Problema da Dieta - Iterated Local Search (ILS)

Mariane Regina Afonso Vieira[1], **Nathany Aparecida Salles**[1]

[1]Instituto de Ciências Exatas e Aplicadas (ICEA)
Universidade Federal de Ouro Preto (UFOP)
Caixa Postal 24 - CEP 35.931-008, João Monlevade - MG - Brasil

mariane.vieira@live.com, nathany.salles@yahoo.com.br

Resumo. *O objetivo do problema da dieta é selecionar um conjunto de alimentos possíveis que satisfazem uma quantidade mínima de nutrientes diários a um custo mínimo. Este documento visa apresentar a implementação deste problema usando o algoritmo de Busca Local Iterada na linguagem Java.*

1. Introdução

Este trabalho aborda o problema da dieta como desafio de otimização utilizando o método da Busca Local Iterada (ILS), escolhido por ser um método simples e produz ótimos resultados. O projeto foi desenvolvido na linguagem Java por ser de conhecimento das integrantes do grupo.

O documento apresenta uma breve revisão teórica na seção 2. Na seção 3 é abordado o desenvolvimento do algoritmo e os resultados alcançados, por fim, uma rápida conclusão demonstrando as impressões sobre o algoritmo estudado.

2. Revisão Bibliográfica

2.1. O Problema da Dieta

O problema da dieta [1] foi um dos primeiros problemas de otimização estudado entre 1930 e 1940. George Joseph Stigler sugeriu o problema motivado pela preocupação do exército americano em alimentar suas tropas com o menor custo possível, ao menos tempo, atendendo os valores nutricionais necessários.

Em 1945, Stigler apresentou o problema: para um homem pesando em médio 70kg, qual a quantidade de alimentos, dentre os 77 possíveis, deve ser ingerida para atender as quantidades mínimas de nutrientes sejam iguais as estabelecidas pelo Conselho Nacional de Pesquisa Norte - Americano e além disso, gere o menor custo. Usando uma heurística, Stigler reduziu o problema a 9 inequações e 77 variáveis positivas, chegando ao custo de 39.93 dólares por ano. Em 1947, Jack Laderman utilizou o método Simplex para resolver o problema. 9 balconistas utilizando calculadoras manuais, 120 dias-homem e chegaram no custo ideal de 39.69 dólares

2.2. Iterated Local Search (ILS)

É um método que gera uma sequência de soluções baseadas por uma heurística de busca local. Esse algoritmo ”guia”a execução para as melhores soluções ao invés de usar testes aleatórios repetidos, esse fato o deixa mais eficiente.[2]

A ideia principal do algoritmo é fazer uma busca no espaço S* dos mínimos locais do problema. A partir de um mínimo local e a cada iteração, é gerado uma pertubação

Algoritmo 1: ITERATED LOCAL SEARCH

```
Entrada: s0                                              /* Solução inicial */
Saída: s                                          /* Melhor solução encontrada */
1  início
2      BuscaLocal(s0)
3      s ← s0
4      custo ← funcaoCusto(s)
       /* Mantém loop se o número máximo de iterações sem melhora foi
       ultrapassado                                                        */
5      while numIter < numMaxIter do
6          s* ← s
7          perturbacao(s*)
8          BuscaLocal(s*)
9          custo* ← funcaoCusto(s*)
10         numIter ← numIter + 1
11         if custo* <= custo then
12             s ← s*
13             numIter ← 0
14         end
15     end
16     return s
17 fim
```

Figura 1. Pseudocódigo ILS

que não deve ser alta para não fugir das boas soluções e não deve ser fraca para não cair em ótimos locais e limitar a diversificação, em seguida é aplicada uma busca local[3]. A heurística incorporada usada neste trabalho foi a Busca Tabu.

2.2.1. Busca Tabu

É um procedimento adaptativo auxiliar para a busca local dentro de um espaço de busca. A partir de uma solução inicial, a busca se move para a melhor solução na vizinhança, não realizando movimentos que já foram realizados por estarem armazenados na lista tabu. A lista permanece na memória durante um determinado tempo ou até um número de iterações sem melhora.

3. Desenvolvimento

O algoritmo foi desenvolvido na linguagem Java, e foram criadas 4 classes:

- **Refeicao.java:** objeto que representa as refeições. Possui como atributos:
  - tipo: pode ser café da manhã, almoço ou lanche;
  - quantidade de alimentos a ser comprada;
  - quantidade mínima de nutrientes a ser ingerida naquela refeição;
  - quantidade de nutrientes que cada alimento possui;
  - preço de cada alimento e custo da compra.

  Além disso possui duas funções:
  - *minNutrientes()*:gera a quantidade aleatória de cada alimento respeitando a quantidade mínima de nutrientes exigida pela refeição;
  - *minCusto()*: calcula o custo da refeição considerando o preço e a quantidade de cada alimento.
- **BuscaTabu.java:** possui duas funções:

  - *buscaTabu(refeicao, tamTabu):* responsável por fazer a busca loca, recebe uma refeição como solução inicial e o tamanho do tabu e retorna um *array* com a quantidade de alimentos;
  - *sortearAlimento(refeicao):* sorteia um alimento e um nutriente e gera o sorteio enquanto a quantidade mínima de nutrientes não for atingida para o alimento sorteado. Essa função é usada também como pertubação para o algoritmo ILS;
- **ILS.java:** é a classe responsável pelo algoritmo ILS;
- **Main.java:** Classe principal, responsável por inicializar variáveis e chamar a função ILS.

3.1. Implementação

```
public void ils(Refeicao refeicao, int numIter){

        Refeicao s0 = (Refeicao) refeicao.clone1(); // solução
           inicial

        BuscaTabu bt = new BuscaTabu();
        ArrayList<Double> buscaTabu = new ArrayList<>();

        buscaTabu = bt.buscaTabu(s0, 10); // busca local

        Refeicao s1 = s0;
        s1.setQtdAlimento(buscaTabu);
        s1.minCusto();

        int i = 0;

        while(i < numIter) { //enquanto não satisfeito o crité
           rio de parada

            //pertubação
            ArrayList<Double> perturbacao = bt.sortearAlimento(
               s1);

            Refeicao s2 = (Refeicao) refeicao.clone1();

            s2.setQtdAlimento(perturbacao);

            //busca local
            s2.setQtdAlimento(bt.buscaTabu(s2, 10));

            s2.minCusto();

            //criteirio de aceitação
            if(s2.custo < s1.custo) {
                s1 = (Refeicao) s2.clone1();
                s1.minCusto();
            }
```

```
35
36              i++;
37          }
38
39          System.out.println("********** SOLUÇÃO ILS **********");
40          System.out.println("Quantidade de cada alimento a ser
               comprado: " + s1.getQtdAlimento());
41          System.out.println("Custo da compra: " + s1.getCusto());
42          System.out.println("
               ****************************************\n");
43      }
```

O critério de parada foi definido pelo número de iterações sem melhora. A pertubação foi gerada pelo método *sortearAlimento(refeicao)* e o critério de aceitação foi o menor custo entre as soluções geradas.

3.2. Resultados

Abaixo encontra-se o registro de uma das execuções geradas:

```
[Calcio, VitaminaA, Proteína] -> [8.0, 10.0, 50.0]
Nutrientes da refeição:  8.659347059074038 26.017178739042237 52.41866214091595
Cardápio: Pão Integral, Suco Natural, Biscoito Integral, Geleia de Frutas
Preço de cada alimento: [5.0, 3.0, 5.5, 15.2]

********** SOLUÇÃO INICIAL **********
Quantidade de cada alimento a ser comprado:[0.5505808910737807, 0.4267418566968618, 0.9935282302808869, 0.26229550905425314]
Custo da compra: 13.484427029629014
****************************************

********** SOLUÇÃO ILS **********
Quantidade de cada alimento a ser comprado: [0.2837749591435781, 0.18975964091112552, 0.6712715174391984, 0.10150650878377288]
Custo da compra: 5.874969438971887
****************************************
```

Figura 2. Café da manhã

```
[Calcio, VitaminaA, Proteína] -> [11.0, 70.0, 250.0]
Nutrientes da refeição:  32.84038266671139 126.86756166751361 253.3270852344447
Cardápio: Brócolis, Cenoura, Peixe, Carne
Preço de cada alimento: [4.0, 2.0, 25.5, 19.75]

********** SOLUÇÃO INICIAL **********
Quantidade de cada alimento a ser comprado:[0.4210611878505738, 0.004221090951342044, 0.5570325878290756, 0.07263899594321699]
Custo da compra: 17.331638092824942
****************************************

********** SOLUÇÃO ILS **********
Quantidade de cada alimento a ser comprado: [0.5390984878973248, 0.7010556020018753, 0.3624065501857051, 0.13777116937168776]
Custo da compra: 9.606941638269197
****************************************
```

Figura 3. Almoço

```
[Calcio, VitaminaA, Proteína] -> [20.0, 30.0, 25.0]
Nutrientes da refeição:  28.712457176003884 62.014866753002565 94.09601764398063
Cardápio: Patê de Atum, Biscoito, Chá, Fruta
Preço de cada alimento: [6.0, 5.0, 10.0, 1.75]

********** SOLUÇÃO INICIAL **********
Quantidade de cada alimento a ser comprado:[0.6921983148723154, 0.6915284280784088, 0.5956109159272965, 0.7573092936949445]
Custo da compra: 14.892232452865056
****************************************

********** SOLUÇÃO ILS **********
Quantidade de cada alimento a ser comprado: [0.7797278137646922, 0.2067261234658858, 0.4510882105618892, 0.4022253806628838]
Custo da compra: 5.091964256103519
****************************************
```

Figura 4. Lanche

4. Conclusão

A implementação do ILS é bastante simples e os resultados obtidos foram satisfatórios visto que a busca local utilizada reduziu bastante o custo das refeições. Em seguida, o algoritmo ILS aplicou uma otimização ainda maior, melhorando consideravelmente o custo. O custo poderia ser ainda mais minimizado se a solução inicial fosse gerada por um algoritmo guloso, por exemplo e não de forma aleatória como foi implementada.

Referências

[1] Programação linear - o problema da dieta. http://www.mat.uc.pt/~mat0829/PL-AndreLiliana2012.pdf. Acesso em: 25 de junho de 2018.

[2] Desenvolvimento de um algoritmo de busca local iterada para o problema dial-a-ride. http://cdsid.org.br/sbpo2015/wp-content/uploads/2015/08/142989.pdf. Acesso em: 25 de junho de 2018.

[3] Uma introdução à busca tabu. https://www.ime.usp.br/~gold/cursos/2009/mac5758/AndreBuscaTabu.pdf. Acesso em: 25 de junho de 2018.