Mini-projeto de Otimização

Mestrado em Engenharia Informática

Universidade de Aveiro

DETI - Aveiro, Portugal

UC - Simulação e Otimização
2022/2023

Ricardo Rodriguez 98388

Mini-projeto de Otimização

June 3, 2023

Contents

1 Introdução **3**

2 *GRASP (Greedy Randomized Adaptive Search Procedure)* **4**
- 2.1 Introdução 4
- 2.2 Implementação 4
- 2.3 Resultados 6

3 Programação Linear Inteira **7**
- 3.1 Introdução 7
- 3.2 Implementação 7
- 3.3 Resultados 9

4 Análise Comparativa **10**

5 Conclusão **11**

1 Introdução

Este relatório visa apresentar a metodologia usada para resolver o problema de otimização propostos para o segundo mini-projeto da unidade curricular de *Simulação e Otimização*, na vertente de otimização, bem como a apresentação de observações retiradas sobre o tópico em questão.

De acordo com o enunciado proposto, este mini-projeto tem como objetivo a implementação e a comparação entre dois métodos alternativos para resolver um problema de otimização: um método metaheurístico e um método exato baseado em programação linear. O objetivo do problema é selecionar o melhor conjunto de *n* switches numa Software Defined Network, ou SDN, para conectar a controladores SDN, minimizando o comprimento médio do caminho mais curto de cada switch até ao controlador mais próximo do mesmo.

Para representar as entidades do problema, são fornecidos três ficheiros que guardam informação sobre os nós/switches, os links entre os switches e o valor do custo entre estes. Além disso, o problema de otimização requer a determinação da seleção ótima de switches que garanta que o comprimento do caminho mais curto entre qualquer par de controladores SDN não ultrapassa um determinado limite (Cmax).

Para resolver este problema de otimização, foram implementados dois métodos: o método metaheurístico através do GRASP (Greedy Randomized Adaptive Search Procedure) e o método exato baseado em técnicas de programação linear inteira aprendidos nas aulas práticas da unidade curricular.

Posteriormente, será realizada uma análise comparativa para avaliar as soluções e os tempos de execução obtidos. No final, serão apresentadas as conclusões relativamente ao projeto.

2 *GRASP (Greedy Randomized Adaptive Search Procedure)*

2.1 Introdução

O GRASP, ou Greedy Randomized Adaptive Search Procedure, é um método metaheurístico que combina dois submétodos: o Greedy Randomized e o Adaptive Search, que serão explicados no próximo capítulo de implementação.

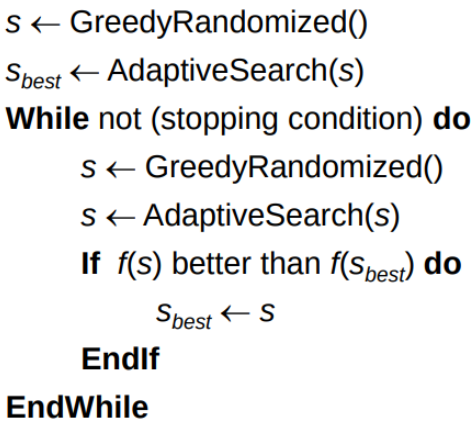

Figure 1: Método GRASP

2.2 Implementação

Para solucionar o problema de otimização com o método GRASP, foi implementado um programa em MATLAB capaz de selecionar uma solução ótima.

Inicialmente, são carregados os três ficheiros com dados relacionados aos nós, *links* e valores de custo de toda a rede SDN denotada, sendo criado um grafo que será usado ao longo do programa.

De seguida, são inicializadas algumas constantes que fazem parte da configuração do problema, nomeadamente o número de controladores de SDN que queremos selecionar, a distância limite entre dois controladores SDN, o número de iterações para o GRASP, o tempo limite para cada iteração (em segundos), a variável r, usada pelo GRASP e, por último, é inicializado uma lista $objectiveValues$ que vai guardar os valores para as funções de otimização para cada iteração/execução do programa.

```
% Initialize graph and its elements
Nodes = load('Nodes2.txt');
Links = load('Links2.txt');
L = load('L2.txt');
nNodes = size(Nodes,1);
nLinks = size(Links,1);
G = graph(L);

% Problem configuration
n = 10;
Cmax = 1000;
nRuns = 10;
runtimeLimit = 30;
r = 3;

% Stores the objective values of each run [1:n]
objectiveValues = zeros(1, nRuns);

% GRASP algorithm
for run = 1:nRuns

    % To keep track of the run elapsed time
    tic

    % Greedy randomized method
    E = 1:nNodes;
    solution = [];
```

Figure 2: Inicialização e configuração do programa

Após a inicialização de todas as variáveis necessárias, o método vai correr n vezes, sendo que em cada um destes usaremos o método Greedy Randomized e o Adaptive Search.

O método Greedy Randomized começa por criar uma lista vazia, $solution$, que irá guardar o conjunto atual de *n* controladores SDN selecionados. Para preencher este conjunto, existe um ciclo que vai ser executado *n* vezes, sendo que cada iteração vai guardar um dos controladores SDN com valores mais promissores.

Para isto, este itera sobre todos os nós/switches da SDN e executa a função AverageSP (calcula o valor médio do caminho mais curto de cada nó para o controlador mais próximo e a distância máxima observada entre dois controladores), sendo que a solução atual conta com todos os controladores previamente selecionados e o nó que está a ser atualmente processado. Existe uma matriz, cujas linhas possuem o identificador do nó e o *average path* calculado, e apenas são considerados os nós cuja solução não possua uma distância maior do que Cmax entre dois controladores SDN, tal como pretendido. Quando a iteração acaba, a matriz dos candidatos é ordenada pelo valor da função objetivo para realizar-se, depois, a seleção de um dos *r* melhores nós candidatos, de forma aleatória. Assim, no final do Greedy Randomized temos um conjunto de nós iniciais com bons valores da função de objetivo.

```
% Greedy randomized method
E = 1:nNodes;
solution = [];

for iter = 1:n
    R = [];
    for j = E
        [out1,out2] = AverageSP_v2(G, [solution j]);
        if (out2 <= Cmax)
            R = [R; j out1];
        end
    end
    R = sortrows(R, 2);
    e = R(randi(r), 1);
    solution(iter) = e;
    E = setdiff(E, e);
end
```

Figure 3: Método Greedy Randomized

O segundo passo do GRASP consiste no Adaptive Search que, neste caso, é o Steepest Ascent Hill Climbing. Este é um método baseado no Hill Climbing que, para cada uma das *n* posições da solução calculada anteriormente, realiza um ciclo. Este ciclo percorre todos os nós vizinhos à solução atual, *i.e.* todos os nós que não fazem parte do conjunto de solução, e troca um elemento do conjunto de solução, numa determinada posição atual, pelo nó vizinho. De seguida, é calculada a função de objetivo e, no caso do *average shortest path* ser melhor do que a solução atual e a distância máxima entre dois controladores ser inferior a Cmax, a melhor solução encontrada até agora passa a ser a solução candidata. Caso isto aconteça, o Steepest Ascent Hill Climbing regista que houve uma melhoria em desde a iteração passada, e continua a correr o método até não haver melhor solução (estagnamento).

```
improved = true;

while improved
    currentSolution = solution;
    currentObjective = AverageSP_v2(G, currentSolution);
    improved = false;

    for i = 1:n
        for j = setdiff(1:nNodes, currentSolution)
            candidateSolution = currentSolution;
            candidateSolution(i) = j;
            [candidateObjective, maxLenServers] = AverageSP_v2(G, candidateSolution);

            if candidateObjective < currentObjective && maxLenServers <= Cmax
                currentSolution = candidateSolution;
                currentObjective = candidateObjective;
                improved = true;
            end
        end
    end

    solution = currentSolution;

end
```

Figure 4: Método Steepest Ascent Hill Climbing

No final, o valor objetivo da solução calculada para a iteração do programa é calculada e registada na lista $objectiveResults$, que é usada posteriormente para calcular, e apresentar, os valores mínimos, médios e máximos das n iterações do GRASP, para o problema pretendido.

2.3 Resultados

Tal como referido anteriormente, o programa regista o valor mínimo, médio e máximo das n iterações do GRASP.

Para $n = 10$, $Cmax = 1000$, $runtimeLimit = 30$ e $nRuns = 10$, os valores obtidos foram os seguintes:

Mínimo	Média	Máximo
155.37	159.22	164.22

3 Programação Linear Inteira

3.1 Introdução

A Programação Linear Inteira (ILP) é uma técnica de otimização que visa maximizar ou minimizar uma função linear sujeita a um conjunto de restrições, sendo que todas as variáveis de decisão têm de ser valores inteiros. É muito usada, por exemplo, para problemas em que se tenham de usar valores inteiros em vez de valores fracionários como, por exemplo, na escolha das quantidades de certos itens para maximizar o lucro de uma viagem.

Para este trabalho, usamos a Programação Linear Inteira para modelar o problema apresentado numa estrutura matemática, definindo um cojunto de variáveis, restrições e objetivos que representem as características e as limitações do problema de otimização, no qual se pretende minimizar o valor da função objetivo.

3.2 Implementação

Como referido no enunciado, é necessário usar o LPSolve para delinear todo o conjunto de variáveis, restrições e a função objetivo do problema apresentado. Para descrever toda esta estrutura matemática, recorreu-se à criação de um programa, escrito em MATLAB, com o intuito de criar o ficheiro no formato LPSolve, que será corrido no próprio IDE para solucionar o problema de otimização.

Inicialmente, tal como no método GRASP, são inicializadas as variáveis necessárias para o problema apresentado, bem como as constantes da configuração do mesmo. A matriz $linksCosts$ percorre toda a estrutura do ficheiro $Links2.txt$ e, a partir dos pares de nós recebidos, recolhe o valor de custo entre estes e guarda-os na matriz de correlação. Esta matriz é usada para criar o grafo que será usado durante todo o programa.

De seguida, é calculada a distância entre todos os nós, também armazenados na matriz $nodesDistance$.

```
% Initialize graph and its elements
Nodes = load('Nodes2.txt');
Links = load('Links2.txt');
L = load('L2.txt');
nNodes = size(Nodes,1);
nLinks = size(Links,1);

% Problem configuration
n = 10;
Cmax = 1000;

linksCosts = zeros(nNodes,nNodes);
for i = 1:nLinks
    n1 = Links(i, 1);
    n2 = Links(i, 2);
    cost = L(n1, n2);
    linksCosts(n1,n2) = cost;
    linksCosts(n2,n1) = cost;
end

% Graph with costs
G = graph(linksCosts);

nodesDistance = zeros(nNodes, nNodes);
for i = 1:nNodes
    for j = 1:nNodes
        nodesDistance(i,j)= distances(G,i,j);
    end
end

% Output filename
filename = string("optimization_problemn" + n + ".lpt");
file = fopen(filename, 'w');
```

Figure 5: Inicialização e configuração de variáveis para o problema

Após a fase de inicialização do programa, inicia-se o processo de escrita no ficheiro de saída, com o formato adequado para ser executado no LPSolve.

Numa primeira fase, é necessário definir a função objetivo para o problema apresentado que, neste caso, será minimizar o valor médio do caminho mais curto entre um switch e o controlador SDN mais próximo.

Para isso, usamos a palavra-chave min, do LPSolve, para especificar que o objetivo é minimizar o valor da função matemática das variáveis que, tendo em conta o problema apresentado, é constituído por todos os *links* entre os nós da rede SDN, bem como o seu coeficiente que denota a distância entre os nós conetados.

```
% Objective function (minimize links)
fprintf(file, 'min\n');
for i = 1:nNodes
    for j = 1:nNodes
        fprintf(file, '+ %d l%d%d ', nodesDistance(i,j) ,i, j);
    end
end
```

Figure 6: Denotação da função objetivo do LPSolve

A estrutura da função objetivo terá a seguinte estrutura:

```
min
+ C1_1 l1_1 + C2 l1_2 + (...) + C200_200 l200_200
```

Contudo, para resolver o problema apresentado, é preciso denotar todas as restrições necessárias.

Como consta no problema, o objetivo é selecionar apenas n nós como controladores SDN entre todos os 200 nós da rede. Para delimitar esta situação, foi criada uma restrição que limita o número de nós cuja variável binária tenha valor positivo para n, estruturado da seguinte maneira:

```
subject to
n1 + n2 + n3 + (...) + n198 + n199 + n200 = n
```

De seguida, é preciso restringir o número de controladores que se conetam a um nó/switch para 1, uma vez que cada switch só se coneta a um controlador.

```
(para cada nó na rede SDN, ex: nó 1)
+ l1_1 + l2_1 + l3_1 + (...) + l199_1 + l200_1 = 1
```

Da mesma forma, o nó que se coneta a um determinado switch da rede deve ser um controlador SDN, por isso, delimita-se a seguinte restrição para cada par de nós da rede:

```
(para cada par de nós da rede, sendo l1_152 a conexão entre os nós 1 e 152, e
n1 a variável binária que define se o nó 1 é, ou não, um controlador)
+ l1_152 - n1 <= 0
```

Em último lugar, é também necessário especificar quais são os pares de nós cuja distância máxima entre dois controladores é superior a Cmax. Para isso, cria-se uma restrição que exclui a possibilidade de ambos os nós estarem presentes enquanto controladores SDN da solução, uma vez que a distância entre estes é superior a Cmax:

```
(neste exemplo, os nós 196 e 45 não podem coexistir enquanto controladores SDN,
uma vez que a distância entre os mesmos é superior a Cmax)
+ n196 + n45 <= 1
```

```
% Constraints
fprintf(file, '\nsubject to\n');

% n servers must be selected
for i = 1:nNodes
    fprintf(file, '+ n%d ', i);
end
fprintf(file, '= %d\n', n);

% One server must be assigned to each node
for i = 1:nNodes
    for j = 1:nNodes
        fprintf(file, '+ l%d%d ', j, i);
    end
    fprintf(file, " = 1\n");
end

% The server assigned to each node must be a server node
for i = 1:nNodes
    for j = 1:nNodes
        fprintf(file, '+ l%d%d - n%d <= 0\n', i, j, i);
    end
end

% Cmax constraint
for i = 1:nNodes
    for j = 1:nNodes
        if (nodesDistance(i,j) > Cmax)
            fprintf(file, '+ n%d + n%d <= 1\n', i, j);
        end
    end
end
```

Figure 7: Denotação das restrições do LPSolve

Em último lugar, é necessário especificar as variáveis que serão usadas pelo LPSolve para resolver o problema de otimização proposto. Deste modo, são precisos dois tipos de variáveis diferentes: as variáveis com prefixo n, variáveis binárias que explicam se o nó em questão é um controlador SDN ou um switch normal, e as variáveis com o prefixo l, igualmente variáveis binárias que denotam se ambos os nós estão conetados, ou não.

```
n1 n2 n3 (...) n199 n200 l1_1 l1_2 (...) l200_199 l200_200
```

```
% Variables
fprintf(file, '\nbinary\n');
for i = 1:nNodes
    fprintf(file, 'n%d ', i);
end
for i = 1:nNodes
    for j = 1:nNodes
        fprintf(file, 'l%d_%d ', i, j);
    end
end

fprintf(file, '\nend');
fclose(file);
```

Figure 8: Denotação das variáveis do LPSolve

3.3 Resultados

O programa linear de inteiros, executado no LPSolve com um limite de 15 minutos, teve os seguintes resultados:

Variables	MILP Fe...	MILP ...	MILP Be...	MILP ...	MILP ...	MILP ...	MILP ...	MILP ...	MILP ...	MILP ...	MILP ...	MILP ...	MILP ...	MILP ...	MILP ...	MILP ...	MILP ...	MILP B...	MILP ...	MILP ...	MILP ...	MILP ...	result
	33497,9...	33023	32990,9...	32984	32828	32781	32769	32672	3262...	32613	32592	32551	32539	32528	32516	32395	32383	32372	32359,...	32341	32286	32263	32249	32249

Figure 9: Resultados do programa ILP

Deste modo, dividindo o valor obtido da função objetivo pelo número de nós na rede SDN (200 nós), obtiveram-se os respetivos valores mínimos, médios e máximos:

Mínimo	Média	Máximo
161.24	163.81	167.48

4 Análise Comparativa

Analisando os resultados obtidos pelos diferentes métodos implementados, o GRASP e o programa ILP, podemos verificar que o GRASP tem melhores resultados do que o método ILP, tal como se pode verificar na tabela representada abaixo:

Método	Mínimo	Média	Máximo
GRASP	155.37	159.22	164.22
ILP	161.24	163.81	167.48

Desta forma, ainda que o ILP seja um bom método para obter uma solução ótima, este ainda está longe de alcançar os resultados de um método metaheurístico, como é o caso do GRASP que combina o Greedy Randomized com o Adaptive Search para obter uma solução ainda melhor.

5 Conclusão

Neste trabalho, foram desenvolvidos e comparados dois métodos para resolver um problema de otimização numa rede SDN. O objetivo, tal como referido, era selecionar um determinado número de switches para conectar a controladores SDN, minimizando o comprimento médio do caminho mais curto de cada switch até ao controlador mais próximo.

Através da implementação de um método meta-heurístico (GRASP) e um método exato baseado em programação linear inteira, foram obtidas soluções para o problema apresentado, verificando-se um desempenho superior por parte do GRASP em relação ao ILP, embora ambos os métodos tenham obtido bons valores de função objetivo.

Este trabalho proporcionou uma compreensão aprofundada das características e desempenho dos métodos meta-heurísticos e exatos, estes últimos mais raros, para problemas de otimização. Além disso, percebeu-se a importância de testar o GRASP com configurações diferentes, uma vez que se obtem diferentes resultados consoante os valores definidos (Cmax, r, etc.). Com base na análise comparativa deste trabalho, é recomendado o uso de métodos meta-heurísticos, como o GRASP, para problemas semelhantes de otimização, levando em consideração a relação entre a qualidade da solução e o tempo de execução.