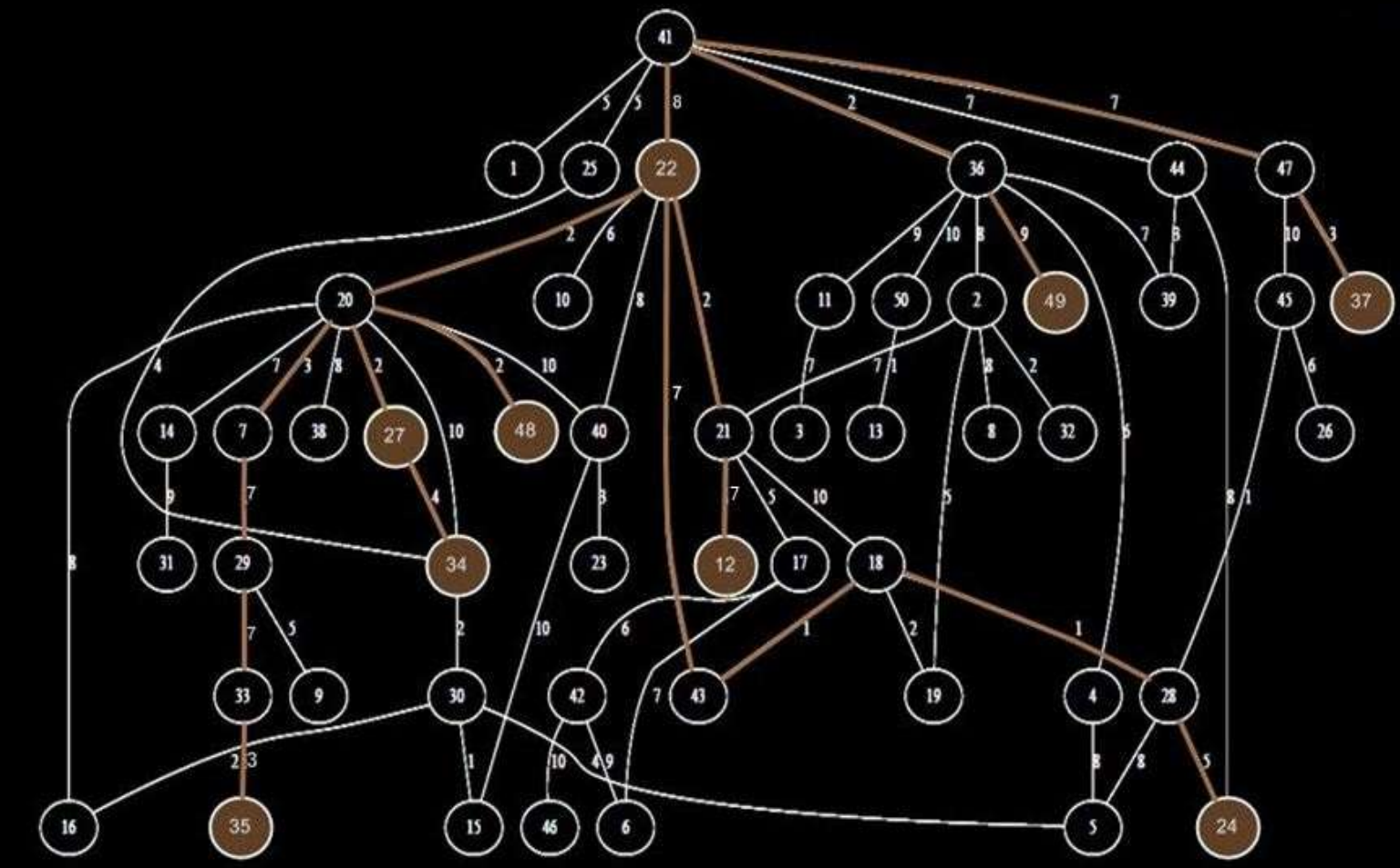


Elisabete Fontes Correia Ribeiro

Árvore de Steiner Bicritério em Redes de Comunicação

Dissertação de Mestrado de Engenharia Eletrotécnica e de Computadores na área de Telecomunicação, sob orientação da Doutora Lúcia Maria dos Reis Albuquerque Martins e Doutor Rui Pedro Charters Lopes Rijo.

Fevereiro/2014

· U C ·

Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Departamento de Engenharia Eletrotécnica e Computadores

Mestrado Integrado em Engenharia Eletrotécnica e Computadores

Árvores de Steiner Bicritério em Redes de Comunicação

Elisabete Fontes Correia Ribeiro

Júri :

José Manuel Fernandes Craveirinha (Presidente)

Lúcia Maria dos Reis Albuquerque Martins (Coorientadora)

Rui Pedro Charters Lopes Rijo(Coorientador)

Luís Alberto da Silva Cruz (Vogal)

Dedico esta dissertação aos meus pais, aos meus irmãos,
ao meu namorado Filomeno Vieira e a minha amiga Dulce Lopes.

Agradecimento

Esta dissertação simboliza a vitória e a prova que consegui ultrapassar todos os obstáculos que a vida me reservou até o momento. Se hoje estou aqui a concluir mais esta etapa da minha vida é porque houve pessoas especiais, que tiveram ao meu lado sempre disponíveis a apoiar-me incondicionalmente. A essas pessoas quero deixar aqui o meu profundo agradecimento.

Em primeiro lugar agradeço aos meus pais pelo apoio, carinho e o amor incondicional que souberam demostrar até agora. Aos meus irmãos e a minha tia Hena pelo carinho e pela confiança de que posso contar com eles para qualquer circunstância da vida.

Ao meu namorado, Filomeno Vieira pelo apoio, dedicação, força e amor demostrado ao longo desta etapa.

A minha amiga Dulce Lopes, nunca esquecerei a dedicação e apoio. Só quero dizer-te um muito obrigado.

A minha coorientadora Professora Lúcia Maria dos Reis Albuquerque Martins e o meu coorientador Rui Pedro Charters Lopes Rijo pela orientação, pelos conhecimentos transmitidos, pela disponibilidade e pelo apoio.

E por fim agradeço a Deus por tudo até agora.

Resumo

O objetivo desta dissertação de mestrado foi o estudo de uma meta-heurística conhecida para resolução do problema de Steiner em grafos e posteriormente fazer a sua extensão para a resolução de um problema de Steiner bicritério. Por já existir uma implementação desta meta-heurística (monocritério), embora com resultados que se desviavam dos apresentados pelos autores em [5], foi necessário começar por corrigir alguns dos seus algoritmos constituintes e, em alguns casos, desenvolver novos algoritmos por forma a corrigir o seu desempenho. Finalmente foi feita uma primeira adaptação desta meta-heurística para a resolução de um problema de Steiner bicritério.

Palavras-chave: optimização, árvore de Steiner, meta-heurística

Abstract

The aim of this master thesis was the study of a known meta-heuristic for solving the Steiner problem in graphs and subsequently extend this approach to solve a bicriteria Steiner problem. As already exist an implementation of this (single criteria) meta-heuristic, but with results that deviated from those presented by the authors in [5], it was necessary to begin by correcting some of the meta-heuristic's algorithms and, in some cases, develop new algorithms to improve its performance. Finally a first adaptation of this meta-heuristic for solving a bicriteria Steiner problem is herein proposed.

Keywords: optimization, Steiner tree, metaheuristic.

Conteúdo

Lista de Figuras

Lista de Tabelas

Listas de Símbolos e abreviaturas

STP	*Steiner Tree Problem*
MPH	*Minimum Path Heuristic*
BCSTP	*BiCriteria Steiner Tree Problem*

Capítulo 1

Introdução

O problema da árvore de Steiner em grafos é um problema de otimização onde se pretende determinar a árvore, ou seja, a ligação sem ciclos, de custo mínimo que interliga alguns nós específicos no grafo, podendo ser utilizados outros nós designados por nós de Steiner.

Este problema é considerado um problema muito complexo, não se conhecendo nenhum algoritmo de complexidade polinomial que o resolva com exatidão. Assim, na resolução do problema são utilizados heurísticas e meta-heurísticas que produzem soluções boas, embora não garantam que encontram a solução ótima.

O problema da árvore de Steiner tem diversas aplicações práticas, nomeadamente em redes de telecomunicações, onde a determinação dos recursos necessários para interligar um subconjunto de nós na rede pode ser considerado um problema de Steiner em grafos. Em alguns casos, para se interligar esse subconjunto de nós pode pretender-se não só a ligação de custo mínimo mas também com o menor número possível de *links*. Este caso pode ser considerado como um problema de Steiner bicritério. Em problemas multicritério geralmente não existe uma única solução que otimize simultaneamente as funções objetivo envolvidas na formulação do problema, mas sim um conjunto de soluções designadas por soluções não dominadas ou soluções ótimas de Pareto. As soluções ótimas de Pareto são todas as soluções para as quais não é possível encontrar outra solução que melhore o valor de uma das funções objetivo sem piorar pelo menos o valor de uma das outras funções objetivo.

O objetivo desta dissertação foi o estudo e implementação de uma meta-heurística (monocritério), apresentada em [5], para a resolução do problema de Steiner em grafos e adaptar esta mesma meta-heurística para uma formulação bicritério desse problema. Nesta formulação bicritério pretende-se minimizar, por um lado, o custo da árvore e, por outro lado, o número de *links* constituintes desta mesma árvore.

Esta dissertação de mestrado vem na sequência de um trabalho anterior [20], em que a meta-heurística monocritério já tinha sido implementada e, por conseguinte, o objetivo inicial era a continuação desse trabalho. No entanto, por se terem detetado algumas discrepâncias de resultados face aos apresentados pelos autores da meta-heurística em [5], foi necessário inicialmente rever todos os algoritmos desenvolvidos antes de se continuar com a extensão da meta-heurística para a formulação bicritério do problema. Este trabalho revelou-se mais complicado do que se esperava acabando por faltar tempo para uma adaptação mais cuidada da meta-heurística para a formulação bicritério idealizada inicialmente.

Apresenta-se de seguida a organização desta dissertação.

No capítulo 2 descreve-se a meta-heurística referida anteriormente e são introduzidos alguns conceitos básicos sobre grafos importantes para formalização do problema. O capítulo termina com a apresentação dos novos resultados experimentais obtidos.

No capítulo 3 são apresentados alguns conceitos básicos relacionados com a otimização multicritério e é apresentada uma primeira adaptação da meta-heurística estudada para o problema bicritério. Finalmente são apresentados alguns resultados experimentais por forma a validar o estudo realizado.

O capítulo 4 apresenta-se as principais conclusões do trabalho desenvolvido.

Capítulo 2

Problema de Steiner em grafos

2.1 Introdução

O problema da obtenção de uma árvore de Steiner (*Steiner Tree Problem* – STP) tem aplicação em várias áreas, nomeadamente em redes de telecomunicações onde pode ser usado para determinar os recursos necessários para interligar um subconjunto de nós na rede.

O problema da árvore de Steiner é muito complexo (NP-Completo), não se conhecendo algoritmos capazes de o resolver de uma forma exata. É possível encontrar na literatura algumas heurísticas e meta-heurísticas capazes de resolver este problema, embora não garantindo a obtenção de soluções exatas mas sim soluções muito próximas. É pois neste contexto que se insere este capítulo, onde é estudado o problema da árvore de Steiner em grafos usando a meta-heurística apresentada em [5]. Esta meta-heurística apresenta resultados muitos bons comparativamente com outras meta-heurísticas conhecidas [11, 12]. Ainda neste capítulo são introduzidos alguns conceitos básicos que ajudam na formalização do problema da árvore de Steiner em grafos.

2.2 Conceitos básicos

Uma rede pode ser representada por um grafo G=(V,E), onde $V = \{1, 2, 3, ..., |V|\}$ e $E = \{(i, j) : i, j \in V\}$ são os conjuntos finitos de nós (ou vértices) e arcos (ou arestas/links), respetivamente. Cada arco (i, j) está associado um custo (aditivo) $C_{ij} > 0$, em que i e j representam os vértices que são os extremos do arco. Por exemplo, na Figura 2.1, o grafo G é definido pelos conjuntos $V = \{1, 2, 3, 4, 5, 6\}$ e $E = \{$(1,2),(1,3),(1,4),(2,4),(2,5),(2,6),(3,4),(3,5),(4,5),(5,6)$\}$.

De acordo com o tipo de comunicação entre dois nós da rede, i e j, o grafo correspondente pode ser classificado da seguinte forma:

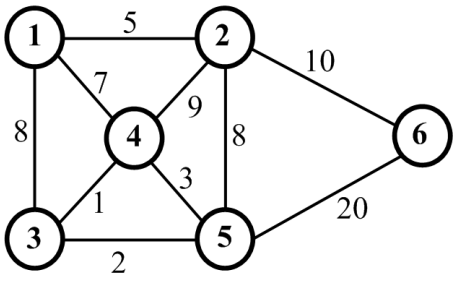


Figura 2.1: Exemplo de um grafo G não dirigido.

- Dirigido → se a comunicação entre os nós for apenas num sentido, por exemplo de i para j (e não de j para i);

- Não dirigido → se a comunicação entre os nós for em ambos os sentidos, i.e. de i para j e de j para i. Os grafos considerados ao longo do trabalho são sempre não dirigidos.

Num grafo não dirigido, dois nós i e j são adjacentes se no grafo existir um arco (i, j) no grafo. O arco (i, j) é dita incidente a ambos os nós i, j. O grau ou ordem de um nó corresponde ao número de arcos incidentes a este mesmo nó.

Considerando dois nós m e n num grafo G, um caminho de m para n é constituído por uma sequência de nós e arcos, $< (m, j), (j, k), ..., (l, n) >$. Uma vez que neste trabalho são considerados apenas custos aditivos, então o custo do caminho é a soma dos custos dos arcos que o constituem. Geralmente entre dois nós m e n existem vários caminhos, aquele com menor custo é considerado o *caminho mais curto* de m para n e pode ser obtida de forma exata através do algoritmo Dijsktra (ver anexo B). Qualquer caminho de um nó para si mesmo define-se por ciclo.

Um grafo não dirigido diz-se conexo se houver um caminho entre todos os pares de nó i e j. Uma folha é um nó de grau um. Uma árvore é um grafo conexo e sem ciclos, ou seja um grafo com um e um só caminho entre quaisquer dois nós. Uma árvore é abrangente num grafo G se contiver todos os nós de G. A *árvore abrangente mínima* é aquela que tem o custo mínimo comparado com o de todas as árvores abrangentes possíveis no grafo e pode ser obtida de forma exata através do algoritmo do Prim (ver anexo C). Um subgrafo de um grafo G=(V,E) é um grafo J=(P,Q) em que P $\subseteq$ V e Q $\subseteq$ E. Um subgrafo $J^{'}$ de G é induzido pelo subgrafo J se $J^{'} = J \cup \{(i, j) \in G : i, j \in J \wedge (i, j) \notin J\}$.

2.3 Definição

O problema da árvore de Steiner em grafos (*Steiner Tree Problem* - STP) consiste em determinar a árvore de custo mínimo, T^*, de um grafo G=(V,E) não dirigido, que interliga todos os nós de um conjunto predefinido, $S \subset V$, designado por conjunto dos nós terminais. Os restantes nós da árvore pertencentes a V\S são designados por nós de Steiner ou nós opcionais. No grafo representado na Figura 2.2, para o conjunto de nós terminais S={1,5,6}, a árvore de Steiner é aquela assinalada a cor verde e tracejado.

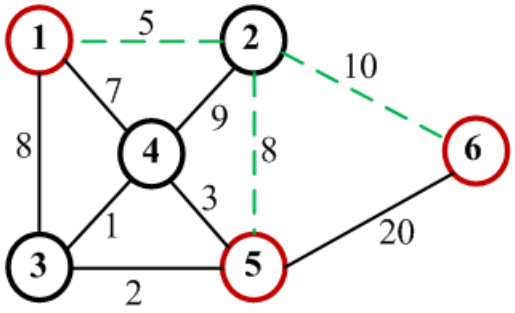


Figura 2.2: Árvore de Steiner para S={ 1,5,6 }.

Considerando Z, o conjunto de todas as árvores possíveis T que interligam todos os nós S sobre o grafo G, recorrendo eventualmente a nós de Steiner, o problema de Steiner em grafos pode ser definido da seguinte forma:

$$\min_{T \in Z} C(T) = \sum_{(i,j) \in T} C_{ij} \tag{2.1}$$

2.4 Complexidade

O STP é considerado um problema NP-Completo [7], não se conhecendo nenhum algoritmo com complexidade polinomial que resolva este problema com exatidão. Na prática são usados métodos aproximados com o propósito de obter árvores com custos próximos da solução ótima.

Existem dois casos particulares na resolução do STP, uma quando |S|=2 e outra quando |S|=|V|. No caso de existirem somente dois nós terminais, |S|=2, o STP resume-se ao *problema do caminho mais curto* entre um par de nós cuja a solução exata pode ser obtida através do algoritmo de Dijkstra (ver anexo A.2). No segundo caso, quando todos os nós do grafo são nós terminais, |S|=|V|, o STP resume-se ao *problema da árvore abrangente miníma* e o algoritmo de Prim (ver anexo A.3) ou de Kruskal [1] permitem encontrar a solução ótima.

Para obter boas aproximações à solução ótima, na resolução do STP, usam-se heurísticas e meta-heurísticas para escolher os nós de Steiner que podem melhorar a solução, isto é, o objetivo é encontrar um conjunto de nós $V' \subset V \backslash S$ em que $V' \cup S$ conjuntamente com os arcos que os interligam oferecem-nos uma boa aproximação da solução ótima à árvore de Steiner. Nesta dissertação é usada a meta-heurística proposta por Michel Gendreau, et al. [5], que por sua vez recorre à heurística proposta por Takahashi e Matsuyama [3], com uma alteração proposta por Rayward-Smith and Clare [4], para obtenção de um conjunto de soluções iniciais, recorrendo posteriormente a Tabu-search por forma a aproximar as soluções previamente obtidas da solução ótima.

O Tabu-search foi elaborado pela primeira vez por Fred Clover [17] em 1986. O método permite superar o problema de se ficar "preso"a um ótimo local em problemas de otimização o que impede de encontrar um ótimo global. Este, baseia-se na exploração de um conjunto de soluções, vizinhança, movimentando em cada iteração a solução atual para uma outra solução melhor localizada na sua vizinhança memorizando as soluções já visitadas. Define-se vizinhança como um conjunto de todas as soluções possíveis obtidas ao modificar adequadamente a solução atual.

2.5 Características básicas da meta-heurística apresentada em [5]

A meta-heurística proposta em [5] começa por reduzir o tamanho do grafo, aplicando as reduções descritas na subsecção 2.5.1. No grafo reduzido são determinadas todas as soluções iniciais para a meta-heurística conforme a subsecção 2.5.2. Para explorar o conjunto de soluções iniciais, existem dois métodos:

i) O P-Tabu (pesquisa local) descrita na subsecção 2.5.3 ;

ii) O F-Tabu (pesquisa global) descrita na subsecção 2.5.5.

2.5.1 Reduções

A dimensão da rede é reduzida, aplicando as reduções descritas de seguida. Através dessas reduções, obtém-se os arcos e os nós de Steiner entre os quais se encontram aqueles que farão parte da solução ótima de Steiner. Algumas vezes a árvore ótima de Steiner é encontrada ao reduzir o grafo.

Antes de se descrever as reduções, apresenta-se o conceito de contração de um grafo a partir de um arco.

Uma contração de G *"ao longo"* do arco (i, j) consiste no seguinte [14]:

- Os arcos incidentes em i ficarão incidentes em j.
- O nó i é removido de G e se i for um nó terminal, consequentemente j passa a ser considerado nó terminal.
- Os ciclos incidentes em j são removidos de G, i.e. se j ficar com um par de arcos paralelos incidentes, o arco com maior custo é removido.

As reduções consideradas são:

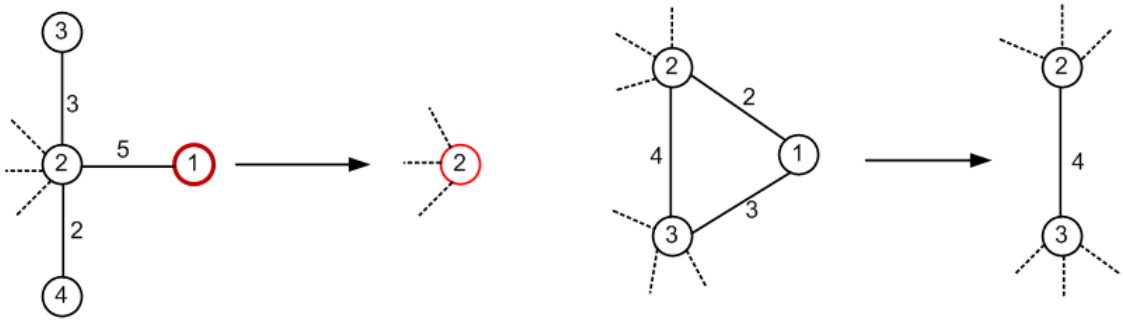
a. Todos os nós de grau 1 podem ser removido a partir de G, bem como o respetivo arco incidente. Caso o nó removido seja um nó terminal, então o arco removido será adicionada a árvore de Steiner ótima e o outro nó extremo do arco passa a ser considerado nó terminal. Ilustrado na Figura 2.3(a), onde os nós terminais estão assinalados a vermelho.

b. Se o nó w for um nó de Steiner de grau 2, os arcos (i, w), (w, j) são substituídas por um arco (i, j) com custo igual ao $\min\{C_{ij}, C_{iw} + C_{wj}\}$. Ilustrada na Figura 2.3(b).

c. Se d_{ij} for o custo do caminho mais curto entre i e j, o arco (i, j) com $C_{ij} > d_{ij}$ pode ser removido a partir de G. Por outro lado se $C_{ij} = d_{ij}$ e o caminho mais curto não contiver o arco (i, j), este arco também pode ser removido. Ilustrada na Figura 2.3(c), o arco (2,3) é removido de G, pois $C_{23} > d_{23}$.

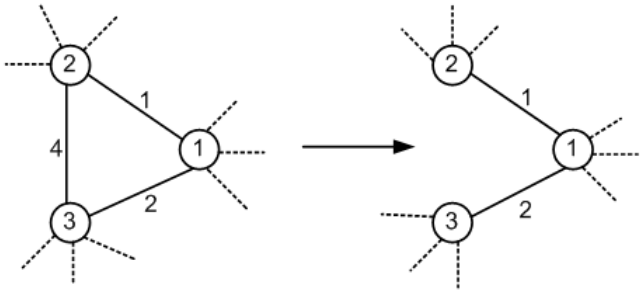
d. Sendo i um nó terminal de G e sendo j e k o primeiro e o segundo nós (de Steiner ou terminais) mais próximos do nó i, respetivamente, então se $C_{ij} + min\{d_{jw} | w \in S \wedge w \neq i\} \leq C_{ik}$ o arco (i, j) vai fazer parte da árvore de Steiner ótima e G pode ser contraída ao longo do arco (i, j). Ilustrada na Figura 2.3(d), os nós 3 e 2 são o primeiro e o segundo nós mais próximos do nó 1 e como $C_{13} + min\{d_{3,4}, d_{3,5}\} \leq C_{12}$ então o arco (1,3) é removido de G e passa a fazer parte da árvore de Steiner ótima e G é contraída ao longo do arco (1,3).

A ordem em que são feitas as reduções neste trabalho é a mesma que foi proposta por Esbensen [12], c→b→d→a. A sequência é repetida até não ser possível reduzir mais o grafo.

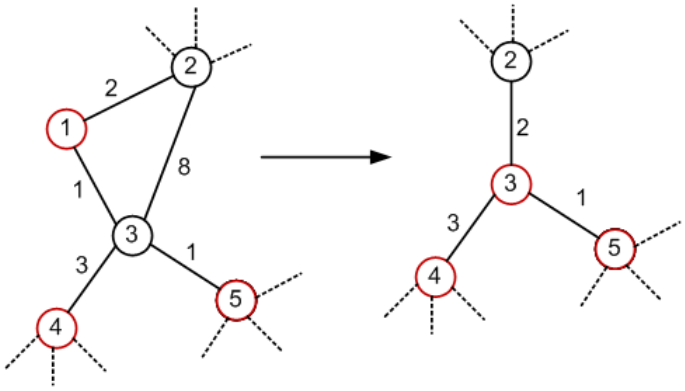


(a) Redução a.

(b) Redução b.



(c) Redução c.



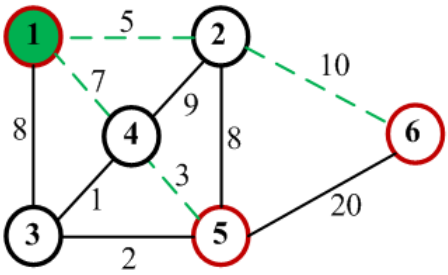
(d) Redução d.

Figura 2.3: Exemplificação das reduções.

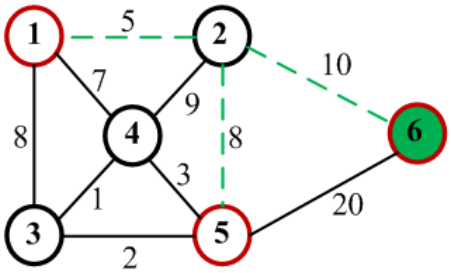
2.5.2 Algoritmo proposto por Takahashi e Matsuyama

O *Minimum Path Heuristic - MPH* proposto por Takahashi e Matsuyama [3] é utilizada para determinar o conjunto de soluções inicias para a meta-heurística. Deve-se referir que esta heurística apresenta melhores resultados e tempo de execução do que a heurística descrita por Kou, et al. [10]. O algoritmo baseia-se na construção progressiva da árvore de Steiner ótima a partir de um nó terminal. A árvore ótima de Steiner vai sendo construída adicionando, através do caminho mais curto, o nó terminal mais próximo da árvore obtida

anteriormente (pode incluir ou não nós de Steiner). Portanto o custo da árvore obtida pelo algoritmo varia de acordo com a escolha do nó terminal inicial. Na Figura 2.4 pode-se observar esta variação, pois tendo $S = \{1, 5, 6\}$, obtém-se árvores diferentes (assinaladas a verde e tracejado) ao escolher o nó 1 ou o nó 6 como sendo os nós iniciais.



(a) Algoritmo de Takahashi a partir do nó 1.



(b) Algoritmo de Takahashi a partir do nó 6.

Figura 2.4: Aplicação de Takahashi.

O algoritmo implementado e descrito a seguir liga o nó terminal mais próximo à arvore obtida até o momento, usando o algoritmo Dijsktra (ver anexo A.2), sendo que este termina quando todos os nós terminais estiverem na árvore.

Algoritmo 1: Takahashi

Entrada: grafo G=(V,E), $s_1 \in S$, S // *S representa o conjunto dos nós Terminais*

1: $T^*:=\emptyset$, S_{AUX} =S
2: $T^* \leftarrow T^* \cup \{s_1\}$
3: Remover $\{s_1\}$ de S_{AUX}
4: **Enquanto** $|S_{AUX}| \neq 0$ **faz**
5: Custo-$SP_{min}=\infty$
6: **Para** cada $s_j \in T^*$
7: Dijkstra(s_j, S_{AUX})
8: **Para** cada $s_i \in S_{AUX}$
9: SP=Caminho mais curto de s_i para s_j
10: **Se** SP != 0
11: **Se** Custo-SP < Custo-SP_{min}
12: SP_{min}=SP
13: Custo-SP_{min} = Custo-SP
14: Noaux1= s_i // *novo nó terminal que originou o caminho mais*
15: // *curto*
16: Noaux2= s_j // *nó da árvore que deu origem ao caminho mais*
17: // *curto*
18: **Senão**
19: Erro na leitura dos dados!
20: **Para** cada $s_k \in SP_{min} \backslash \{\text{Noaux2}\}$
21: Adiciona s_k em T^*
22: Remover Noaux1 de S_{AUX}
23: **Fim**

Método proposto por Rayward-Smith and Clare

O método proposto por Rayward-Smith and Clare [4] é acrescentado na heurística de Takahashi de forma a obter melhores resultados e consiste em:

1. Criar um grafo induzido para cada árvore de Steiner encontrada pelo algoritmo de Takahashi.

2. Aplicar o algoritmo Prim (descrito no anexo C) sobre o grafo induzido.

3. Eliminar todos os nós folha que não sejam nós terminais.

2.5.3 Tabu Search (P- Tabu)

No caso da abordagem apresentada em [5], o Tabu-Search parte de 100 soluções iniciais que são árvores abrangentes mínimas ($T_s \subset Z$), obtidas através do MPH descrita na subsecção 2.5.2.

Para cada solução inicial o objetivo do Tabu-Search consiste em explorar o espaço de pesquisa modificando em cada iteração a solução (árvore) atual para uma melhor na sua "vizinhança".

Neste contexto uma vizinhança N(T) é o conjunto de todas as árvores abrangentes possíveis mínimas que se obtêm removendo ou adicionando um único nó de Steiner da solução (árvore) atual.

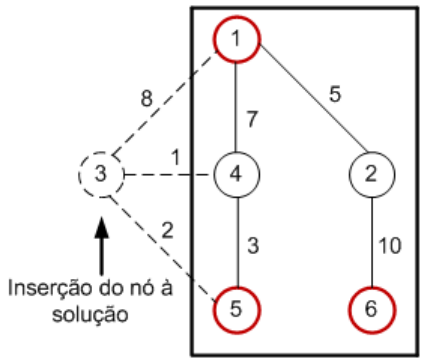
Quando um nó é escolhido, este fica inativo (é declarado tabu) durante um determinado número de iterações de modo a evitar-se o retorno a solução anterior.

Para uma maior eficiência em termos do algoritmo de manipulação das árvores, em cada iteração as novas soluções são obtidas por manipulação de uma estrutura de dados que guarda uma árvore com um nó raiz. A estrutura de dados e a manipulação feita é descrita em seguida.

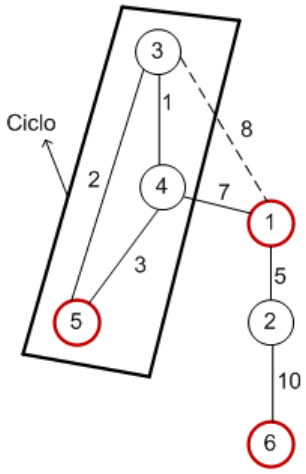
Manipulação da estrutura de dados onde são guardadas as árvores

A estrutura de dados considerada é uma estrutura em que um nó arbitrário é designado por raiz da árvore. Nesta estrutura cada nó está apontado para seu único "pai"que é o seu nó antecessor na árvore e o nó raiz é portanto o "pai"de todos os nós e por conseguinte é o único nó que não tem nó antecessor.

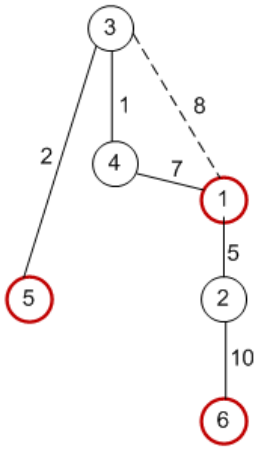
Quando se adiciona um nó a uma solução, são considerados todos os arcos que o interligam ao resto da solução. Um destes arcos incidentes é candidato a ser excluído para que possa obter uma árvore, ou seja, uma ligação sem ciclos (cada nó aponta para o seu único pai). Para eliminar os ciclos ao adicionar o nó na solução, começa-se por adicionar o arco incidente com menor custo e de seguida os outros (com uma sequência do menor para o maior custo). Em cada passo analisa-se a hipótese de haver ciclos. Inicialmente coloca-se o nó adicionado à solução como nó raiz para ser mais fácil analisar os ciclos causados pela adição deste nó. Caso haja ciclos, é removido o arco de maior custo por forma a quebrar o ciclo e a árvore é reorganizada. Na Figura 2.5 é ilustrada o processo descrito anteriormente correspondente à adição de um nó a uma árvore correspondente ao algoritmo 2. Na Figura 2.5 (a) o nó 3 é adicionado a árvore (realçada no retângulo). A Figura 2.5(b) mostra a formação do primeiro ciclo (identificado pelo retângulo) que é eliminado na Figura 2.5(c).



(a) Adição de um nó a solução.



(b) Aparecimento de ciclo.



(c) Eliminação do ciclo.

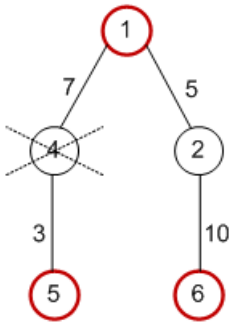
Figura 2.5: Processo de inserção de um nó na solução

Algoritmo 2: Adição de um nó á arvore

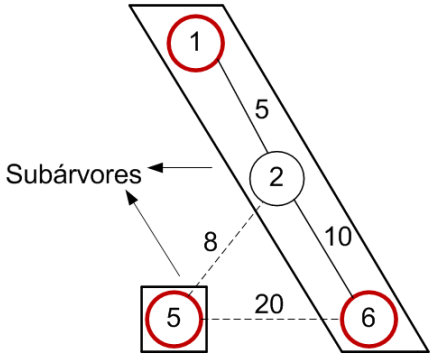
Entrada: v_1, G=(V,A) *// v_1 é o nó de Steiner a adicionar à árvore e G é o grafo original*
Saída: T *// árvore resultante com a adição do nó v_1*
1: Para cada arco incidente ao nó v_1 para qual o outro nó pertence a T **faz**
2: Guarda os arcos em um arraylist Aux e ordena-os por ordem crescente do custo
3: Escolhe o arco com menor custo e liga o nó v_1 a T através deste arco
4: Para cada arco $\in$ a Aux **faz**
5: Adiciona o arco a T
6: v_1 passa a nó raiz de T *//É feito o reroot da árvore, ou seja, a relação "pai – filho"é trocada até à raiz*
7: Elimina o arco de maior custo no ciclo resultante
8:Fim

Quando se remove um nó da solução, todos os arcos incidentes são removidos criando, por conseguinte, subárvores diferentes e separadas. Estas árvores são reconectadas usando os arcos existentes na rede. Se na solução existir w arcos incidentes conectados ao nó removido,

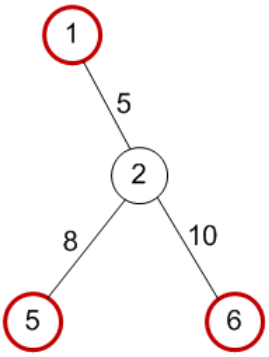
então são criadas w subárvores distintas e elas podem ser ligadas usando $w-1$ arcos existentes na rede. Usa-se os arcos que favorecem melhor a solução de modo a obter-se uma árvore abrangente mínima. A figura 2.6 ilustra o processo descrito para remoção de um nó na solução.



(a) Remoção de um nó a solução.



(b) Formação de subárvores.



(c) Obtenção de uma nova solução.

Figura 2.6: Processo de remoção de um nó a solução.

Para a remoção de um nó da árvore e a formação de subárvores e também a junção destas subárvores na obtenção de uma nova solução foram implementados os algoritmos 3 e 4.

O procedimento básico da meta-heurística, designada por P-Tabu pode ser descrito em três etapas:

1. Selecionar as 100 soluções (árvores) obtidas através do MPH (subsecção 2.5.2.).

2. Para cada uma das 100 soluções, são obtidas novas soluções através do procedimento básico de pesquisa tabu, adicionando ou removendo um nó de Steiner à solução atual. Se o nó existe na solução este é removido, caso contrário é adicionado. O nó escolhido fica inativo durante um certo número de iterações. O número total de iterações deste procedimento básico da pesquisa tabu, descrito no algoritmo 5, é o dobro do número

Algoritmo 3: Remoção de um nó de Steiner da árvore e formação de sub-árvores

Entrada: Árvore T, v_1 // v_1 *é o nó de Steiner a ser removido da árvore T*
Saída: Subárvores // *arraylist com todas as sub-árvores gerada*
1: NoEncontrado←false
2: **Para** cada $v_i \in$ T
3: **Se** !NoEncontrado
4: **Se** Pai do nó $v_i == v_1$
5: Erro! O nó a remover é "orfão". Grafo desconexo.
6: Termina a execução do programa
7: **Senão** $v_i == v_1$
8: Remove v_i de T
9: NoEncontrado←verdadeiro
10: **Senão**
11: **Se** Pai do nó $v_i == v_1$ // *Verifica se* v_i *é filho de* v_1
12: Cria uma nova sub-árvore em Subárvores com v_i como raíz
13: Adiciona v_i a esta sub-árvore
14: Remove v_i de T
15: **Senão** Para cada sub-árvore $\in$ Subárvores
16: **Se** Pai do $v_i \in$ sub-árvore
17: Adiciona v_i á sub-árvore
18: Remove v_i de T
19: **Se** T$\neq$ 0
20: Adiciona T à Subárvores // *Se T não ficar vazia , então é adicionada à Subárvores*
21: **Fim**

de nós da rede. A melhor solução em cada iteração é determinada tendo como critérios de avaliação a árvore inicial, a iteração em que cada nó deixa de ser considerado tabu e a diversificação contínua (descrita na subsecção seguinte).

3. A melhor solução encontrada pode ou não ser a solução ótima STP.

A iteração em que um nó w fica tabu, ou seja não pode ser considerado nem para ser adicionado nem para ser removido, é calculada da seguinte forma:

$$\textit{Tabu(w)} := \textit{iteração} + \textit{Random()} + 1 \ ,$$

onde *Random()* corresponde um número inteiro aleatório pertencente uma gama de valores (ver tabela 2.1) e a *iteração* corresponde a iteração corrente.

2.5.4 Mecanismos de diversificação

Os mecanismos da diversificação contínua e de caminhos são usados no procedimento P-Tabu, descrito anteriormente e no F-Tabu descrito na secção 2.5.5, respetivamente.

Tabela 2.1: Gama de números aleatórios de acordo ao número de nós do grafo

Número de nós no grafo	**Nº de iterações em que um nó é "tabu"**
≤30	4-10
≤ 80	4-14
≤150	8-18
≤350	12-27
>350	20-47

Algoritmo 4: União das sub-árvores que resultaram da remoção de um nó de Steiner, através dos arcos de menor custo

Entrada: Subárvores *// arraylist com todas as sub-árvores gerada apartir do algorítmo 3*
Saída: T *// árvore resultante da união das sub-árvores*
1: Para cada sub-árvore ∈ Subárvores **faz**
2: *//Considerar uma etiqueta para cada uma delas*
3: **Para** cada nó v_i ∈ à cada sub-árvore **faz**
4: Copiar a lista de arcos adjacentes a esse nó a partir do grafo original G=(V,A)
5: *//Criando uma lista de adjacências auxiliar*
6: Atribuir a etiqueta da sub-árvore ao nó
5: Para cada nova lista de adjacências e para cada arco **faz**
6: Eliminar os arcos cujos nós têm a mesma etiqueta
7: Eliminar os arcos que envolvam os nós que não existam em nenhuma das suas sub-árvores
8: $(i,j)^*$ ← arco com menor custo
9: Enquanto não haver uma única sub-árvore **faz**
9:*//Quando todos os nós da lista de adjacências auxiliar tiverem a mesma etiqueta*
10: Com o arco$(i,j)^*$ considerar a árvore de etiqueta mais baixa (associada ao nó i ou ao nó j), fazer o *reroot* da árvore de etiqueta mais elevada
11: T ← união das duas árvores através do arco $(i,j)^*$
12: Re-etiquetar os nós na lista de ajacências auxiliar, correspondentes à árvore de etiqueta mais mais elevada atribuindo-lhes a etiqueta mais baixa
13: **Para** cada nova lista de adjacências e para cada arco **faz**
14: Eliminar arcos cujos nós têm a mesma etiqueta *//Considerar apenas a menor etiqueta da união das 2 últimas árvores*
15: $(i,j)^*$ ← arco com menor custo
16:Fim

Diversificação Contínua

A diversificação contínua baseia-se nas memórias de médio e longo prazo relativamente à pesquisa da melhor solução. Esta diversificação consiste em remover os nós de Steiner que já fazem parte da solução há algum tempo, adicionando novos nós de Steiner com o propósito de permitir a obtenção de uma solução melhor. Isto é feito através de uma penalização atribuída ao nó:

- se o nó está para ser adicionado :

$$Penalização{:=}\log(n+1)$$

- se o nó está para ser removido :

$$Penalização{:=}-\log(m+1)$$

onde n corresponde o número de vezes que o nó tem estado em soluções anteriores e m corresponde o número de iterações que o nó tem estado continuamente na mesma solução.

Algoritmo 5: Obtenção de uma nova solução através do movimento tabu (etapa 2 do P-Tabu)

Entrada:T, G=(V,A) *// T é uma árvore obtida do MPH e G=(V,A) é o grafo original*
Saída: T* *// árvore resultante*
1: $Nx[V\backslash S] = 0$, T* ← T *// melhor árvore obtida até ao momento*
2: Custo_Aux=∞ , C(T*)= custo da árvore T*
3: Enquanto o número de iterações $< 2 * |V|$ **faz**
4: **Para** cada nó $i \in V\backslash S$**faz**
5: Custo_Árvore_Adicionar/Remover(i)= Custo da árvore ao adicionar ou remover o nó de Steiner i
6: **Se** Custo_Árvore_Adicionar/Remover(i) + Penalização≤ Custo_Aux e [Tabu(i)≤ iteração ou Custo_AuxÁrvore_AuxAdicionar/Remover(i) ≤ C(T*)] **faz**
7: Custo_Aux←Custo_Árvore_Adicionar/Remover(i) + Penalização
8: T*=Árvore_Adicionar/Remover(i)
9: Nx(i):=Nx(i)+1 *//Número de vezes que o nó w foi escolhido*
10: Tabu(i):= iteração +*Random*()
11: **Se** C(T) < C(T*) **faz**
12: T* ← T
13:Fim

Diversificação de caminhos

O mecanismo da diversificação de caminhos inicia-se quando não se verificar melhoria na solução após um certo número de iterações (tipicamente 20 iterações). A diversificação consiste em adicionar um novo caminho entre um par de nós $i-j$ da melhor solução encontrada até o momento[1]. Para todos os pares de nós $i-j$ da solução que pertencem as diferentes subárvores, são calculadas os custos dos caminhos entre esses pares de nós $i-j$. O par $i-j$ que apresentar a maior diferença entre o custo calculado e o custo do caminho mais curto no grafo original é atribuída um novo caminho igual ao caminho mais curto do grafo original. Os nós deste caminho que não pertencem à solução são então adicionados.

[1]É de notar que os nós i e j podem ser de qualquer tipo (de Steiner ou terminais).

A diversificação de caminhos pode ser descrita da forma seguinte :

$$\textbf{Se } itr\_melhor + itr\_paragem \leq \text{iteração } \textbf{então}$$
$$\text{T:=T*}\cup\ caminho(i,j)$$
$$itr\_melhor = \text{iteração ,}$$

onde, *itr_melhor* corresponde à iteração em que a solução melhorou pela última vez e *itr_paragem* corresponde ao número de iterações (20 iterações) sem se conseguir melhorar a solução antes de ativar a diversificação de caminhos. O $caminho(i,j)$ é o novo caminho adicionado à solução.

2.5.5 Full Tabu

O procedimento completo da meta-heurística, designado por F-Tabu (*Full Tabu*), pode ser descrito pelas seguintes etapas:

1. Selecionar as 100 soluções (árvores) obtidas através do MPH (subsecção 2.5.2).
2. Para cada uma das soluções é aplicada o P-Tabu com um número máximo de iterações igual a 20.
3. Para a melhor solução obtida na etapa anterior, aplica-se o P-Tabu com a diversificação de caminhos e com um número máximo de iterações sem melhoria igual ao dobro do nº de nós no grafo.
4. A melhor árvore encontrada é a solução.

2.5.6 Resultados obtidos

A presente secção é dedicada a apresentação dos resultados obtidos com as redes B, C, D e E da Steinlib [15]. Para as redes C, D e E é feita a comparação com os resultados obtidos em [5] por forma a testar a correção dos algoritmos implementados.

Nas tabelas 2.2 - 2.5 são apresentados os resultados da redução das redes B, C, D e E. Com base nestas tabelas foi verificado que em algumas redes os resultados da redução são próximos daqueles obtidos em [5], mas por outro lado foi também constatado que existe um número, embora muito pequeno, de redes em que há uma grande discrepância nos resultados (por exemplo nas redes c20, d20 e e20). Mas se a comparação for feita com os resultados apresentados em [12] estes já são muitos próximos dos agora obtidos. No caso das redes b03 e b09 foi possível encontrar a solução ótima somente com as reduções implementadas.

As tabelas 2.6 - 2.9 ilustram os resultados de MPH, P-Tabu e F-Tabu obtidas com as redes B, C, D e E. Nestas tabelas a indicação "*orig*"diz respeito aos resultados obtidos em [5] e as indicações $\%Opt/MPH$, $\%Opt/PTabu$ e $\%Opt/FTabu$ referem-se às diferenças relativas aos ótimos (em percentagem) obtidas por cada um dos algoritmos (MPH, P-Tabu e F-Tabu). Relativamente ao MPH é possível constatar que os resultados são coerentes com os apresentados em [5]. Os resultados de P-Tabu e F-Tabu, de um modo geral, são ligeiramente diferentes dos apresentados em [5], mas em algumas redes são iguais. Estas diferenças podem ser justificadas com base nos resultados obtidos através das reduções, uma vez que foi também verificado que os resultados das reduções são ligeiramente diferentes. Isto deve-se ao facto do P-Tabu e F-Tabu serem fortemente dependentes dos nós de Steiner existente no grafo reduzido.

Tabela 2.2: Reduções das redes B.

	Redes Originais			**Redes Obtidos**		
	$\|V\|$	$\|S\|$	$\|E\|$	$\|V\|$	$\|S\|$	$\|E\|$
b01	50	9	63	7	3	12
b02	50	13	63	4	7	12
b03	50	25	63	0	0	0
b04	50	9	100	39	8	83
b05	50	13	100	37	10	79
b06	50	25	100	25	10	60
b07	75	13	94	18	6	29
b08	75	19	94	16	7	25
b09	75	38	94	0	0	0
b10	75	13	150	50	10	115
b11	75	19	150	49	9	114
b12	75	38	150	34	13	80
b13	100	17	125	29	10	48
b14	100	25	125	29	11	51
b15	100	50	125	18	10	31
b16	100	17	200	71	12	160
b17	100	25	200	50	11	108
b18	100	50	200	41	15	104

Tabela 2.3: Reduções das redes C.

	Redes Originais			**Redes em [5]**			**Redes Obtidos**		
	$\|V\|$	$\|S\|$	$\|E\|$	$\|V\|$	$\|S\|$	$\|E\|$	$\|V\|$	$\|S\|$	$\|E\|$
c01	500	5	625	145	5	263	145	5	263
c02	500	10	625	130	8	239	130	8	239
c03	500	83	625	125	39	237	131	39	243
c04	500	125	625	116	42	233	117	46	244
c05	500	250	625	47	24	117	43	22	107
c06	500	5	1000	369	5	847	397	5	847
c07	500	10	1000	382	9	869	382	9	869
c08	500	83	1000	335	53	817	359	58	843
c09	500	125	1000	351	80	834	351	80	834
c10	500	250	1000	205	68	603	220	74	632
c11	500	5	2500	499	5	2184	499	5	2184
c12	500	10	2500	498	9	2238	498	9	2238
c13	500	83	2500	458	60	2066	463	65	2139
c14	500	125	2500	414	68	1886	433	78	1957
c15	500	250	2500	275	70	1276	303	96	1428
c16	500	5	12500	500	5	4740	500	5	4740
c17	500	10	12500	500	10	4704	499	9	4698
c18	500	83	12500	483	67	4637	489	73	4677
c19	500	125	12500	433	58	3431	473	98	4490
c20	500	250	12500	262	19	1692	388	145	3864

Tabela 2.4: Reduções das redes D.

	Redes Originais			**Redes em [5]**			**Redes Obtidos**		
	$\|V\|$	$\|S\|$	$\|E\|$	$\|V\|$	$\|S\|$	$\|E\|$	$\|V\|$	$\|S\|$	$\|E\|$
d01	1000	5	1250	274	5	510	274	5	510
d02	1000	10	1250	285	10	523	285	10	523
d03	1000	167	1250	228	70	445	271	85	506
d04	1000	250	1250	180	81	376	184	75	378
d05	1000	500	1250	118	67	282	113	56	272
d06	1000	5	2000	761	5	1741	761	5	1741
d07	1000	10	2000	754	10	1735	754	10	1735
d08	1000	167	2000	732	124	1711	758	132	1737
d09	1000	250	2000	660	157	1620	706	176	1678
d10	1000	500	2000	407	136	1282	407	136	1282
d11	1000	5	5000	993	5	4674	993	5	4674
d12	1000	10	5000	999	9	4669	999	9	4669
d13	1000	167	5000	916	116	4374	947	130	4495
d14	1000	250	5000	840	147	4048	853	160	4173
d15	1000	500	5000	525	135	2655	587	177	3044
d16	1000	5	25000	1000	5	10595	1000	5	10595
d17	1000	10	25000	999	9	10531	1000	10	10548
d18	1000	167	25000	944	111	9331	974	141	10097
d19	1000	250	25000	874	129	7943	938	193	9676
d20	1000	500	25000	523	42	3341	799	309	8749

Tabela 2.5: Reduções das redes E.

	Redes Originais			**Redes em [5]**			**Redes Obtidos**		
	$\|V\|$	$\|S\|$	$\|E\|$	$\|V\|$	$\|S\|$	$\|E\|$	$\|V\|$	$\|S\|$	$\|E\|$
e01	2500	5	3125	680	5	1286	680	5	1286
e02	2500	10	3125	710	9	1328	710	9	1328
e03	2500	417	3125	658	213	1257	725	235	1334
e04	2500	625	3125	471	189	1010	471	189	1010
e05	2500	1250	3125	290	161	767	260	133	707
e06	2500	5	5000	1845	5	4318	1845	5	4318
e07	2500	10	5000	1891	10	4372	1891	10	4372
e08	2500	417	5000	1728	286	4202	1807	314	4287
e09	2500	625	5000	1612	359	4071	1604	352	4049
e10	2500	1250	5000	1023	343	3348	1046	366	3389
e11	2500	5	12500	2498	5	12093	2498	5	12093
e12	2500	10	12500	2500	10	12123	2500	10	12123
e13	2500	417	12500	2340	320	11745	2381	329	11835
e14	2500	625	12500	2100	349	11106	2190	393	11459
e15	2500	1250	12500	1362	348	7636	1509	449	8722
e16	2500	5	62500	2500	5	29332	2500	5	29032
e17	2500	10	62500	2500	10	29090	2500	10	29090
e18	2500	417	62500	2402	328	27876	2433	359	28421
e19	2500	625	62500	2095	229	16490	2353	487	27715
e20	2500	1250	62500	1337	108	9506	1988	758	24423

Tabela 2.6: Qualidade das soluções nas redes B.

Rede	Otp	$\%Opt/MPH$	$\%Opt/PTabu$	$\%Opt/FTabu$
b01	82	0	0	0
b02	83	0	0	0
b03	138	0	0	0
b04	59	0	0	0
b05	61	0	0	0
b06	122	1,64	1,64	1,64
b07	111	0	0	0
b08	104	0	0	0
b09	220	0	0	0
b10	86	0	0	0
b11	88	0	0	0
b12	174	0	0	0
b13	165	4,24	3,03	3,03
b14	235	0	0	0
b15	318	0	0	0
b16	127	0	0	0
b17	131	0	0	0
b18	218	0	0	0

Tabela 2.7: Comparação da qualidade das soluções dos diferentes algoritmos para as redes C relativamente ao [5]

.

Rede	**Opt**	$\%Opt/MPH_{orig}$	$\%Opt/MPH$	$\%Opt/PTabu_{orig}$	$\%Opt/PTabu$	$\%Opt/FTabu_{orig}$	$\%Opt/FTabu$
c01	85	0	0	0	0	0	0
c02	144	0	0	0	0	0	0
c03	754	0	0	0	0	0	0
c04	1079	0,09	0,09	0	0	0	0
c05	1579	0	0	0	0	0	0
c06	55	0	0	0	0	0	0
c07	102	0	0	0	0	0	0
c08	509	0	0,39	0	0,39	0	0
c09	707	0,99	0,99	0,14	0,71	0,14	0,71
c10	1093	0,09	0	0	0	0	0
c11	32	0	0	0	0	0	0
c12	46	0	0	0	0	0	0
c13	258	0,78	1,55	0	1,16	0	0,78
c14	323	1,24	1,24	0,31	0,93	0,31	0,31
c15	556	0,18	0,18	0	0	0	0
c16	11	0	0	0	0	0	0
c17	18	0	0	0	0	0	0
c18	113	5,31	2,58	0,88	0,88	0	0,88
c19	146	4,79	2,05	0,68	0,68	0	0,68
c20	267	0,37	0	0	0	0	0

Tabela 2.8: Comparação da qualidade das soluções dos diferentes algoritmos para as redes D relativamente ao [5]

.

Rede	**Opt**	$\%Opt/MPH_{orig}$	$\%Opt/MPH$	$\%Opt/PTabu_{orig}$	$\%Opt/PTabu$	$\%Opt/FTabu_{orig}$	$\%Opt/FTabu$
d01	106	0	0	0	0	0	0
d02	220	0	0	0	0	0	0
d03	1565	0,77	0,58	0,26	0,48	0,26	0,48
d04	1935	0,16	0	0	0	0	0
d05	3250	0,06	0,12	0	0	0	0
d06	67	0	4,48	0	4,48	0	4,48
d07	103	0	0	0	0	0	0
d08	1072	1,59	2,14	0,47	1,87	0,37	0,93
d09	1448	0,83	1,38	0,41	1,24	0,21	0,69
d10	2110	0,38	0,38	0	0,38	0	0,38
d11	29	0	0	0	0	0	0
d12	42	0	0	0	0	0	0
d13	500	2	1,8	0	1,2	0	1,2
d14	667	0,75	1,49	0,15	1,04	0,15	1,04
d15	1116	0,36	0,45	0	0	0	0
d16	13	0	0	0	0	0	0
d17	23	0	0	0	0	0	0
d18	223	6,28	2,69	1,35	1,79	0,9	1,79
d19	310	5,81	2,58	1,68	1,29	0,32	0,97
d20	537	0,19	0,19	0	0	0	0

Tabela 2.9: Comparação da qualidade das soluções dos diferentes algoritmos para as redes E relativamente ao [5]

.

Rede	**Opt**	$\%Opt/MPH_{orig}$	$\%Opt/MPH$	$\%Opt/PTabu_{orig}$	$\%Opt/PTabu$	$\%Opt/FTabu_{orig}$	$\%Opt/FTabu$
e01	111	0	0	0	0	0	0
e02	214	0	0,93	0	0,93	0	0,93
e03	4013	1,07	0,85	0,42	0,85	0,32	0,25
e04	5101	0,18	0,2	0	0	0	0
e05	8128	0,02	0,02	0	0	0	0
e06	73	0	4,11	0	4,11	0	4,11
e07	145	2,07	2,76	2,07	2,76	0	2,76
e08	2640	1,63	0,83	0,49	0,83	0,3	0,39
e09	3604	1,17	1,19	0,42	0,89	0,11	0,78
e10	5600	0,21	0,41	0,04	0,2	0,04	0,16
e11	34	0	0	0	0	0	0
e12	67	1,49	1,49	1,49	1,49	0	1,49
e13	1280	1,88	2,81	0,78	2,1	0,55	2,03
e14	1732	1,04	1,32	0,29	1,09	0,23	0,92
e15	2784	0,22	0,22	0,11	0,11	0,07	0,07
e16	15	0	0	0	0	0	0
e17	25	0	0	0	0	0	0
e18	564	7,62	7,62	2,66	3,01	1,06	2,48
e19	758	4,35	4,22	1,19	1,58	1,19	1,45
e20	1342	0,67	0,6	0	0,15	0	0

Capítulo 3

Árvore de Steiner Bicritério

3.1 Introdução

A formulação do problema de Steiner bicritério que se apresenta neste capítulo resultou da necessidade prática de se encontrar a interligação de custo mínimo entre um subconjunto de nós, numa rede de telecomunicações, utilizando o número mínimo de *links*, onde cada *link* tem associado um custo que pode refletir, por exemplo, a ocupação do *link*. Por um lado pretende-se obter uma árvore de Steiner com o custo mínimo mas, por outro, lado pretende-se também uma árvore o mais curta possível (ou seja, com um número mínimo de *links*, ou com *hop count* mínimo). Neste contexto a árvore de menor custo pode ter um número elevado de *links* e a árvore com o número mínimo de *links* pode ter um custo muito elevado. Neste tipo de problemas de otimização multicritério normalmente procura-se obter um conjunto de soluções (soluções não dominadas ou soluções ótimas de Pareto) ao invés de uma única solução ótima (ou de um conjunto de ótimos alternativos) pois a solução que optimizaria simultaneamente as duas funções objetivo ou não existe ou não é admissível.

Neste capítulo, na secção 3.2 são introduzidas algumas definições básicas referentes a otimização multicritério e é apresentada a formulação do problema de Steiner bicritério. Na secção 3.3 é apresentada uma primeira adaptação, para o caso bicritério, da meta-heurística descrita no capítulo anterior. Finalmente na secção 3.4 são apresentados os resultados obtidos com esta nova abordagem.

3.2 Descrição do Problema

O problema da árvore de Steiner Bicritério (BCSTP – BiCriteria Steiner Tree Problem) consiste em determinar a árvore de custo mínimo que interliga todos os nós terminais, S, usando o menor número possível de arcos. Cada arco está associado a dois custos aditivos:

- $C_{ij}^1 \rightarrow$ custo associado ao arco (i, j)
- $C_{ij}^2 \rightarrow$ *hopcount* (custo unitário para cada arco).

O custo C^1 é igual ao que foi considerado no problema STP, descrito no capítulo anterior. O custo C^2 permite determinar o número de *links* (ou arcos) constituintes da árvore T.

A formulação do problema da árvore de Steiner bicritério é então a seguinte :

$$\min_{T \in Z} C^1(T) = \sum_{(i,j) \in T} C_{ij}^1 \tag{3.1}$$

$$\min_{T \in Z} C^2(T) = \sum_{(i,j) \in T} C_{ij}^2 \tag{3.2}$$

Seguem-se duas definições importantes:

Definição 1 *Num problema de minimização, uma solução T domina outra solução T' se e só se* $(C^1(T), C^2(T)) \leq (C^1(T'), C^2(T'))$[1]*, sendo a desigualdade anterior estrita para pelo menos uma das funções objetivo. Por outras palavras, T' diz-se dominada por T se ao passar de T' para T diminuir o valor de uma das funções objectivo, sem aumentar o valor da outra função.*

Definição 2 *Uma solução T é considerado uma solução não dominada ou solução ótima de Pareto (também designada por solução eficiente ou não inferior) se não existir outra solução* $T' \in Z$ *que melhore simultaneamente as duas funções objetivo.*

A Figura 3.1 ilustra o conceito de dominância para um problema de minimização bicritério.

É de notar que pode haver casos em que a árvore correspondente à solução ótima para uma das funções objetivo coincide com a árvore que também é a solução ótima para outra função objetivo, embora em geral não exista uma árvore T $\in$ Z que otimize simultaneamente as duas funções objetivo definidas através das equações (3.1) e (3.2). Sendo assim, em geral não existe uma solução ótima, mas sim um conjunto de soluções não dominadas ou soluções ótimas de Pareto. Então o problema resume-se a encontrar o conjunto de soluções não dominadas, ou seja, todas as árvores T $\in$ Z tal que :

$$(C^1(T), C^2(T)) \leq (C^1(T'), C^2(T')), \forall T' \in Z, \tag{3.3}$$

[1]Considera-se que $(a, b) \leq (c, d)$ sse $a \leq c$ e $b \leq d$.

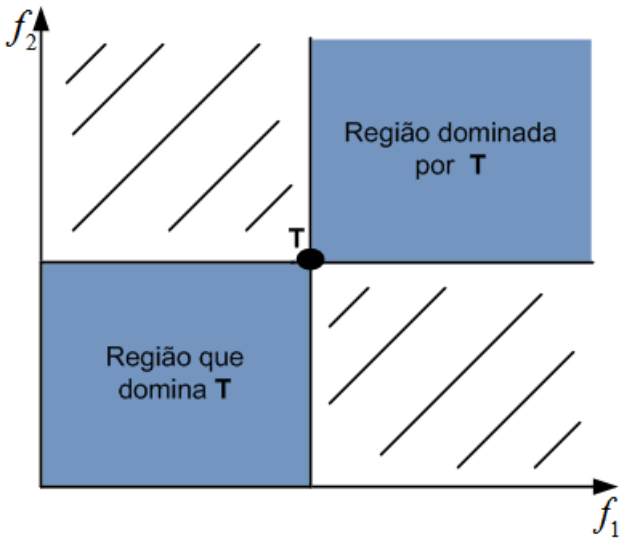


Figura 3.1: Conceito de dominância e não dominância em um problema bicritério.

3.3 Primeira meta-heurística para o problema bicritério

A meta-heurística, designada por BCSTPF-Tabu, proposta nesta dissertação é a primeira adaptação da meta-heurística estudada no capítulo anterior e pode ser resumida nos seguintes passos:

1. O grafo inicial é reduzido aplicando-se apenas a redução a da subsecção 2.5.1 do capítulo 2. As reduções b, c e d não foram usadas neste problema bicritério por poderem conduzir a piores soluções do ponto de vista da métrica *hop count*.

2. A cada arco do grafo reduzido são de seguida associados dois custos C1 e C2. C1 é o custo já usado no problema monocritério de Steiner e C2 é *hop count* (custo unitário para cada arco).

3. Determinam-se as 100 soluções para o custo C1, usando o MPH.

4. Executa-se o P-Tabu no conjunto das 100 soluções e determina-se o conjunto das soluções não dominadas que vão sendo encontradas.

5. Executa-se o F-Tabu no conjunto das soluções não dominadas da etapa anterior e determina-se o novo conjunto de soluções não dominadas para o problema.

6. As soluções obtidas (no grafo reduzido) são finalmente transpostas para o grafo inicial, recalculando-se para cada solução o valor das duas métricas.

3.4 Resultados obtidos

Nesta secção, é apresentado um conjunto de resultados obtidos com o BCSTPF-Tabu. Para avaliar este algoritmo foram considerados os conjuntos de redes B, C, I080 e I640 da Steinlib [15]. Os resultados obtidos são comparados com os obtidos pelos algoritmos BCSTP(2-10) e BCSTP descritos em [19] que pretendem resolver o mesmo problema.

A heurística BCSTP(2-10) baseia-se na redução do grafo original a um subgrafo completo constituído por todos os nós terminais e os arcos que os interligam. Cada arco no subgrafo corresponde ao caminho mais curto, no grafo completo, entre cada par de nós terminais. Para cada par de nós no subgrafo podem existir no máximo 10 arcos paralelos e para os determinar utilizou-se o algoritmo MPS [21], que permite obter os k–caminhos mais curtos entre cada par de nós. Cada arco neste subgrafo tem associados dois custos, sendo um deles o custo do caminho e o segundo custo procura representar a importância potencial de cada arco na árvore de Steiner pretendida. Utiliza-se ainda o algoritmo de Kruskal para a obtenção de árvores abrangentes, que são soluções não dominadas do problema bicritério definido do subgrafo. Finalmente as árvores abrangentes são transpostas para o grafo original e são retirados os ciclos.

A heurística BCSTP baseia-se também na redução do grafo original a vários subgrafos completos constituídos pelos nós terminais e os arcos que os interligam. Cada arco no subgrafo (tal como no caso do BCSTP(2-10)) corresponde a um caminho no grafo original. A heurística considera dois arcos paralelos para cada par de nós terminais.

A análise dos resultados obtidos vai ser feita com base nas figuras apresentados a seguir, para cada uma das funções objetivo em separado. É de notar que o valor ótimo para o problema associado à função objetivo C^1 (equação 3.1) é conhecido (disponível na Steinlib [15]).

A figura 3.2 mostra as percentagens de ótimos obtido, para o custo 1, para cada tipo de redes referidas anteriormente, por cada uma das heurísticas. Nesta figura é possível observar que com o BCSTPF-Tabu obtém-se $94,4\%$ e 50% para as redes B e C, respetivamente. É importante referir que os valores encontrados são muito melhores que aqueles encontrados com as outras heurísticas. Para as redes I080 e I640 as percentagens de ótimos obtidos pelo BCSTPF-Tabu são piores do que os obtidos pela heurística BCSTP mas melhores do que os obtidos pela heurística BCSTP(2-10).

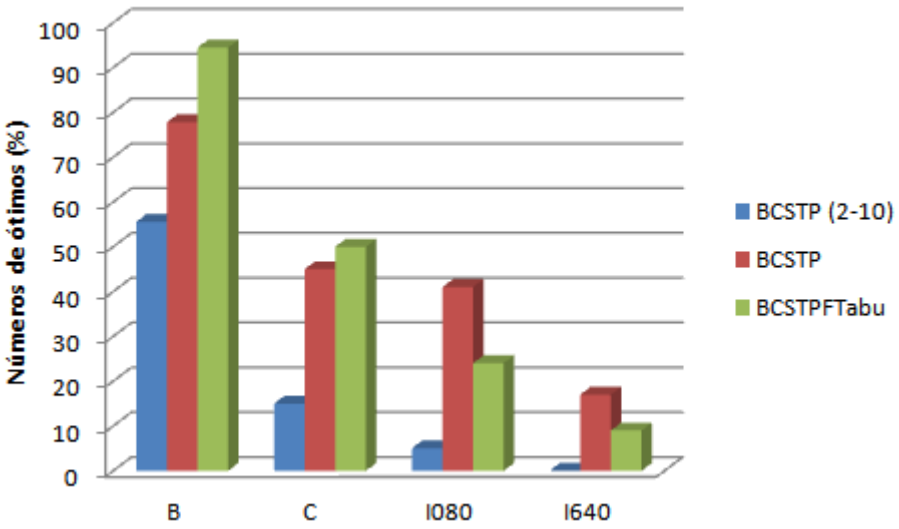


Figura 3.2: Número de ótimos obtidos.

A figura 3.3 mostra os desvios médios, para todo o conjunto de redes do mesmo tipo, da melhor solução obtida para o custo 1 (do conjunto de soluções não dominadas para cada rede), em percentagem, relativamente ao valor ótimo para cada uma redes, obtidos por cada uma das heurísticas. Nota-se que a meta-heurística implementada é muito melhor que a heurística BCSTP(2-10). Comparando os resultados com a heurística BCSTP é possível verificar que o BCSTPF-Tabu obtém os melhores resultados para as redes B, C e I640, sendo que para a rede I080 o resultado obtido é pior do que obtido pelo BCSTP.

A figura 3.4 mostra o número médio de soluções não dominadas obtido por cada uma das heurística para cada conjunto de redes. Nesta figura pode-se observar que o BCSTPF-Tabu é o que obtém maior número de soluções não dominadas para as redes I080 e I640 . Para as redes B e C o maior número de soluções não dominadas é obtido pela heurística BCSTP, sendo os resultados obtidos pelo BCSTPF-Tabu ligeiramente melhores do que os obtidos pelo BCSTP(2-10).

Ao contrário do custo 1, no caso do *hop count* não se conhecem as soluções ótimas, embora em alguns casos seja possível verificar que a solução, do ponto de vista da métrica *hop count*, é ótima, dado que as soluções obtidas tem $|S| - 1$ *links*, sendo $|S|$ o número de nós terminais, como se ilustra adiante. Não se conhecendo então essa solução determinou-se, para cada rede, no conjunto de todas as soluções obtidas por todas as heurística, o mínimo de *hop count* encontrado. Na figura 3.5 estão representados os desvios médios, relativos ao mínimo de *hop count* referido, em percentagem, obtidos por cada heurística para cada conjunto de redes. Nesta figura é possível notar que os resultados obtidos pelo BCSTPFTabu são piores do que os obtidos pelas outras heurísticas, sendo as diferenças mais significativas

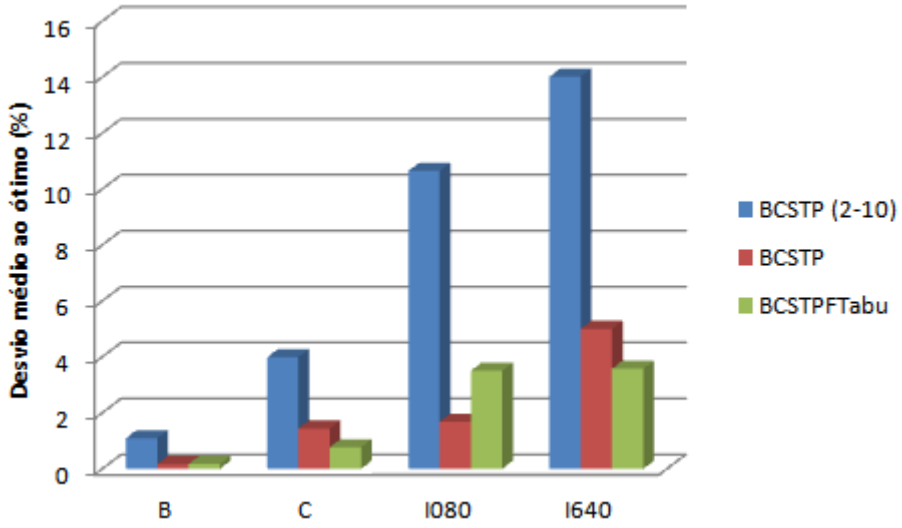


Figura 3.3: Desvio médio ao ótimo.

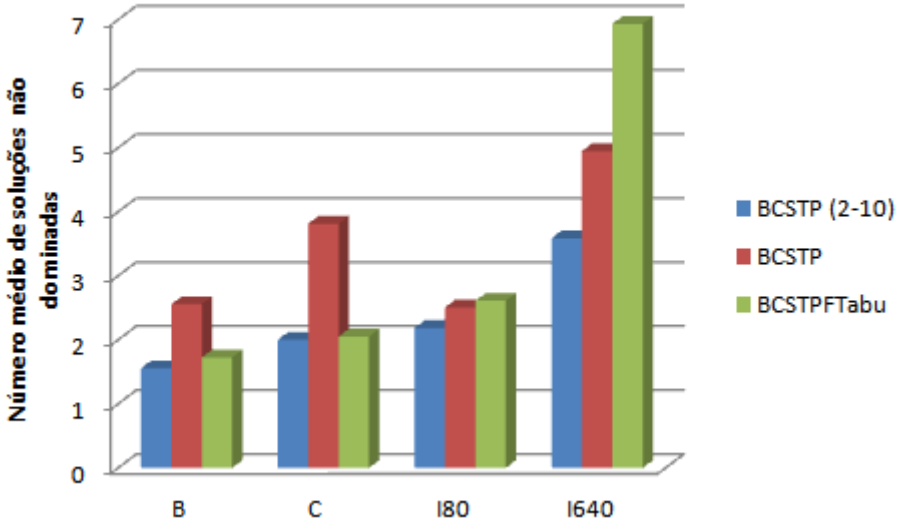


Figura 3.4: Número médio de soluções não dominadas.

no caso das redes B, C.

Na figura 3.6 pode-se observar uma das soluções obtida por BCSTPF-Tabu para uma das redes I080 (a solução correspondente ao valor mínimo *hop count*) que só contém nós terminais. Não tendo nenhum nó de Steiner, esta solução é ótima do ponto de vista do número de links existentes na solução. Para esta rede apenas se encontraram duas soluções não dominadas, estando a segunda solução representada na figura 3.7. Esta solução também é ótima do ponto de vista do custo 1 mas já tem um link adicional relativamente a outra solução referida.

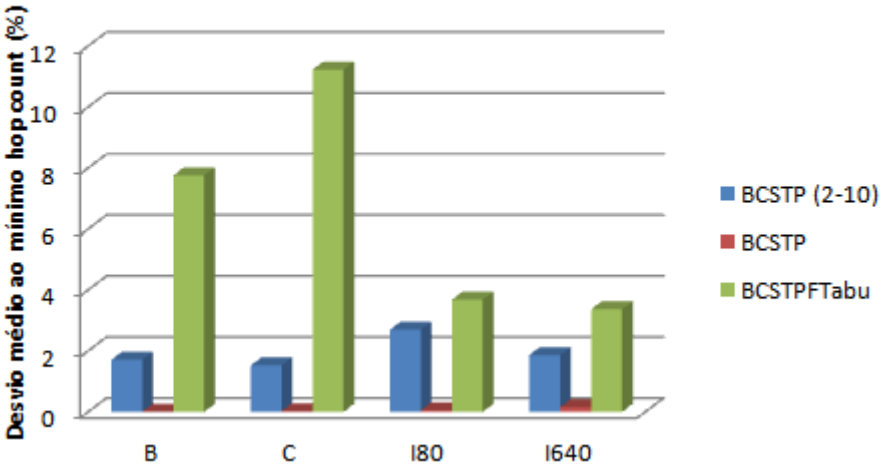


Figura 3.5: Desvio médio ao mínimo hop count.

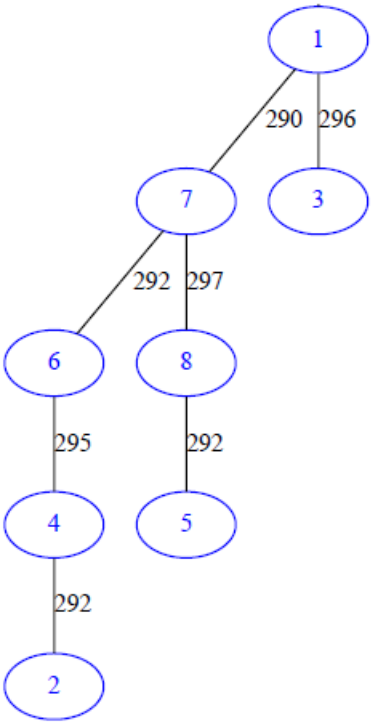


Figura 3.6: Rede I080-121, solução com mínimo hop count.

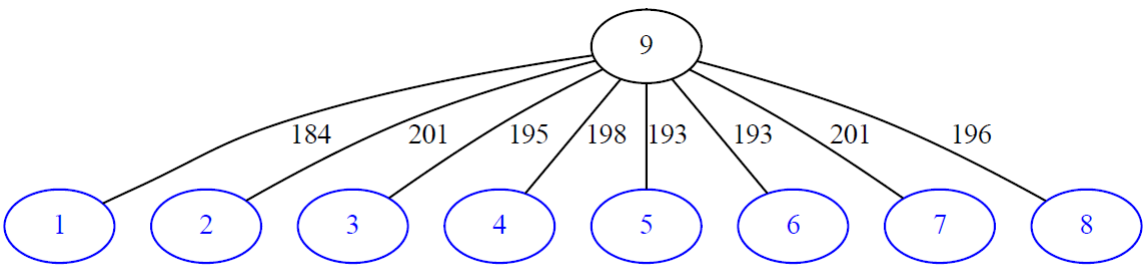


Figura 3.7: Rede I080-121, solução ótima.

Capítulo 4

Conclusão

Neste trabalho foi estudada uma meta-heurística conhecida [5] para a determinação de árvores de Steiner. Como existia uma implementação anterior desta meta-heurística, ela foi usada como ponto de partida, embora se tenha logo verificado que os resultados que esta implementação permitia obter não estavam corretos. Foi então necessário estudar em detalhe toda a meta-heurística de modo a corrigir alguns erros que existiam em alguns dos algoritmos que a constituem. Desenvolveram-se ainda novos algoritmos para substituir aqueles que tinham uma implementação mais duvidosa.

Como se mostra através dos resultados apresentados, estes já são bastante próximos dos obtidos pelos autores desta meta-heurística e bastante melhores dos que se obtiam inicialmente.

O tempo despendido nesta tarefa foi superior ao esperado e a extensão da meta-heurística anterior a um problema bicritério de Steiner, que era o objectivo inicial desta dissertação, ficou aquém do inicialmente previsto. Assim apenas se fizeram as alterações mínimas à heurística original adicionando-se um segundo custo aos arcos e adaptando toda a meta-heurística para ir guardando todas as soluções não dominadas que iam sendo encontradas.

A extensão já implementada da meta-heurística ao problema de Steiner Bicritério apresenta resultados satisfatórios ficando por implementar, por falta de tempo, uma parte significativa da extensão inicialmente prevista que, esperava-se, permitiria melhorar ainda os resultados obtidos.

Neste trabalho todos os algoritmos foram implementados recorrendo à linguagem JAVA.

Bibliografia

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows - Theory, Algorithms and Applications*. Prentice-Hall, Inc., 1993.

[2] Pawel Winter and J. MacGregor Smith. Path-Distance Heuristics for the Steiner Problem in Undirected Networks.*Algorithmica*,7 : 309 − 327, 1992

[3] Hiromitsu Takahashi and Akira Matsuyama. An approximate Solution for the Steiner problem in graphs. *Math. Japonica* 24, No.6, 573-577,1980

[4] V. J. Rayward-Smith and A. Clare. On finding steiner vertices. *Networks*, Volume 16, Issue 3,283-294,1986

[5] Michel Gendreau, Jean-Francois Larochelle and Brunilde Sanso. A Tabu Search Heuristic for the Steiner Tree Problem.*Networks*,34.2 (1999): 162-172.1986

[6] S. E. Dreyfus, R. A. Wagner. The steiner problem in graphs. *Networks*, Volume 1, Issue 3,283-294,1971

[7] Garey, M. and D. Johnson. Computers and Intractability.*A Guide to the Theory of NP-Completeness*, 1979

[8] Joseph B. Kruskal. On the shortese spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, Vol.7.No.1,Feb.,1956

[9] João Carlos Namorado Clímaco, Carlos Henggeler Antunes, Maria João Teixeira Gomes Alves. Programação Linear Multiobjectivo - do modelo de programação linear clássico à consideração explícita de várias funções objectivo. Imprensa da Universidade de Coimbra (ISBN 972-8704-13-5), 2003.

[10] L. Kou, G.Markowsky, and L.Berman. A Fast Algorithm for Steiner Trees. *Acta Informática*, 15,141-145,1981

[11] S. Voss. Steiner's problem in graphs. *Discrete Applied Mathematics*, Volume 40,Issue 1,45-72,1992

[12] Henrik Esbensen. Computing near-optimal solutions to the steiner problem in a graph using a genetic algorithm. *Networks*, Volume 16,Issue 4,173-185, December1995

[13] htt://metaheuristics.net

[14] Winter, Pawel. Steiner problem in networks: a survey. *Networks*, Volume 17, Issue 2,129-167,1987

[15] http://steinlib.zib.de/steinlib

[16] Glover, F. Future paths for integer programming and links to artificial intelligence. *Comput Oper Res 13*, 533-549., 1986

[17] Glover, F. Tabu Search – Part I. *ORSA Journal on Computing*, Vol. 1, No. 3, 190-206, 1989

[18] Glover, F. Tabu Search – Part II. *ORSA Journal on Computing*, Vol. 2, No. 1, 4-32, 1990

[19] Lúcia Martins and Nuno Gomes Ferreira. A bi-criteria approach for Steiner's tree problems in communication networks. *Proceedings of the 2011 International Workshop on Modeling, Analysis, and Control of Complex Networks. International Teletraffic Congress, 2011.*

[20] Lúcia Martins, Demilson Morais Gomes and Rui Rijo. Árvores de Steiner Bi-Critério. *Research report No. 10/2012 Inesc Coimbra.*

[21] E. Q. V. Martins, M. M. B. Pascoal, and J. L. E. Santos. Deviation algorithms for ranking shortest paths. *International Jounal of Foundations of Computer Science, 10:247-263, 1999*

Apêndice A

Anexos

De seguida estão descritas os algoritmos de Prim e de Dijkstra [1]. Como ambos requerem uma heap binária, esta é descrita previamente.

A.1 Heap binário

Uma heap, também designada por fila prioritária, é uma estrutura de dados que permite uma ordenação eficiente dos mesmos. A heap tem uma organização em árvores, cada elemento está associada a uma chave e os arcos representam um par, pai-filho, em que no caso de uma ordenação crescente, as chaves dos filhos são maiores ou iguais que a chave do pai . No caso da heap binária, cada nó possui no máximo dois filhos.

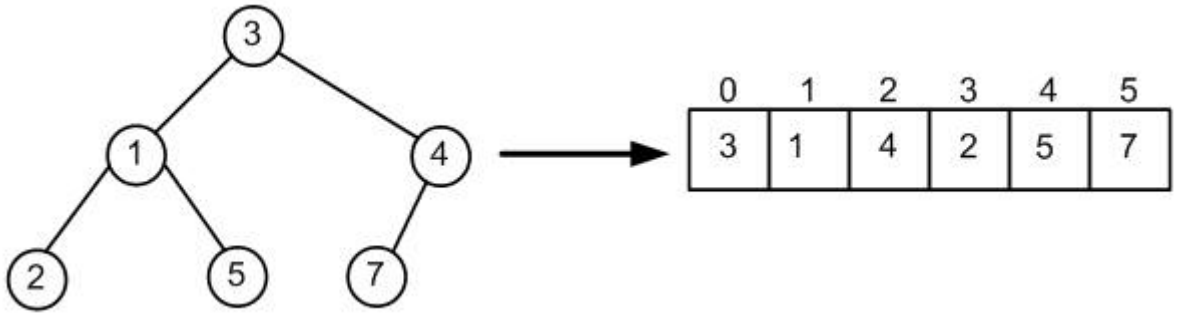


Figura A.1: Representação de uma heap binária.

Em termos computacionais, a estrutura da heap é vista como um vector. A figura A.1 ilustra a representação da heap binária organizada em árvore binária e na forma de um vector. A representação vectorial da árvore binária deve obedecer a seguinte propriedade:

Propriedade A. 1 *O pai (predecessor) de um nó na posição i está localizada na posição*

$\lfloor (i-1)/2 \rfloor$. *O nó raiz não tem pai e por convecção atribui um pai igual a* -1. *Os filhos (sucessores) de um nó na posição* i *estão localizados nas posições* $2i+1$ *e* $2i+2$.

A ordenação crescente do vector (heap) obedece a seguinte propriedade:

Propriedade A. 2 *A chave do nó* i *na heap é sempre menor ou igual as chaves dos filhos. O nó raíz possui a menor chave.*

Para reposicionar os elementos na heap por forma a respeitar a propriedade A.2 é necessário dois procedimentos:

a) *Reposicionar-acima (i)* $\rightarrow$ Aplicada quando se insere um novo elemento i com uma chave menor que o seu predecessor. Esta operação troca as posições do predecessor de i e i (no caso do nó i não ser o nó raiz) se i tiver uma chave menor ou igual à chave do seu predecessor.

b) *Reposicionar-abaixo (i)* $\rightarrow$ Aplicada quando se insere um novo elemento i com uma chave maior que a menor chave entre os seus sucessores. Neste caso (se o nó i não for um nó folha) troca as posições do nó i e do sucessor de i que tiver a menor chave.

A implementação da heap binária com uma ordenação crescente requer as seguintes operações [1]:

- Criar-Heap (H) $\rightarrow$ Cria um vector H (heap) vazio.
- Inserir(i, H) $\rightarrow$ Incrementa o tamanho da heap de uma unidade e armazena o nó i na última posição da heap. De seguida reposiciona-se os elementos da heap de forma a restaurar a ordenação usando o procedimento *Reposicionar-acima (i)*.
- Encontrar-Remover-Mínimo(i, H) $\rightarrow$ O nó i é o nó raiz segundo a propriedade A.2. Depois de remover o elemento i, coloca na raiz o nó que se encontrava na última posição da heap e de seguida reposiciona os elementos da heap usando o procedimento *Reposicionar-abaixo (i)*.
- Diminui-chave(valor, i, H) $\rightarrow$ Diminui a chave do nó i para valor caso este seja menor que a chave de i. Em seguida usa o procedimento *Reposicionar-acima (i)* para restaurar a ordenação.

A.2 Algoritmo de Dijsktra

O objetivo do algoritmo de Dijkstra [1] consiste em determinar o caminho mais curto entre um nó de origem e um nó de destino, entre um nó de origem e um sub-conjunto de nós destino ou entre um dado nó origem e todos os outros nós da rede. Cada arco da rede está associado a um valor não negativo designado por comprimento ou custo (aditivo) do arco. O caminho mais curto é portanto, o somatório dos custos dos arcos que constituem este caminho.

Neste algoritmo associa-se uma chave a cada nó, designado por etiqueta. A etiqueta pode ser:

• **Permanente** → Quando representa a verdadeira distância mais curta do nó de origem a um determinado nó.

• **Temporária** → Quando representa a distância mais curta provisório (calculado até o momento) a um determinado nó na rede.

Algoritmo 6: Dijkstra

Entrada: grafo G=(V,E), raiz s
1: Cria-heap(H)
2: d(j):=∞ com j $\in$ N *// chave dos vértices*
3: d(s):=0 e predecessor(s):=0 *// chave no nó raiz*
4: Inserir(s,H)
5: **Enquanto** H$\neq$0 **faz**
5: Encontrar-Remover-Mínimo(i,H)
6: **Para** cada (i,j) $\in$ E **faz**
7: Valor:=d(i)+C_{ij} // C_{ij} custo do arco (i,j)
8: **Se** d(j) > Valor **então**
9: **Se** d(j)=∞ **então** d(j):=Valor, pred(j):=i Inserir(i,H)
10: **Senão** d(j)=valor, pred(j):=i Diminuir-chave(valor, i, H)
11: **Fim**
12: **Fim**

Inicialmente atribui-se à raiz uma etiqueta permanente igual a zero e os restantes nós etiquetas temporárias igual a ∞. Em cada iteração, a etiqueta temporária dos nós ligados ao nó com etiqueta permanente, é a distância da raiz até estes nós. O nó que possuir a menor etiqueta temporária é o escolhido em cada iteração para passar a etiqueta permanente. O caminho vai sendo construído a partir dos nós com etiqueta permanente. A Figura A.2 ilustra aplicação do algoritmo Dijkstra de um nó para outro.

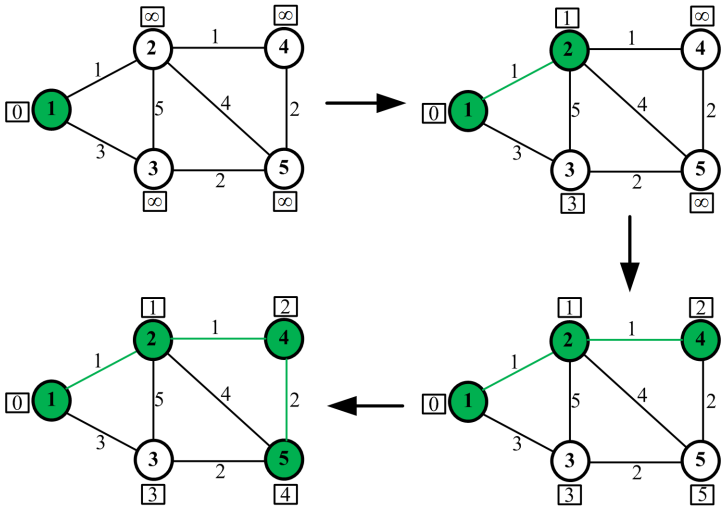


Figura A.2: Determinação do caminho mais curto do nó 1 para o nó 5 aplicando o Dijkstra

A.3 Algoritmo de Prim

O objectivo do algoritmo Prim[1] consiste em determinar a árvore abrangente miníma, ou seja aquela que contem o custo mínimo.

Algoritmo 7: Prim

Entrada: grafo G=(V,E), raiz s
1: Criar-heap(H)
2: **Para** cada N-{1} **faz** d(j):=∞ *// chave dos vértices*
3: d(1):=0 e predecessor(1):=0 *// chave da raiz*
4: **Para** cada j $\in$ N, insere todos os vértices na heap, excepto a raiz
5: **T***=vazio *//**T*** *armazena a solução (árvore abrangente miníma)*
6: **Enquanto** |**T***| < (n-1) **faz**
7: Encontrar-Remover-Mínimo(i, H)
8: **T***:=**T***$\bigcup$(predecessor(i),i) *// construindo a árvore abrangente miníma*
9: **Para** cada (i,j) $\in$ A(i) com j $\in$ H **faz**
10: **Se** d(j) > C_{ij}
11: d(j) := C_{ij}
12: predecessor(j) = i Diminuir-chave(i,valor,H)
13: **Fim**
14: **Fim**

Considerando a definição de corte (**T**, $\bar{\mathbf{T}}$), conjunto de todos os arcos (i, j) tal que $i \in$ **T** e $j \in \bar{\mathbf{T}}$. A árvore abrangente mínima **T*** vai sendo construída, adicionando em cada iteração o nó j e o respectivo arco (i, j) que possui o menor custo. O algoritmo termina quando todos os nós da rede pertencerem à árvore **T*** , quando não houver mais corte, como se ilustra na Figura A.3.

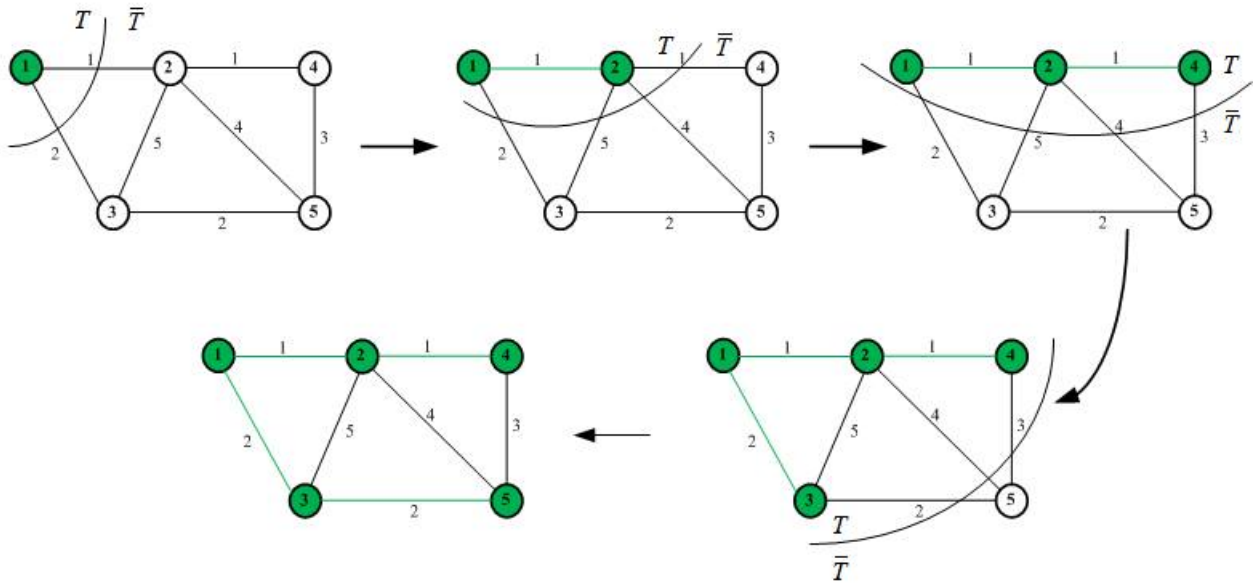


Figura A.3: Ilustração do Prim