problema bem como a modelagem utilizada na resolução pelo método simplex.

A seção 3 descreve com detalhes a implementação de algoritmos genéticos utilizada. São descritas as características dos principais operadores, bem como relatos de problemas encontrados e justificava para decisões tomadas durante a implementação.

Na seção 4 encontra-se  tabelas comparativas entre as duas abordagens mencionadas e discussão sobre os resultados obtidos.

## 2 MULTIDIMENSIONAL KNAPSACK

O problema conhecido como multidimensional knapsack é um problema da classe NP-Hard e consiste em, dado uma knapsack com d dimensões e n itens com um valor e d medidas cada, definir quais itens serão colocados na knapsack, de forma a aumentar o lucro (valor total dos itens na knapsack) respeitando o limite de cada dimensão da knapsack. O problema pode ser formulado matematicamente utilizando as seguintes variáves:

Cj : capacidade da knapsack na dimensão j
Vi : valor do item i
Wij : tamanho do item i na dimensão j
Xi : 1 se item está na knapsack, 0 caso contrário.

$$MAX \sum_{i=1}^{n} Xi * Vi$$

Sujeito a:

$$\sum_{i=1}^{n} (Xi * Wij \leqslant Cj) \forall j$$

$$Xi \in \{0,1\} \forall i$$

## 3 ALGORITMO GENÉTICO

A heurística escolhida para solucionar o problema da muldimensional knapsack foi Algoritmos Genéticos

Cada possível solução para o problema é representada como um cromossomo. O tamanho de cada cromossomo é igual à quantidade de itens na instância do problema, sendo que cada alelo pode assumir dois valores: 1 se o item está presente na knapsack e 0 caso contrário.

### 3.1 MUTAÇÃO

A operação de mutação é aplicada a todos os cromossomos da população, antes que o crossover seja executado. Cada alelo do cromossomo tem uma possibilidade de 60% de sofrer mutação. A alta chance de mutação se deve ao fato de que o cromossomo modificado é representado como um novo cromossomo, ou seja, a população possui o cromossomo antes e depois da mutação. Para evitar que muitos cromossomos iguais existam na população, o que dificultaria o avanço do algoritmo devido a baixa diversidade, o valor alto foi estipulado para aumentar as chances de que ao menos uma modificação seja feita no cromossomo.

Valores menores forem experimentados (5%, 10%, 25%, 40%), bem como valores maiores (80%, 90%) mas os melhores resultados foram encontrados com 60%. Valores muito baixos mostraram-se pouco eficientes pois em muitos casos não ocorria modificação no cromossomo, duplicando-o na população e dificultando avanço. Por outro lado, valores muito altos dificultaram a convergência do algoritmo, uma vez que a busca se tornava muito aleatória.

### 3.2 CROSSOVER

Operação de crossover também foi incluída no algoritmo. Inicialmente, utilizou-se crossover com partição fixa, ou seja, aplicando crossover sempre na mesma posição em todos os cromossomos. Essa alternativa mostrou-se problemática por mover blocos muito grande dos cromossomos pais, fazendo com que alelos próximos sempre fossem mantidas nas próximas gerações. Tentou-se introduzir aleatoriedade na escolha do tamanho de cada partição, mas o problema mencionado anteriormente continuo ocorrendo.

Por fim, utilizou a técnica de crossover uniforme, em que cada alelo é considerado indivudalmente. A técnica consiste em randomizar um valor entre 0 e 1, dividir este intervalo em duas partições iguais, e associar cada pai a uma dessas partições. Desta forma, cada filho recebe informação de um dos pais, dependendo do valor randomizado.

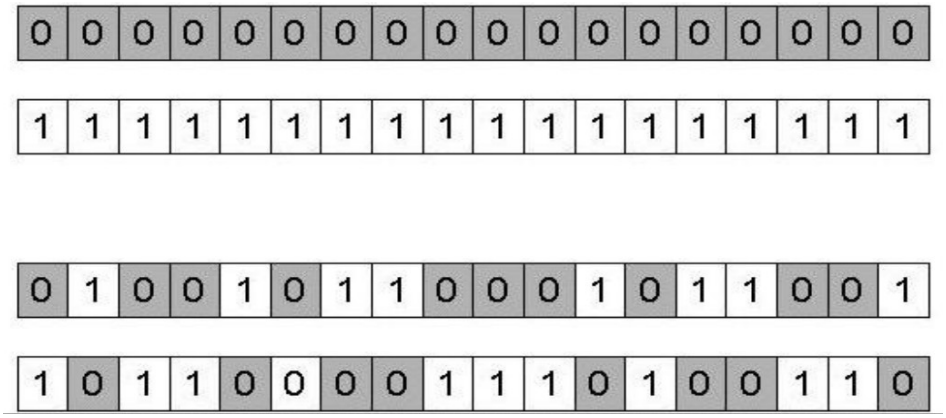


*Imagem 1: Exemplo de crossover uniforme. Os dois primeiros cromossomos representam os pais e os dois ultimos os filhos.*

### 3.3 FITNESS

Para avaliar cada solução dois valores foram considerados: valor dos itens na solução e uma penalidade para soluções que não respeitem as dimensões da knapsack. O cálculo da penalidade para cada knapsack é feito da seguinte forma:

$$penalidade = -(\sum_{j=1}^{d} excesso(\sum_{i=1}^{n} Wij, j))$$

A função excesso recebe o tamanho total da possível solução na dimensão j, e o valor máximo que pode ser utilizado na dimensão j na knapsack final e retorna zero caso o tamanho total seja menor ou igual a capacidade da dimensão j, ou a diferença entre tamanho total e capacidade caso contrário.

$$excesso(total, j) = 0 \; se \; total \leq Cj$$
$$Ou$$
$$total - Cj, caso \; contrário$$

O valor da penalidade é sempre um número negativo. Como a ordenação de cromossomos para seleção para próxima geração é decrescente, garante-se que soluções factíveis tenham prioridade na escolha.

Para soluções em que a penalidade é igual a zero, ou seja, os items escolhidos cabem dentro das restrições, o valor da função fitness é o valor total da knapsack (positivo ou zero).

## 3.4 POPULAÇÃO

O tamanho da população é configurável, mas possui valor padrão de 50. Este valor foi o que trouxe resultados mais satisfatórios e mais rápidos. Valores diversos foram testados (entre 20 e 180) mas não obtiveram resultados superiores a 50.

A cada geração, são selecionados os 10 melhores individuos e outros cinco escolhidos aleatoriamente para garantir que haja diversidade. Estes 15 individuos sofrem mutação, gerando outros 15. Além disso, estes cromossomos realizam crossover com outro cromossomo escolhido aleatóriamente.

Devido a essas operações, a população atinge valores maiores que o máximo estipulado. Para manter o valor escolhido, os individuos são ordenados por valor de fitness e os n primeiros são considerados, onde n é o tamanho máximo da população. O ordenando nesta etapa é importante para que não sejam descartados cromossomos com soluções melhores.

## 3.4 CRITÉRIO DE PARADA

Diferente do método simplex utilizado pelo GLPK para resolver o problema, não é possível se certificar de que a solução encontrada por um algoritmo genético é ótima durante a execução do algoritmo. Por este motivo, algum processamento além do necessário é feito para se certificar de que outra solução melhor não é encontrada.

O critério de parada testado inicialmente foi o número de gerações. Diversos casos de testes foram feitos com limite de gerações variando de 20 até 20.0000. Para instâncias menores do problema, a solução ótima era encontrada em menos de 50 gerações. Instâncias maiores, no entanto, levavam várias gerações e, em muitos casos, não atingiam o valor ótimo.

A estratégia de parada foi alterada para considerar o número de gerações que passaram sem que houvesse melhoria na solução. Inicialmente, estipulou-se que, se após 50 gerações não houvesse melhoria na solução, a execução era terminada. Este valor mostrou-se muito baixo devido ao grande espaço de busca nas gerações iniciais que contém casos não-factíveis ou com solução muito abaixo da solução ótima. Após diversos testes com outros valores, o limite utilizado foi de 5.000 gerações sem melhorias.

## 4 RESULTADOS

A seguir, são apresentadas tabelas dos resultados obtidos pela execução das duas abordagens. A Tabela 1 apresenta os resultados obtidos pelo software GLPK. Na Tabela 2 são mostrados os resultados obtidos com o algoritmo genético desenvolvido.

| Instância | Tempo (em segundos) | Melhor valor obtido | Solução ótima |
|---|---|---|---|
| Pb7 | 0.1 | 1035 | Sim |
| Sento1 | < 0.0 | 7772 | Sim |
| Hp1 | < 0.0 | 3418 | Sim |
| Weish30 | < 0.0 | 11191 | Sim |
| OR5x100-0.25_1 | 6.7 | 24381 | Sim |
| OR5x100-0.75_1 | 0.4 | 59822 | Sim |
| OR30x500-0.25_1 | 7200 | 115865 | Não |
| OR30x500-0.75_1 | 7200 | 301643 | Não |

*Tabela 1: Resultados obtidos através do software GLPK*

| Instância | Tempo (em segundos) | Melhor valor obtido | Solução ótima | Desvio percentual |
|---|---|---|---|---|
| Pb7 | 5.5 | 1024 | Não | 1.06 |
| Sento1 | 22.8 | 7772 | Sim | 0 |
| Hp1 | 1 | 3418 | Sim | 0 |
| Weish30 | 7.1 | 11137 | Não | 0.48 |
| OR5x100-0.25_1 | 14.23 | 23685 | Não | 2.85 |
| OR5x100-0.75_1 | 0.4 | 58788 | Não | 1.72 |
| OR30x500-0.25_1 | 2470.04 | 97710 | Não | 15.66 |
| OR30x500-0.75_1 | 23188.17 | 290769 | Não | 3.6 |

*Tabela 2: Resultados obtidos através da execução do algoritmo genético apresentado*

Comparando os resultados obtidos, é notável o melhor desempenho do GLPK em todas as instâncias do problema. Principalmente em problemas com instâncias pequenas, no qual o método simplex é muitas vezes mais performático que metaheurísticas.

Ainda assim, o algoritmo genético mostrou-se mais lento em instâncias maiores. Notou-se que o desempenho médio do algoritmo era, em alguns casos, superior ao GLPK, mas ao se aproximar muito da solução ótima, o desempenho caía abruptamente. A dificuldade de avançar nos útlimos passos da busca fizeram com que muito tempo fosse perdido e que a performance média fosse menor.

A alta taxa de mutação necessária para as gerações iniciais pode ser a causa da dificuldade de convergência quando o algoritmo está próximo da solução ótima. Uma alternativa que pode ser testada neste contexto é a aplicação de conceitos de *simulated annealing,* diminuindo a taxa de mutação durante a execução do algoritmo, fazendo com que gerações inicias tenham um taxa maior de mutação e que gerações mais próximas do valor ótimo tenham taxas menores. No entanto, essa abordagem pode dificultar a saída de máximos locais que não sejam máximos globais. Ao diminuir a taxa de mutação, a tendência é que o algoritmo se prende cada vez mais a máximos, sendo globais ou não.