INSTITUTO FEDERAL
Fluminense
Campus Campos Centro

BUSCA LOCAL ITERADA NA SOLUÇÃO DO PROBLEMA DA MOCHILA BINÁRIA

Valmir Monteiro[1]

[1]Bacharelado em Engenharia de Computação

Campos dos Goytacazes
26 de Julho de 2021

1 INSTRUÇÕES

Para compilar o algoritmo em linux, abra uma janela do terminal no diretório onde estão os arquivos .c e execute a seguinte linha de comando:

```
gcc ILS_Main.c ILS_Base.c ILS_Simples.c Quicksort.c -lm -o ILS_Main
```

Antes de executar o programa, certifique-se que o arquivo contendo os itens da mochila estão no mesmo diretório e obedecem a formatação definida pelo trabalho. Estando tudo de acordo, execute a linha de comando correspondente a versão que deseja executar.

• Para executar a versão com solução inicial gulosa:

```
./ILS_Main -g NomeDoArquivo
```

• Para executar a versão com solução inicial aleatória:

```
./ILS_Main -a NomeDoArquivo
```

O nome do arquivo inserido deve obrigatoriamente estar igual ao nome do arquivo original, obedecendo a formatação de letras maiúsculas, minúsculas e também deve incluir a extensão do arquivo. Por exemplo, se o nome do arquivo for **Mochila.txt** e a versão a ser executada será com **solução inicial gulosa**, o comando executado deve ser:

```
./ILS_Main -g Mochila.txt
```

1.1 Ajustes

No arquivo ILS_Header.h há algumas constantes que podem ser alteradas para ajustar o funcionamento do algoritmo.

- **LIMITE_ITERACOES_SEM_MELHORA:** quantidade de vezes que a busca local continuará rodando quando não encontrar uma solução global melhor. Quanto mais alto, mais tempo o algoritmo levará para concluir, porém maior será a chance de encontrar uma solução melhor.
- **GRAU_PERTURBACAO:** porcentagem da solução base que será perturbada. Quanto maior esse valor, menor será a parcela da solução original na solução após a perturbação. Intervalo [0,1]
- **RESET_PERTURBACAO:** por padrão, a perturbação é aplicada apenas na melhor solução local. Para evitar ficar presa em ótimos locais, essa constante controla a frequência com que a perturbação escolherá uma solução aleatória na memória. Quanto menor, mais frequente a perturbação utilizará esse recurso. Intervalo [0,1]
- **MODIFICADOR_RAIO_VIZINHANCA**: multiplicador do raio que a vizinhança abrange. Se for muito pequeno, as soluções da vizinhança não vão ter itens suficientes para formar uma solução boa. Se for muito grande, perderemos diversidade pois muitas soluções da vizinhança ficarão iguais. Deve ser ajustado levando em consideração a densidade dos pontos.

2 ALGORITMO

2.1 Introdução

Para o desenvolvimento desse algoritmo escolhemos utilizar a proporção entre o valor e o peso de cada candidato (valor/peso) como principal critério de escolha. Primeiro todos os itens são dispostos em um plano cartesiano, sendo (valor, peso) a coordenada de cada item.

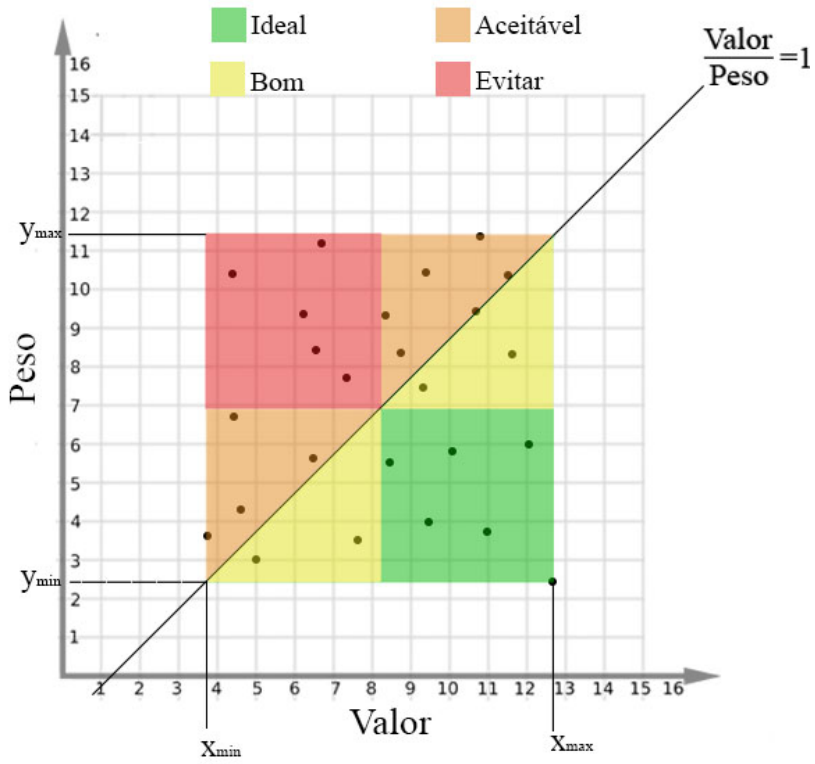

Figura 1: Plano cartesiano formado pelos candidatos.

Analisando o gráfico podemos ver que os itens no canto inferior direito apresentam as maiores proporções de valor/peso, e portanto são os itens de maior interesse. Não é desejável operar apenas na zona verde, pois pode levar a uma falta de diversidade nas escolhas, mas queremos evitar ao máximo entrar na zona vermelha, pois é onde estão os itens com menor ganho e maior peso.

2.2 Vizinhança

Para a criação da vizinhança foi utilizada como base a distância entre cada item no plano. Delimita-se um raio e, para cada item, é feita uma busca comparando a distância com todos os outros itens. Os candidatos que estão dentro do raio são incluídos na vizinhança do referido

item. Para armazenar as vizinhanças foi utilizada uma matriz de adjacência, porém algumas otimizações foram necessárias.[1]

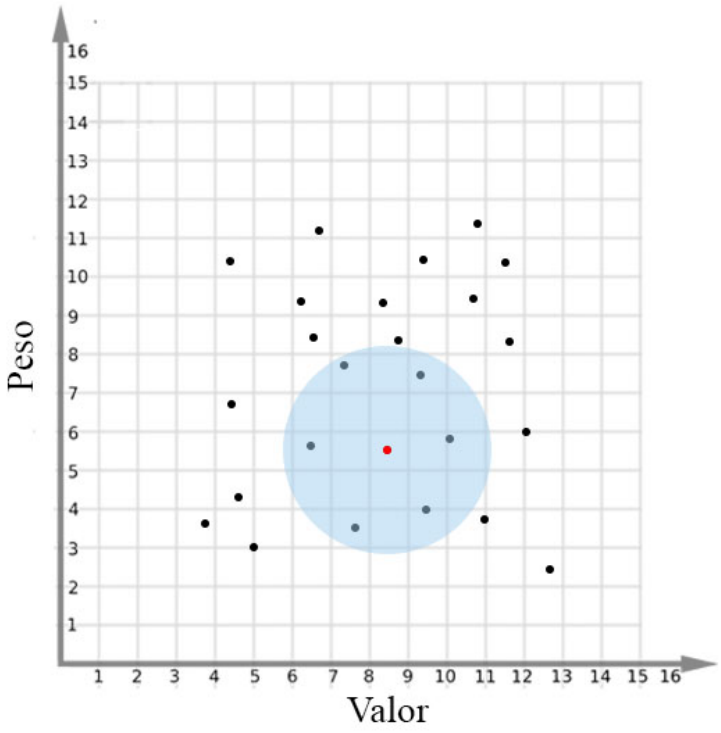

Figura 2: Para cada ponto é feita uma busca para determinar quais itens fazem parte da vizinhança.

Como a disposição dos pontos é diferente para cada conjunto de candidatos, buscamos uma forma de generalizar a definição do raio da vizinhança e chegamos à seguinte relação:

$$Raio = \frac{Capacidade}{Média\ de\ peso} * Média\ da\ distância * Modificador$$

onde:

- Capacidade: capacidade total da mochila
- Média de peso: média de peso dos candidatos
- Média da distância: magnitude do vetor distância média V_m

$V_m = \langle (maiorValor - menorValor)/quantidade, (maiorPeso - menorPeso)/quantidade \rangle$

- Modificador: índice ajustado no arquivo ILS_Header.h de acordo com a densidade dos pontos (MODIFICADOR_RAIO_VIZINHANCA)

[1] Um dos requisitos do trabalho é que o algoritmo possa funcionar com até 10.000 itens e, caso fosse utilizado uma matriz quadrada de inteiros, o programa poderia consumir mais de 380MB de memória (10.000^2 x 4 bytes) apenas com a alocação dessa matriz. Visando reduzir o impacto dessa operação, escolhemos utilizar uma matriz triangular de caracteres, sendo a escrita e leitura feitas utilizando operações de bit shifting. Dessa forma é necessário apenas um bit para guardar cada aresta do grafo, reduzindo a demanda de memória em aproximadamente 98% ((10.000^2 / 2 – 10.000) x 1 bit).

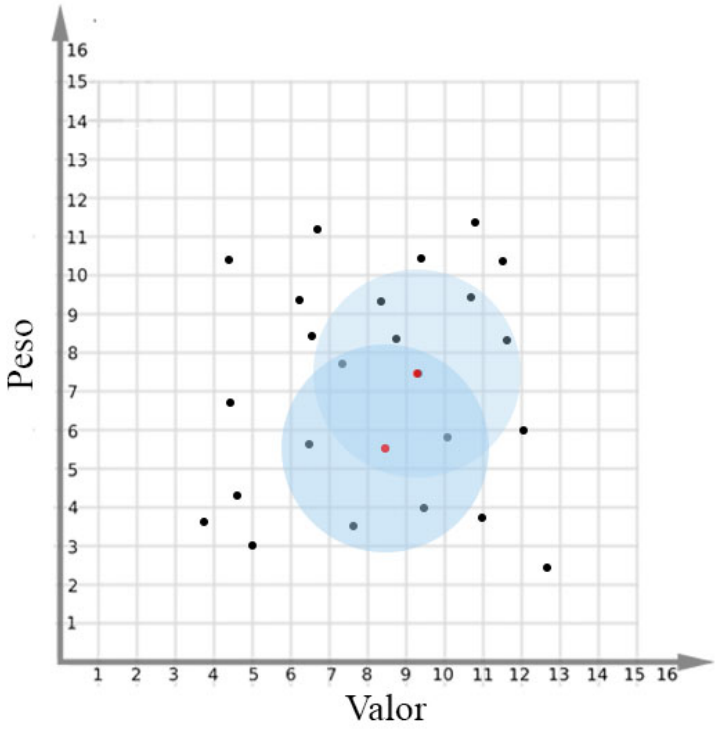

Figura 3: As soluções associadas a itens vizinhos tendem a possuir elementos em comum

2.3 Solução Inicial

A solução inicial pode ser gerada de forma aleatória ou utilizando um algoritmo guloso, que busca inserir os itens de maior proporção valor/peso na solução. Em todos os testes, os resultados utilizando a solução inicial gulosa foram superiores aos que utilizaram a solução aleatória.

2.4 Busca Local

Antes de realizarmos a busca local, primeiro associamos a cada item uma solução. Essas soluções são criadas com base na vizinhança de cada candidato, e utiliza também uma estratégia gulosa para ser gerada, visando maximizar o valor que cada uma delas possui. Todas as soluções são armazenadas em um vetor memória, e a posição de uma solução no vetor corresponde ao índice do item ao qual a solução está associada. Dessa forma não há a necessidade de calcular essas soluções toda vez que forem solicitadas pela busca local, basta apenas acessar o vetor na posição desejada.

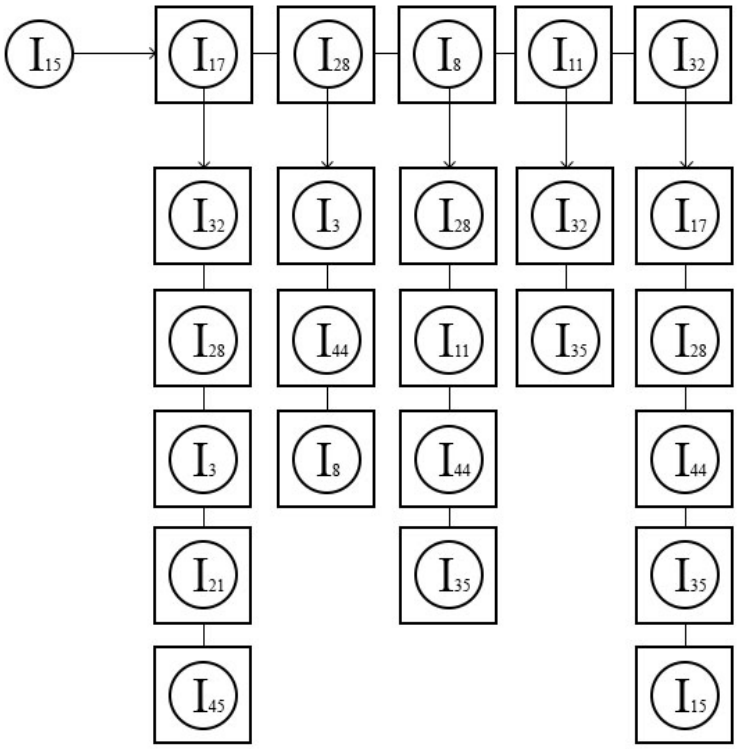

Figura 4: O item 15 possui uma solução associada a ele. Cada item dessa solução, por sua vez, também possui uma solução associada.

Ao entrar na busca local temos uma solução base que será utilizada, na primeira iteração essa solução é a solução inicial. Agora iremos comparar essa solução base com as soluções que estão armazenadas na memória nas posições dos respectivos índices dos itens que compõem essa solução base. Depois dessa comparação teremos a melhor solução local. Essa melhor solução local será perturbada e o resultado dessa perturbação será utilizado na próxima iteração da busca local.

Como condição de saída, definimos um número máximo de iterações sem que haja melhora na melhor solução global. Quando esse limite é atingido, retornamos a melhor solução encontrada até o momento.

2.5 Perturbação

Na perturbação são escolhidos, de forma aleatória, alguns itens para serem removidos da solução base. Após a remoção é feito uma busca, também de forma aleatória, por itens viáveis nas soluções associadas aos itens originais que compõem a solução base. Cada item é testado para que não seja excedida a capacidade máxima da mochila e, se o item for aceito, ele será adicionado na solução.

2.6 Testes

O algoritmo foi testado utilizando entradas que variam entre 3 e 10.000 itens. Para cada entrada foram feitas cinco tentativas com cada versão do algoritmo, e o melhor resultado de cada versão foi armazenado em uma tabela. A seguir estão as configurações no arquivo ILS_Header.h e os resultados obtidos.

LIMITE_ITERACOES_SEM_MELHORA = 100
GRAU_PERTURBACAO = 0.3
RESET_PERTURBACAO = 0.25
MODIFICADOR_RAIO_VIZINHANCA = 16

Número de Itens	Solução Inicial	Valor Total	Tempo de Execução (seg)
3	Gulosa	160	0,00500
	Aleatória	**220**	0,00600
10	Gulosa	**309**	0,00600
	Aleatória	266	0,00600
50	Gulosa	**544**	0,00700
	Aleatória	388	0,00800
100	Gulosa	**1548**	0,00800
	Aleatória	1443	0,00800
500	Gulosa	**3719**	0,01600
	Aleatória	2968	0,01400
1000	Gulosa	**6612**	0,03400
	Aleatória	4864	0,03600
5000	Gulosa	**13802**	0,49300
	Aleatória	10111	0,41700
10000	Gulosa	**22493**	1,63900
	Aleatória	13661	1,79200

Tabela 1: Resultados obtidos no teste do algoritmo.

Analisando a tabela nota-se que em quase todos os casos a solução inicial gulosa o consegue chegar a uma solução global melhor do que a encontrada utilizando uma solução inicial aleatória.

REFERÊNCIAS

OLIVEIRA, Eduardo; AMARAL, André. **Desenvolvimento de um algoritmo de busca local iterada para o problema dial-a-ride.** XLVII Simpósio Brasileiro de Pesquisa Operacional, 2015, Porto de Galinhas, Pernambuco.

GENDREAU, Michel; POTVIN, Jean-Yves. **Handbook of Metaheuristics.** 2ª edição, Nova York: Springer Publishing Company, 2010.

VIEIRA, Carlos Eduardo. **Heurísticas para o Problema das p-Medianas Conectadas.** 2006. 191 f. Tese (Doutorado em Informática) – Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2006.