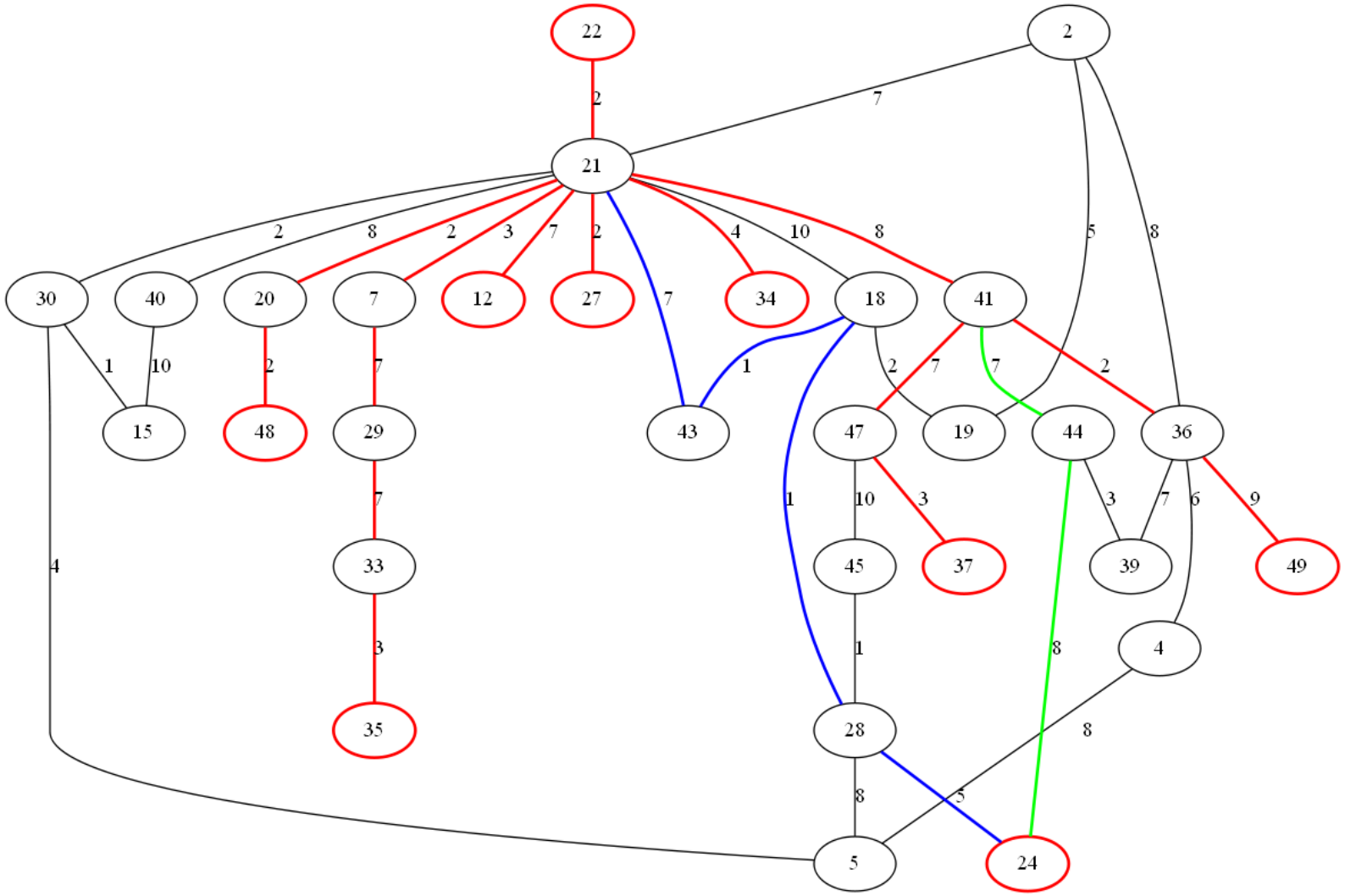

Rui Miguel Lopes Ribeiro Ferreira Barbosa

Encaminhamento de ligações multiponto

Dissertação de Mestrado em Engenharia Eletrotécnica e de Computadores,
especialização em Telecomunicações

Julho / 2014

UNIVERSIDADE DE COIMBRA

Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Departamento de Engenharia Eletrotécnica e de Computadores

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Encaminhamento de ligações multiponto

Rui Miguel Lopes Ribeiro Ferreira Barbosa

Júri:

José Manuel Fernandes Craveirinha (Presidente)

Lúcia Maria dos Reis Albuquerque Martins (Orientadora)

Tony Richard de Oliveira de Almeida (Vogal)

Coimbra, Julho de 2014

Agradecimentos

À minha família e amigos o apoio todo que me deram e a paciência que tiveram comigo ao longo dos últimos meses.

À Professora Teresa Gomes, por me ter sugerido o artigo de identificação de sub-grafos conexos num grafo desconexo, que foi muito importante ao longo de todo o trabalho.

À minha orientadora, Professora Doutora Lúcia Maria dos Reis Albuquerque Martins, pelo tempo todo que me dispensou e todos os conhecimentos que me transmitiu ao longo da execução desta tese. Agradeço também o apoio e a boa orientação.

Resumo

O objetivo deste trabalho foi o estudo do problema de Steiner em grafos e sua aplicação na determinação de ligações multiponto-multiponto em redes de Telecomunicações, em particular em redes de transporte. Sendo assim, estudou-se mais profundamente uma heurística de referência para o problema de Steiner e ainda uma meta-heurística que obtém muito bons resultados para este problema.

Em problemas reais, para além de se querer obter as árvores de menor custo (aditivo) para a interligação de um subconjunto de nós da rede envolvidos em cada ligação multiponto-multiponto, custo esse que pode representar, por exemplo, a ocupação de cada *link*, também se pode pretender obter as árvores com um número mínimo de arcos, por forma a que as ligações multiponto-multiponto possam ser implementadas com o mínimo de recursos da rede. Assim formula-se um problema de Steiner bi-critério para o qual se propõe uma extensão da meta-heurística anterior de modo a resolver esse problema.

Os resultados obtidos pela implementação da meta-heurística original (mono-critério) são comparados exaustivamente com os resultados obtidos pelos seus autores, o que permitiu tirar conclusões sobre a forma pouco clara como estes últimos resultados foram conseguidos, bem como identificar aspetos em que a meta-heurística poderia ser melhorada. Estas conclusões foram fundamentais para o desenvolvimento de uma meta-heurística, baseada em princípios idênticos à anteriormente referida, para a resolução do problema bi-critério.

No problema bi-critério, a meta-heurística desenvolvida revelou-se bastante superior a três outras heurísticas desenvolvidas anteriormente para a resolução do mesmo problema. Os resultados obtidos permitem ainda identificar aspetos que podem vir a ser melhorados em trabalhos futuros.

Palavras-chave: problema de Steiner, otimização multi-objetivo, *tabu search*, ligações multiponto-multiponto.

Abstract

The main purpose of this master thesis was the study of the Steiner problem in graphs for the obtaining of multipoint-to-multipoint virtual connections in communication transport networks. Therefore, a reference heuristic and also a meta-heuristic that obtains very good results for this problem were studied and implemented.

In real problems it may be advantageous not only the obtaining of the lowest-cost trees for interconnecting a given subset of network nodes involved in each multipoint-to-multipoint virtual connection, where the cost can represent, for instance, the occupancy of each link, but also the obtaining of trees with the minimum number of arcs, so that multipoint-to-multipoint virtual connections might be implemented with the minimum of network resources. So a bi-criteria Steiner tree problem was formulated and in order to solve it an extension of the previous meta-heuristic was proposed.

The results obtained by the implementation of the original meta-heuristic (single-criterion) are compared in detail with the results obtained by their authors. From this comparison, two main conclusions were achieved: i) some very good results presented by the authors of the meta-heuristic were obtained through procedures not clearly documented; ii) some aspects of the meta-heuristic can be improved. These conclusions were taken into account in the development of a meta-heuristic for the bi-criteria problem, based on identical principles.

The last meta-heuristic proved to be superior to three other previously developed heuristics for the same problem. The results also allowed the identification of some aspects that may be improved in future work.

Keywords: Steiner problem, multi-objective optimization, tabu search, multipoint-to-multipoint virtual connections.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e Abreviaturas

mp2mp *multiponto-multiponto*

STP *problema de Steiner em grafos*

BSTP *problema de Steiner bi-critério*

BSTP-Tabu *algoritmo que resolve o problema de Steiner bi-critério*

F-Tabu *Full Tabu*

Capítulo 1

Introdução

Nas redes de telecomunicações tem-se registado um grande aumento do número de serviços *multicast* para os quais é necessário o estabelecimento de ligações multiponto-multiponto. Surge assim a necessidade de se ligar apenas um subconjunto de nós numa rede através das interligações de custo total mínimo. Esse problema pode ser formulado como um problema de Steiner em grafos.

Uma rede pode ser representada através de um grafo em que todos os elementos da rede, sejam eles nós ou arcos, podem ter custos associados. Neste trabalho apenas os arcos vão ter custos (não negativos).

O problema de Steiner apareceu inicialmente como problema de Steiner Euclidiano, que consiste em determinar a ligação de comprimento total mínimo, de modo a interligar um conjunto de pontos num plano. A interligação final pode ligar diretamente os pontos pretendidos ou pode ligá-los recorrendo a pontos adicionais, sendo estes últimos designados por pontos de Steiner. Esse mesmo problema foi mais tarde estendido a grafos, surgindo assim o problema de Steiner em grafos.

O problema de Steiner em grafos consiste na determinação da ligação de menor custo entre um subconjunto de nós da rede, designado por conjunto de nós terminais, ou seja, consiste na obtenção de uma árvore de custo mínimo para a interligação desses nós. Essa árvore pode incluir ainda nós adicionais, designados por nós de Steiner. Este problema não tem nenhum algoritmo conhecido que o resolva de forma exata em tempo polinomial e, dada a sua complexidade, é considerado um problema NP-completo [8].

O custo associado a cada arco representa sempre uma grandeza que pode ser, por exemplo, a distância, como no caso do problema de Steiner Euclidiano, a ocupação do arco, o atraso, etc. A solução ótima é, então, aquela cujo custo total da árvore é mínimo. No caso do

problema de Steiner o custo total da árvore é aditivo, ou seja, é igual à soma dos custos associados aos arcos da árvore.

O problema de Steiner em grafos, como já foi dito, é NP-completo, o que significa que não existem algoritmos que resolvam instâncias desse problema de dimensão razoável de forma ótima em tempo útil. Isso leva a que existam, na literatura, imensos algoritmos que tentam resolver esse problema mas sem garantias de encontrar a solução ótima. Existem alguns algoritmos de referência, alguns propostos há algumas décadas atrás, tal como o algoritmo de Kou et al. [14] ou o de Takahashi e Matsuyama [10], que permitem resolver o problema com um erro relativamente baixo, i. e. com uma diferença, entre o custo da árvore obtida e a solução ótima, baixa. Mais recentemente surgiram algoritmos baseados em metaheurísticas que apresentam bons resultados, tal como os apresentados por Gendreau et al. [4] e Ribeiro e Souza [11] que são baseados em *tabu search*, sendo que continuam constantemente a aparecer na literatura algoritmos que garantem cada vez melhores resultados e/ou de forma mais rápida, tal como o apresentado por Krishna et al. [15] ou Vygen [13]. E existem muitos muitos outros, existindo até algoritmos híbridos, ou seja, algoritmos que usam mais do que um método de otimização para a resolução de um problema, como o apresentado por Noferesti e Shah-Hosseini [16].

Em muitos dos problemas reais existem múltiplos critérios envolvidos na escolha de soluções e não apenas um. Assim pode ser vantajoso formular um problema com mais do que uma função objetivo, explicitando formalmente a natureza multidimensional do problema. Neste caso não existe geralmente uma solução ótima que satisfaça todas as funções objetivo simultaneamente pois para se otimizar um objetivo, por norma, piora-se outro. A solução ótima é assim substituída por um conjunto de soluções não dominadas, do qual depois se pode escolher uma solução de compromisso. Entre quaisquer duas soluções não dominadas verifica-se uma melhoria num dos objetivos em detrimento de, pelo menos, um dos outros. Diz-se, então, que uma solução é dominada por outra quando existe uma melhoria da outra em pelo menos um objetivo permanecendo inalterados os restantes objetivos.

Se o problema de Steiner mono-critério é muito complexo, o problema de Steiner bi-critério ainda é mais. Para o problema bi-critério não existe nenhum algoritmo de referência, apesar do crescente interesse que tem surgido devido às suas aplicações práticas. Encontram-se na literatura alguns algoritmos como o apresentado por Pinto e Laporte [12] para um problema triobjetivo em que, para além do objetivo de minimização do número de arcos da raiz da árvore até cada um dos nós terminais e da minimização do comprimento máximo de cada arco, também existe o objetivo da maximização da receita, considerando que cada nó que

pertença à solução tem uma receita associada.

1.1 Objetivo do trabalho

O objetivo desta dissertação de mestrado foi o estudo e desenvolvimento de algoritmos para a resolução do problema de Steiner com o propósito de se poderem posteriormente adaptar à obtenção de ligações multiponto-multiponto (mp2mp) em redes de transporte. Para isso, o problema clássico de Steiner foi estendido para uma formulação bi-critério, com o propósito de se obterem não só as árvores de menor custo (aditivo) para a interligação de um subconjunto de nós da rede envolvidos em cada ligação mp2mp, mas também as árvores com um número mínimo de arcos, por forma a que as ligações mp2mp possam ser implementadas com o mínimo de recursos da rede. O custo aditivo associado a cada arco deverá ser, em principio, um custo que representa a ocupação do arco mas pode ser outra métrica qualquer desde que seja aditiva.

A abordagem escolhida para o desenvolvimento deste trabalho foi a apresentada por Gendreau et al. [4] por já ter existido contacto com ela previamente no grupo de investigação em que este projeto se insere e por ser uma das que apresenta melhores desempenhos.

Foram então estudados os algoritmos em que se baseia a meta-heurística referida e foi ainda implementada essa meta-heurística que recorre a *tabu search* para a resolução do problema de Steiner mono-critério. É ainda proposta uma extensão desta meta-heurística para a resolução do problema de Steiner bi-critério mencionado.

Na sequência de um problema proposto pela PT Inovação foi ainda estudado um caso particular do problema de Steiner (mono-critério) com restrições em que se pretende interligar um conjunto de nós terminais através de uma árvore de custo mínimo que deverá ser constituída obrigatoriamente por um conjunto de elementos da rede (nós e/ou arcos) pré-definidos. Este problema tem como sub-problema a verificação da coerência dos elementos de rede obrigatórios face à solução final pretendida. Sendo esta solução uma árvore, os elementos obrigatórios não poderão conter ciclos. Os resultados (provisórios) deste estudo apresentam-se no apêndice A.

1.2 Conteúdo da tese

A tese está assim estruturada em capítulos da seguinte forma:

- No capítulo 2 é apresentada a notação básica usada ao longo do trabalho e alguns

algoritmos que resolvem problemas mais simples de otimização em redes, que são mais tarde usados no âmbito das heurísticas e meta-heurísticas para a resolução do problema de Steiner mono e bi-critério.

- No capítulo 3 é abordado o problema de Steiner em grafos e são apresentadas mais detalhadamente uma heurística [10] e uma meta-heurística [4] que o permitem resolver no caso do mono-critério. É ainda apresentada uma generalização da meta-heurística anterior para a resolução do problema de Steiner bi-critério.

- No capítulo 4 são apresentados e analisados os resultados obtidos pelos diferentes algoritmos referidos no capítulo 3.

- No capítulo 5 são apresentadas as principais conclusões do trabalho realizado, e são referidos ainda os aspetos possíveis de melhorar em trabalho futuro, nomeadamente relacionados com o problema de Steiner bi-critério.

Capítulo 2

Algoritmos básicos

Neste capítulo é introduzida a notação relativa à representação da rede bem como algumas definições necessárias à compreensão dos algoritmos apresentados. De seguida, é apresentada a *heap*, uma estrutura de dados usada por alguns algoritmos para uma implementação mais eficiente. São ainda descritos dois algoritmos que recorrem a uma *heap*: o algoritmo de Dijkstra, que calcula o caminho mais curto entre qualquer par de nós da rede e o algoritmo de Prim, que calcula a árvore mínima abrangente de uma rede. No final do capítulo apresenta-se ainda um método que identifica sub-grafos conexos a partir de um grafo eventualmente desconexo.

2.1 Notação e Definições

O grafo $G = (N, A)$ consiste num conjunto N de nós, com $N = \{1, 2, 3, \ldots, |N|\}$, e num conjunto A de arcos, sendo $A = \{(i, j) : i, j \in N \wedge i \neq j\}$.

A cada arco está associado um valor designado genericamente por custo, que pode representar a ocupação do arco, o atraso, a capacidade disponível, etc. Por exemplo, entre o nó i e j, o custo é representado por $c(i, j)$, o qual também pode ser designado por comprimento do arco.

Um grafo pode ser dirigido ou não dirigido. Para ser dirigido, os pares de nós constituintes de cada arco são referidos de forma ordenada. Num grafo não dirigido pode-se referir o arco que liga o nó i a j tanto como (i,j) como (j,i).

Sempre que existir o arco $(i,j) \in A$, dizemos que j é adjacente ao nó i, no caso do grafo ser dirigido. Se não for, ambos os nós são adjacentes um do outro. Isso permite-nos criar listas de adjacências. A lista de adjacências $A(i)$ de um nó i é o conjunto de arcos ligados a esse nó, também denominados por arcos incidentes no nó i. Ou seja, $A(i) = \{(i, j) \in A : j \in N\}$.

O grau de um nó corresponde ao número de arcos incidentes nesse nó.

Neste trabalho apenas são considerados grafos não dirigidos.

O grafo $G' = (N', A')$ é um sub-grafo de $G = (N, A)$ se $N' \subseteq N$ e $A' \subseteq A$.

Um caminho p_{ij} num grafo G é um sub-grafo de G que consiste numa sequência de arcos e nós $i - (i, l) - l - (l, u) - \cdots - k - (k, j) - j$, sem repetição de nenhum nó.

Nesta dissertação de mestrado apenas se consideram custos aditivos, logo o custo total do p_{ij} é a soma do custo dos arcos desse caminho, o qual também poderá ser referido como a distância entre o nó i e j. Um caminho pode também ser representado apenas pela sequência de arcos. Geralmente existe mais do que um caminho entre o nó i e j, representando-se o caminho mais curto entre esses nós por p^*_{ij}. É de notar que o caminho mais curto pode não ser único.

Um ciclo é o caminho $i - (i, l) - \cdots - (u, k) - k - (k, i) - i$.

Diz-se que um grafo é conexo quando todos os seus nós estão ligados entre si, ou seja, quando existe um caminho a interligar 2 nós quaisquer do grafo. Uma árvore é um grafo conexo e sem ciclos.

Então, chama-se árvore a um grafo que tenha as seguintes propriedades:

- Uma árvore com n nós tem exatamente n-1 arcos.
- Uma árvore tem pelo menos 2 nós "folha", ou seja, dois nós que estão apenas ligados a um outro nó.
- Dois nós quaisquer na árvore estão ligados por apenas um único caminho.

Uma árvore abrangente de G, ou *spanning tree*, é uma árvore que contém todos os nós de G. Na figura 2.1 está assinalada a vermelho uma árvore abrangente definida sobre o grafo representado.

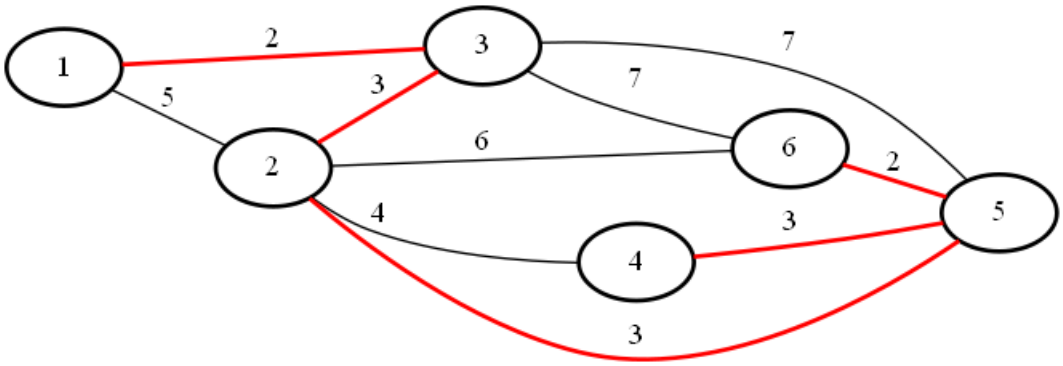

Figura 2.1: Árvore mínima abrangente de um grafo.

2.2 A *Heap*

A *heap* é uma estrutura de dados usada para guardar e manipular eficientemente um conjunto H, em que cada elemento $i \in H$ tem associado um número real, denominado por chave (ou *key*).

Numa *heap*, os elementos são guardados numa árvore com origem em apenas um nó designado por raiz da árvore, e os arcos da árvore representam relações pai-filho. Os elementos são inseridos na *heap* com profundidade crescente. Elementos com igual profundidade são inseridos pela ordem de chegada. Profundidade de um elemento é o número de arcos no caminho único entre esse elemento e a raiz da árvore.

Propriedade 1. *A chave de um nó i é menor ou igual à chave dos seus sucessores, ou seja, é sempre menor ou igual que a chave dos seus filhos tal como as chaves dos filhos dos seus filhos, etc.*

Isso leva-nos a outra propriedade importante:

Propriedade 2. *O elemento situado na raiz da heap tem a chave mínima.*

Numa *heap* podem-se efetuar as seguintes operações:

- create(H). Cria uma *heap* H vazia.
- insert(i, H). Insere um elemento i na *heap*. Normalmente cada elemento da *heap* é composto por vários campos sendo um deles o campo chave.
- find-min(i,H). Encontra o elemento i com a menor chave na *heap*.
- delete-min(i,H). Apaga o elemento i com a menor *chave* na *heap*.
- delete(i,H). Apaga o elemento i na *heap*.
- decrease-key(i,*valor*,H). Diminui a chave do elemento i para *valor*.
- increase-key(i,*valor*,H). Aumenta a chave do elemento i para *valor*.

As funções insert(i, H), delete-min(i,H), delete(i,H), decrease-key(i,*valor*,H) e increase-key(i,*valor*,H) têm em atenção as propriedades definidas anteriormente. Isso significa que, por exemplo, no decrease-key(i,*valor*,H), não só se diminui a chave do elemento i substituindo-o por *valor* como depois se verifica se essa diminuição da chave viola a propriedade 1. Caso isso se confirme, a própria função trata de fazer as alterações nas posições dos nós de modo a manter a *heap* ordenada, ou seja, de modo a manter as propriedades referidas anteriormente.

A *heap* implementada é binária, ou seja, cada nó tem no máximo dois nós filhos.

2.3 Algoritmo de Dijkstra

O algoritmo de Dijkstra determina os caminhos mais curtos entre um dado nó e todos os outros nós da rede, com custos não negativos associados aos arcos.

O problema de obtenção dos caminhos mais curtos é fulcral do ponto de vista da gestão de uma rede de telecomunicações, uma vez que os caminhos mais curtos precisam de ser calculados sempre que se quer enviar algum tipo de informação entre dois pontos numa rede da forma mais rápida, eficiente ou mais barata possível. Este tipo de problema ainda surge muitas vezes como sub-problema de outros problemas mais complexos de otimização em redes de telecomunicações.

Ao longo da execução do algoritmo de Dijkstra este atribui valores, ou etiquetas, a cada nó. Essas etiquetas podem ser temporárias ou permanentes. Uma etiqueta temporária representa a distância do nó origem a um outro nó encontrado até um dado momento da execução do algoritmo, enquanto que uma etiqueta permanente representa a distância mais curta do nó origem a esse nó. A etiqueta temporária também representa um limite superior da distância mais curta.

Basicamente, o algoritmo parte do nó s e vai etiquetando permanentemente cada um dos outros nós pela ordem da sua distância a s. Começa por atribuir a etiqueta permanente 0 ao nó s, ou seja, como se pode ver pelo algoritmo 1, $d(s) = 0$, e etiqueta temporária ∞ a todos os outros nós. Em cada iteração, a etiqueta de cada nó representa a distância mínima do nó origem a esse nó passando apenas por nós já com etiquetas permanentes. Passa a permanente a etiqueta temporária com menor valor, sendo depois etiquetados os seus nós vizinhos ainda com etiquetas temporárias de modo a poder atualizar os valores das etiquetas para valores inferiores, caso seja possível. O algoritmo termina quando todos os nós tiverem etiquetas permanentes.

Cada elemento e_i da árvore de caminhos mais curtos, T_D, obtida com o algoritmo de Dijkstra, entre um nó origem s e todos os outros nós da rede, é composto pelos campos nó, nó predecessor e custo. O custo representa a distância de cada nó ao seu predecessor e é o campo chave dos elementos na *heap*. Assim, o 1º elemento é $e_s = (s, pred(s), d(s)) = (s, 0, 0)$.

No algoritmo 1 apresenta-se formalmente o algoritmo de Dijkstra que calcula o caminho do nó origem s para todos os outros nós da rede. Caso o objetivo seja só calcular os caminhos para um certo subconjunto de nós, ou apenas para um outro nó, é necessário incluir uma condição de paragem que termina o algoritmo depois desses caminhos terem sido obtidos.

A representação $Dijkstra(s, j)$, refere-se à aplicação do algoritmo de Dijkstra entre o nó

origem s e o nó j e $Dijkstra(s, J)$ à construção da árvore de caminhos mais curtos entre o nó origem s e cada um dos nós do conjunto J tal que $J \subseteq N$. Desta árvore podem ser extraídos os caminhos p_{sj} com origem em s e destino $j \in J$.

```
d(s) = 0 e pred(s) = 0;
d(j) = ∞, ∀j ∈ N \ {s};
create(H);
insert(e_s, H);
T_D = ∅;
while H ≠ ∅ do
    find − min(e_i, H);
    delete − min(e_i, H);
    T_D = T_D ∪ {e_i};
    for each(i, j) ∈ A(i) do
        value = d(i) + c(i, j);
        if value < d(j) then
            if d(j) = ∞ then
                d(j) = value;
                pred(j) = i;
                insert(e_j, H);
            else if e_j ∈ H then
                d(j) = value;
                pred(j) = i;
                decrease − key(e_j, value, H);
            end
        end
    end
end
```

Algoritmo 1: Algoritmo de Dijkstra.

2.4 Algoritmo de Prim

O algoritmo de Prim calcula a árvore mínima abrangente, ou seja, descobre os arcos da rede necessários e suficientes para unir todos os nós, sem formação de ciclos, de modo a que a soma desses arcos seja mínima. No caso do grafo da figura 2.1, este algoritmo encontra a árvore abrangente que está representada a vermelho nesta figura, dado que não é possível encontrar neste grafo outra árvore abrangente com custo inferior.

O algoritmo de Prim parte de um nó e vai adicionando arcos um a um. Equivale a ter 2 conjuntos de nós: ligados e não ligados. Junta-se ao conjunto dos nós ligados o nó não ligado mais próximo de um dos outros já ligados e repete esse processo até todos os nós estarem ligados.

Voltam-se a usar as etiquetas temporárias e permanentes que foram referidas no algoritmo de Dijkstra. Começa-se por atribuir a etiqueta permanente com valor 0 ao nó por onde vamos começar e a etiqueta temporária com o valor ∞ a todos os outros nós. Em cada iteração do algoritmo passa a permanente a etiqueta temporária com o menor valor e atualizam-se os valores das etiquetas temporárias através da comparação do valor dessas etiquetas com as distâncias dos nós não ligados ao novo nó ligado. Acaba quando todas as etiquetas forem permanentes.

Os elementos da árvore construída pelo algoritmo de Prim, T_P, têm o mesmo formato que os elementos da árvore construída no algoritmo de Dijkstra.

No algoritmo 2 apresenta-se formalmente o algoritmo de Prim.

$d(s) = 0$ e $pred(s) = 0$;
$d(j) = \infty, \forall j \in N \setminus \{s\}$;
$create(H)$;
$T_P = \emptyset$;
for $j \in N \setminus \{s\}$ **do**
  $insert(e_j, H)$;
end
while $H \neq \emptyset$ **do**
  $find - min(e_i, H)$;
  $delete - min(e_i, H)$;
  $T_P = T_P \cup e_i$;
  for $each(i, j) \in A(i)$ com $e_j \in H$ **do**
    $value = c(i, j)$;
    if $value < d(j)$ **then**
      $d(j) = value$;
      $pred(j) = i$;
      $decrease - key(e_j, value, H)$;
    end
  end
end

Algoritmo 2: Algoritmo de Prim.

2.5 Identificação de grafos distintos

Em [3] é apresentado um algoritmo que permite identificar sub-grafos conexos distintos a partir de um conjunto de nós e de arcos, constituintes de um grafo que pode ser desconexo. Esse algoritmo recorre a uma pilha onde vai guardando os nós de cada sub-grafo.

O algoritmo vai verificando se existe algum arco que ligue o nó do topo da pilha a outro nó. Caso exista, o arco é adicionado ao grafo que está a ser identificado nesse momento, e

esse novo nó é acrescentado ao topo da pilha, se não tiver sido selecionado anteriormente pelo algoritmo. Caso esse arco não exista, elimina o nó do topo da pilha. Quando a pilha estiver vazia vê se ainda existe algum nó que ainda não tenha sido selecionado. Se não existir, o algoritmo acaba mas, se existir, recomeça a partir desse nó, sabendo que agora está a identificar um grafo distinto dos grafos já descobertos anteriormente.

Os detalhes do algoritmo estão representados no fluxograma da figura 2.2, onde 'nó desconhecido' significa nó que nunca foi selecionado pelo algoritmo.

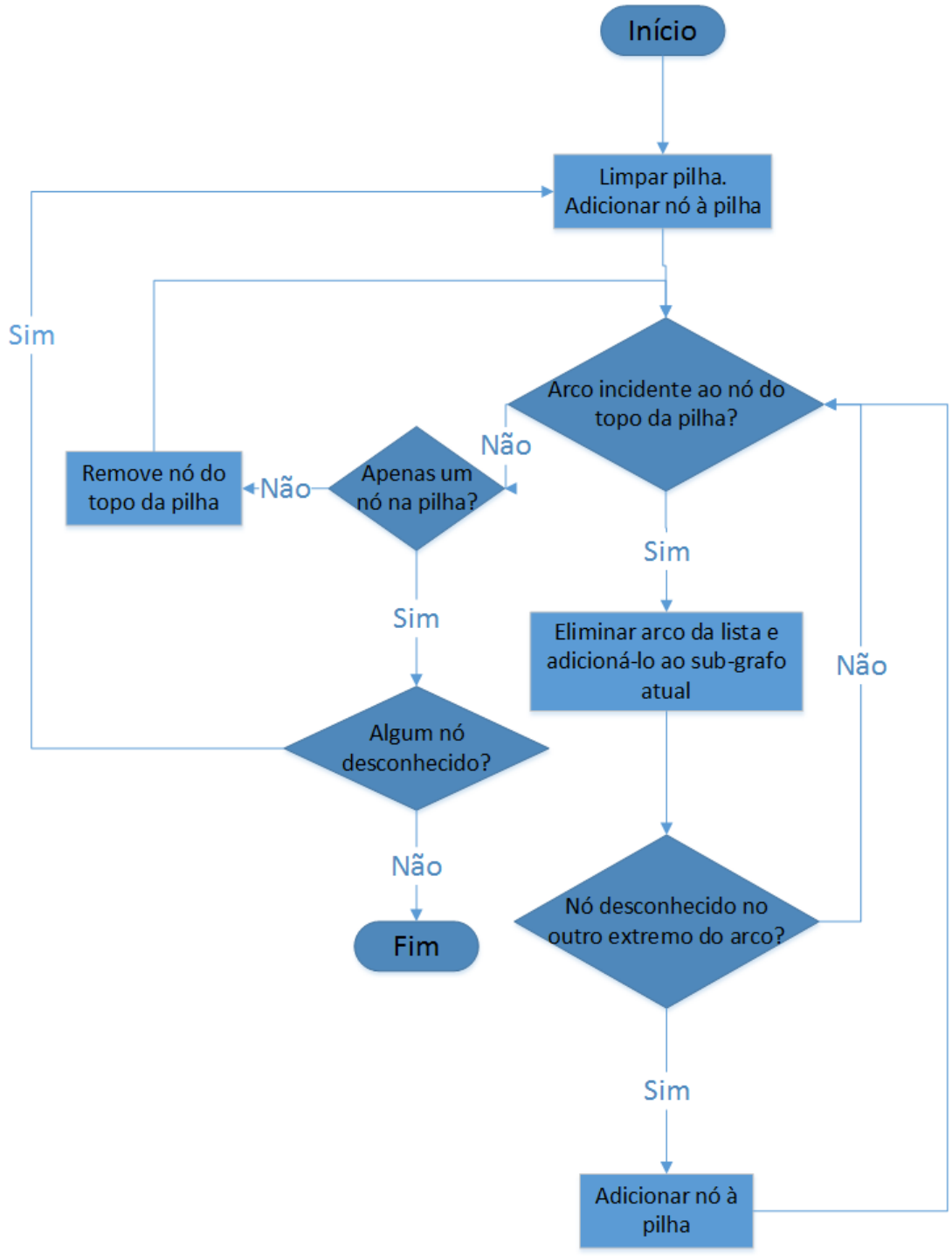

Figura 2.2: Fluxograma adaptado do apresentado em [3].

Capítulo 3

Problema de Steiner em grafos

O problema de Steiner em grafos ou *Steiner Tree Problem* (STP) refere-se à determinação de um sub-grafo conexo $G' \subset G$ composto obrigatoriamente por um conjunto pré-definido de nós pertencentes a G, designados por nós terminais, de tal forma a que o somatório dos custos dos arcos do sub-grafo seja o menor possível. Prova-se que o sub-grafo G' é uma árvore. Para além dos nós terminais, a árvore resultante contém também outros nós que são designados por nós de Steiner. O problema pode então apresentar-se da seguinte forma:

Sendo $\mathcal{T}$ o conjunto de todas as árvores T possíveis que interligam os nós terminais S e sendo $c(T)$ o custo total da árvore, o problema de Steiner pode ser formulado como

$$\min_{T \in \mathcal{T}} c(T) = \sum_{(i,j) \in T} c(i,j) \tag{3.1}$$

Existem 2 casos especiais cuja solução ótima é facilmente calculável. Se $|N'| = 2$ o problema transforma-se num problema de caminhos mais curtos cuja solução é dada, por exemplo, pelo algoritmo de Dijkstra e se $|N'| = |N|$ o problema transforma-se na determinação da árvore mínima abrangente que é resolvido, por exemplo, pelo algoritmo de Prim. Para os outros casos existem diversas heurísticas e meta-heurísticas que calculam soluções aproximadas da ótima podendo, em alguns casos, encontrar a solução ótima.

Neste capítulo começa-se por apresentar as reduções possíveis dos grafos no contexto do problema de Steiner. De seguida, é apresentada uma heurística que calcula as soluções iniciais de uma meta-heurística que resolve o problema de Steiner. No final do capítulo essa meta-heurística é estendida ao problema de Steiner bi-critério ou *Bicriteria Steiner Tree Problem* (BSTP).

3.1 Reduções

Existem diversos métodos que permitem reduzir o tamanho do grafo original tendo em vista o problema de Steiner. De seguida referem-se os métodos, propostos em [5], que vão ser aplicados no âmbito deste trabalho. Para isso, considere-se os nós i e j do grafo.

(a) Se i tem grau 1 e existe o arco (i, j), então tanto o arco como o nó i podem ser retirados do grafo. Se i for nó terminal então o nó j passa a nó terminal (no caso de não o ser) e tanto o nó i como o arco (i, j) são adicionados à solução.

(b) Sendo i nó de Steiner de grau 2 e existindo os arcos (j, i) e (i, l), então i, (j, i) e (i, l) são removidos do grafo e o arco (j, l) é adicionado ao grafo, se não existir este arco, com um custo associado de $c(j, l) = c(j, i) + c(i, l)$. Se já existir o arco (j, l), o seu custo será $c(j, l) = \min\{c(j, l), c(j, i) + c(i, l)\}$.

(c) Se $c(i, j) > c(p^*_{ij})$ elimina-se o arco (i, j), sendo p^*_{ij} o caminho mais curto entre o nó i e o nó j e $c(p^*_{ij})$ o custo desse caminho.

(d) Seja i um nó terminal e sejam j e l os 2 nós vizinhos mais próximos de i, respetivamente, e z o nó terminal mais próximo de i. Se $c(i, j) + c(p^*_{jz}) \leq c(i, l)$, então o arco (i, j) faz parte da solução e o grafo pode ser contraído ao longo deste arco. Ou seja, o nó j é removido, os arcos incidentes em j ficarão incidentes em i e caso i fique com um par de arcos paralelos incidentes, o arco com maior custo é removido.

    É importante realçar que nesta redução, caso j e l estejam à mesma distância de i, a redução só é feita se j e l não forem ambos nós terminais. Se só um deles for nó terminal, então considera-se esse nó como sendo o vizinho mais próximo.

Para a implementação destas reduções é necessário conhecer-se a matriz $D(G)$, de $|N|\times|N|$, dos caminhos mais curtos entre cada par de nós do grafo, sendo esta atualizada ao longo do processo de redução do grafo sem ter que se recorrer ao algoritmo de Dijkstra. Isso significa apagar uma linha e uma coluna da matriz $D(G)$ aquando de uma redução (a) ou (b). A linha e a coluna apagadas são as correspondentes ao nó i. Numa redução (d) a coluna e a linha correspondentes ao nó i ficam com os seguintes valores (sendo depois apagada também a linha e a coluna correspondente ao nó j):

1. Inicialização: $N_1 = \emptyset$ e $N_2 = \emptyset$

2. Para cada $m \in N$, se $c(p^*_{mj}) > c(p^*_{mi})$

$$N_1 \leftarrow N_1 \cup \{m\},\ c(p^*_{mj}) = c(p^*_{mi})\ e\ c(p^*_{jm}) = c(p^*_{mj})$$

3. Para cada $l \in N$, se $c(p^*_{il}) > c(p^*_{jl})$

$$N_2 \leftarrow N_2 \cup \{l\},\ c(p^*_{il}) = c(p^*_{jl})\ e\ c(p^*_{li}) = c(p^*_{il})$$

4. Para cada $(m, l) \in N_1 \times N_2$, se $c(p^*_{ml}) > c(p^*_{mi}) + c(p^*_{il})$

$$c(p^*_{ml}) = c(p^*_{mi}) + c(p^*_{il})\ e\ c(p^*_{lm}) = c(p^*_{ml})$$

Não havendo nenhuma prova da melhor maneira de aplicar estas reduções, elas serão realizadas pela seguinte ordem: c-b-d-a, tal como proposto em [5]. É de notar que não se realiza apenas 1 ciclo de reduções mas sim vários até deixar de se conseguir reduzir mais o grafo.

3.2 Algoritmo de Takahashi

O algoritmo de Takahashi, apresentado por Takahashi e Matsuyama em [10], permite obter uma solução aproximada para o problema de Steiner em grafos, ou seja, permite obter uma solução próxima da ótima (ou mesmo a ótima) quando se pretende ligar um conjunto de nós terminais S, com $S \subset N$, de tal forma que o custo total da árvore seja mínimo.

O algoritmo parte apenas de um dos nós do conjunto e depois adiciona os outros um a um em cada iteração. Pode-se voltar a usar o método da etiquetagem temporária e permanente na escolha do nó que se vai ligar ao sub-grafo que está a ser construído, à semelhança do algoritmo de Prim. Cada nó do subconjunto de nós que se quer ligar tem uma etiqueta temporária cujo valor corresponde ao custo do caminho obtido pelo algoritmo de Dijkstra entre esse nó e um dos nós pertencentes à sub-árvore da solução já encontrada até ao momento. A etiqueta que passa a permanente em cada iteração é a etiqueta temporária com menor valor. Quando passa a permanente, adicionam-se ao sub-grafo todos os nós do grafo original do caminho com menor custo.

A árvore resultante do algoritmo de Takahashi, T_T, é composta pelos diferentes caminhos para a interligação de cada um dos nós terminais.

No algoritmo 3 apresenta-se formalmente o algoritmo de Takahashi.

Cada elemento da *heap* é constituído pelos seguintes campos: nó origem, nó destino, custo do caminho, que neste caso é o campo chave, e o caminho propriamente dito obtido através do algoritmo de Dijkstra de i para cada um dos elementos $j \in J$. Seja $P_{i,J}$ o conjunto desses caminhos: $P_{i,J} = \{p^*_{ij} : j \in J \wedge p^*_{ij} \in Dijkstra(i, J)\}$.

Seja ainda $N_{ab} \subset N$, o conjunto de nós pertencentes ao caminho p_{ab}, excetuando o nó a.

$J = S - \{s\}$;
$T_T \leftarrow (s, 0, 0)$;
$P_{s,J} \leftarrow Dijkstra(s, J)$;
$create(H)$;
for $j \in J$ **do**
  $insert(p_{sj}, H)$;
end
while $J \neq \emptyset$ **do**
  $find - min(p_{ab}, H)$;
  $delete - min(p_{ab}, H)$;
  if $b \in J$ **then** `// verifica se o caminho retirado da heap liga a nós terminais que ainda faltam ligar`
    $J = J - b$;
    for *each* $n \in N_{ab}$ **do**
      $P_{n,J} = Dijkstra(n, J)$;
      for $j \in J$ **do**
        $insert(p_{nj}, H)$;
      end
    end
    $T_T \leftarrow T_T \cup p_{ab}$;
  end
end

Algoritmo 3: Algoritmo de Takahashi.

Existem duas formas de melhorar os resultados do algoritmo de Takahashi, que podem ser usadas. Uma delas é mudando o nó terminal inicial. A árvore resultante pode depender do nó inicial e, portanto, o ideal é experimentar cada um dos nós terminais como nó inicial e escolher a melhor solução, tal como foi mostrado por Winter e MacGregor Smith [1]. A outra forma de melhorar o resultado dado pelo algoritmo de Takahashi foi proposta por Rayward-Smith e Clare [2]. Neste caso aplica-se um algoritmo de determinação da árvore mínima abrangente (por exemplo o algoritmo de Prim) ao sub-grafo induzido composto pela solução obtida pelo algoritmo de Takahashi e pelos arcos que interligam esses nós no grafo original, mas que não pertencem à solução. Depois verifica-se se existem, na árvore abrangente, nós folha (de grau 1) que não sejam nós terminais. Se existirem eliminam-se tal

como ao arco que os liga ao resto da solução.

Na figura 3.2 está representado uma rede com os nós terminais e a árvore de Steiner ótima assinalados a vermelho. Esta solução ótima nunca poderia ser encontrada pelo algoritmo de Takahashi, o que ilustra a complexidade do problema de Steiner.

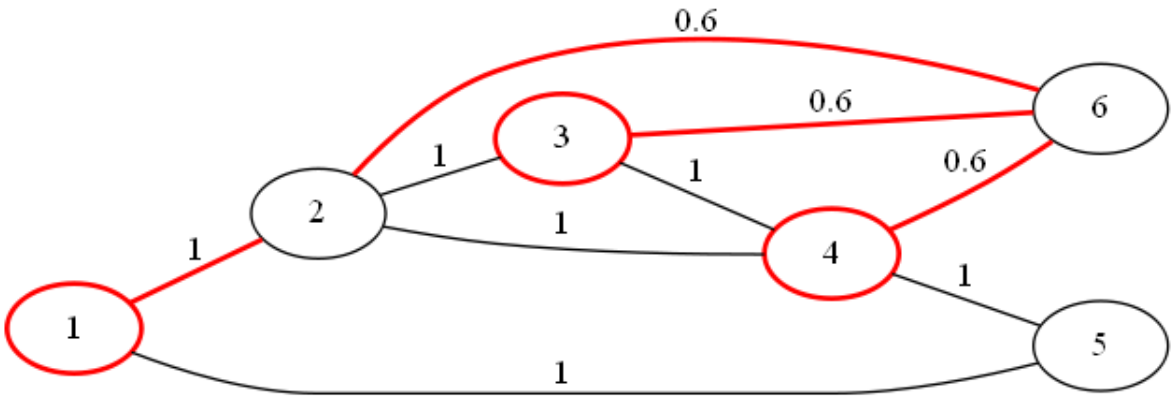

Figura 3.1: Árvore de Steiner ótima.

A heurística de Takahashi adapta-se muito facilmente à resolução de um problema de Steiner com restrições sugerido pela PT Inovação. Neste problema pretende-se que, para além dos nós terminais, um conjunto de nós e/ou arcos pertençam à árvore final. No apêndice A descrevem-se as adaptações feitas ao algoritmo de Takahashi para a resolução deste problema.

3.3 A utilização de uma meta-heurística para a resolução do STP

Em [4], foi proposta uma abordagem para resolver o problema de Steiner em grafos recorrendo a *tabu search*. *Tabu search* é o nome dado a uma meta-heurística que iterativamente procura a melhor solução na vizinhança da solução atual. No processo de pesquisa de melhores soluções recorre-se a operações que permitem encontrar soluções piores do que as anteriormente obtidas, evitando-se assim cair em ótimos locais. Para evitar o regresso a uma solução anterior existem operações proibidas por um determinado número de ciclos. Essas operações proibidas são chamadas de 'tabu'. Existem também métodos de diversificação que permitem a pesquisa de soluções em regiões ainda por explorar no espaço de soluções.

Neste caso do problema de Steiner o método de pesquisa de soluções na vizinhança de uma dada solução é baseada em 2 operações básicas: adicionar e remover um nó. De seguida descrevem-se essas operações que depois são usadas no procedimento básico de *tabu search*.

3.3.1 Operações Fundamentais

Para adicionar um novo nó à árvore tem de se considerar todos os possíveis arcos incidentes na árvore. Por isso, sabemos que depois do primeiro arco ser inserido, todos os outros arcos, se existirem no grafo, criarão ciclos ao serem adicionados à solução. Cada um desses ciclos pode ser eliminado removendo o arco de maior custo. Neste trabalho o procedimento para adicionar um novo nó à árvore recorre ao algoritmo de Prim e é constituído pelos seguintes passos:

1. Adiciona o nó à árvore através do primeiro arco encontrado incidente no nó e num dos nós da árvore;

2. Se ainda existe mais algum arco incidente nesse nó e em mais algum nó da árvore, adicionam-se esses outros arcos e no final aplica-se o algoritmo de Prim ao grafo resultante.

Quando se pretende eliminar um nó de uma árvore, eliminam-se também todos os arcos na árvore incidentes a esse nó. Isso leva à criação de n sub-grafos que precisam de ser conectados através de $n-1$ arcos. Para isso é necessário identificar os nós de cada um dos sub-grafos e depois encontrar, através de métodos semelhantes aos usados, por exemplo, pelo algoritmo de Prim, a combinação de arcos com menor custo para ligar os sub-grafos. O procedimento para remover um nó da árvore é constituído pelos seguintes passos:

1. Remove o nó da árvore tal como todos os arcos incidentes no nó que interligam com outros nós da árvore;

2. Identificam-se todos os nós de cada sub-grafo resultante da remoção de um nó pelo algoritmo proposto em [3];

3. Aplica-se o algoritmo 4 para a união dos sub-grafos anteriores, onde 'grafos' representa o conjunto de sub-grafos identificados no passo anterior.

```
Entrada: grafos // arraylist com todos os sub-grafos que se quer unir;
n = 1;
T = grafo_1;
grafos ← grafos \ {grafo_1};
while n < numero_de_sub_grafos do
    C = ∞;
    for each (i, j) com i ∈ T e ∀j ∈ grafos do
        if c(i, j) < C then
            C = c(i, j);
            w = j;
            arco = (i, j);
        end
    end
    t = 1;
    while t ≤ numero_de_sub_grafos − n do
        if j ∈ grafo_t then
            T ← T ∪ grafo_t ∪ arco;
            grafos ← grafos \ {grafo_t};
            break;
        end
        t = t + 1;
    end
    n = n + 1;
end
```

Algoritmo 4: União dos grafos.

É de referir que as operações de adicionar ou remover um nó da árvore não seguiram os procedimentos propostos em [4]. Por já terem sido desenvolvidos algoritmos que poderiam ser facilmente adaptados a estas operações, o seu uso permitiu acelerar a obtenção de resultados, embora de forma eventualmente menos eficiente quando comparados com os propostos em [4]. O algoritmo de identificação de sub-grafos distintos está descrito na secção 2.5 e o algoritmo 4 foi desenvolvido com base nesse. A eficiência destes algoritmos precisa ainda de ser avaliada.

3.3.2 Técnicas de Diversificação

As técnicas de diversificação permitem melhorar os resultados do *tabu search* por conduzirem a pesquisas de novas soluções em áreas não exploradas no espaço de soluções. Aqui são consideradas duas técnicas de diversificação: diversificação contínua e diversificação de caminhos.

Diversificação contínua

A diversificação contínua é baseada na memória a médio e longo prazo. Basicamente consiste em remover nós que estão há muito tempo na solução e favorecer a introdução de nós que ainda não tenham sido escolhidos para a solução. Isto é feito introduzindo uma penalização,

que tanto pode ser positiva como negativa, que será adicionada ao custo total da árvore resultante da adição ou da remoção de um nó. Uma função logarítmica da penalização foi proposta em [7].

Sendo n o número de vezes que um nó esteve presente na solução e m o número de iterações seguidas desde que o nó está presente na solução, a função que calcula o valor da penalização do custo da árvore resultante da adição ou remoção do nó j é dada pelo algoritmo 5.

if *j vai ser adicionado à solução* **then**
  $penalização(j) = log(n + 1)$;
else if *j vai ser removido da solução* **then**
  $penalização(j) = -log(m + 1)$;
end

Algoritmo 5: Penalização do custo da árvore.

Diversificação de caminhos

A diversificação interrompe a procura de uma solução melhor numa determinada região de pesquisa após um certo número de iterações sem se conseguir melhorar a solução. Quando é ativada faz com que se volte à melhor solução encontrada até ao momento e se substitua o caminho entre dois nós (i, j) presentes na solução. Os nós i e j são aqueles, de todos os nós presentes na solução, que têm a maior diferença entre o custo do caminho entre eles na solução e o custo do caminho mais curto no grafo original. Depois de descoberto o par de nós substitui-se o caminho da solução pelo caminho mais curto entre eles no grafo original. Para além disso, é preciso descobrir que outros nós terminais ficaram separados do resto da solução com a substituição do caminho. Esses nós voltam a ser ligados à árvore tal como no algoritmo de Takahashi.

A diversificação de caminhos é ativada após um certo número de iterações sem se conseguir melhorar a melhor solução obtida após a última diversificação de caminhos. Esse número de iterações foi ajustado, no âmbito desta dissertação, tendo-se optado, depois de vários testes, por ser igual a metade do número de nós presentes no grafo. Assim esta diversificação é chamada um número de vezes suficiente para permitir melhores resultados sem que esse número seja excessivamente elevado, o que poderia ter um impacto negativo nos resultados e conduzir a um grande esforço computacional.

3.3.3 Procedimento básico de *tabu search*

O procedimento de *tabu search* completo é denominado por Full Tabu (F-Tabu) cujo núcleo é o P-Tabu, que é descrito de seguida. O procedimento básico de *tabu search* é constituído pela obtenção das soluções iniciais e pelo P-Tabu.

As soluções iniciais para o problema são obtidas através do algoritmo de Takahashi com os procedimentos adicionais apresentados antes, que consistem em usar cada um dos nós terminais como nó inicial no algoritmo de Takahashi e usar o algoritmo de Prim no grafo induzido pela árvore resultante da aplicação do algoritmo de Takahashi no grafo original, eliminando os nós folha que não sejam nós terminais. Limita-se a 100 o número de soluções obtidas desta forma.

A cada uma das soluções (T), aplica-se as operações descritas anteriormente de adição ou remoção de nó a cada um dos nós de Steiner do grafo, consoante estejam ou não na solução. Assim determina-se uma nova solução existente na vizinhança da solução anterior e guarda-se a melhor encontrada até ao momento (T^*). No P-Tabu considera-se a diversificação contínua e as movimentações tabu. Considera-se que uma movimentação é tabu se a a operação de adição ou remoção de um nó estiver proibida. O número de iterações de movimentações tabu para cada nó pode ser consultado na tabela 3.1. O P-Tabu é constituído pelos procedimentos que constam no pseudo-código apresentado no algoritmo 6, onde $c(T)$ é o custo total da árvore T, como já foi referido.

```
iteração = 0;
T* ← melhor solução encontrada;
while iteração < 20 do
    P = ∞;
    for cada nó de Steiner j do
        T' ← adicionar_ou_remover_nó(j, T);
        if ((c(T') + penalização(j) < P) e (Tabu(j) ≤ iteração ou c(T') < c(T*)))
        then
            P = c(T');
            w = j;
        end
    end
    T = adicionar_ou_remover_nó(w, T);
    Tabu(w) = iteração + Random(x, y) + 1;
    if c(T) < c(T*) then
        T* ← T;
        iteração = −1;
    end
    iteração = iteração + 1;
end
```

Algoritmo 6: Procedimento P-Tabu.

É importante explicar alguns dos métodos do algoritmo 6. Para começar, o método $adicionar\_ou\_remover\_nó(j, T)$ consiste na construção de uma árvore adicionando ou removendo o nó de Steiner j da árvore T, consoante pertença ou não a essa árvore. O método $adicionar\_ou\_remover\_nó(w, T)$ atualiza a solução atual para a melhor solução na vizinhança, ou seja, adiciona ou remove o nó de Steiner w, não contando com a penalização imposta pela diversificação contínua. Esse nó é selecionado após se removerem todos os nós de Steiner, presentes na árvore atual, e adicionarem todos os nós de Steiner que não pertencem à árvore atual, um de cada vez. Assim é escolhida a melhor solução na vizinhança e atualiza-se a solução atual para essa. Não esquecer que essa melhor solução na vizinhança não pode ser obtida numa iteração tabu a não ser que o custo total da árvore seja o mínimo encontrado até ao momento. $Tabu(w)$ corresponde à iteração a partir da qual se pode remover ou adicionar o nó w da solução. O valor dessa iteração é calculado através da função $Random(x, y)$ que devolve um valor aleatório entre x e y. Esse intervalo de valores aleatórios está representado na tabela 3.1, sendo que x e y têm valor constante ao longo do algoritmo pois dependem apenas do tamanho da rede.

Número de nós	(x, y)
$\leq$30	4-10
$\leq$80	4-14
$\leq$150	8-18
$\leq$350	12-27
>350	20-40

Tabela 3.1: Definição de (x, y) em $Random(x, y)$ em função do nº de nós da rede.

É de notar que nem sempre é possível adicionar ou remover um nó de Steiner. Para que se possa adicionar um novo nó de Steiner à solução é preciso que haja um arco desse nó para a solução atual. O mesmo sucede quando se tenta remover um nó de Steiner da solução. Isso só é possível quando existem arcos que possam interligar os sub-grafos que resultaram da remoção desse nó.

3.3.4 Full Tabu

O Full Tabu, ou F-Tabu, é o procedimento completo de *tabu search*, como já foi referido. Começa por uma procura intensiva da solução ótima através do procedimento básico de *tabu search*. De seguida é aplicado o P-Tabu à melhor solução encontrada até ao momento usando todos os mecanismos de diversificação, i. e., usando a diversificação contínua e a diversificação de caminhos, com o número máximo de iterações igual ao dobro do número de nós do grafo.

O F-Tabu é então constituído pelos seguintes procedimentos:

1. Procedimento básico de *tabu search*, constituído pela obtenção das soluções inicias e pelo P-Tabu com 20 iterações, tal como está definido no algoritmo 6;

2. Procedimento P-Tabu, usando como solução inicial a melhor solução encontrada pelo procedimento básico, com um número de iterações igual ao dobro do número de nós do grafo, com todos os métodos de diversificação.

3.4 Problema de Steiner bi-critério

Até agora considerou-se o problema clássico da determinação de árvores de Steiner em grafos. O único objetivo por detrás deste problema é o de encontrar a árvore de custo (aditivo) mínimo que interliga um conjunto S de nós terminais. No entanto, em redes de telecomunicações, pode haver interesse em encontrar, não apenas a árvore de custo mínimo, mas também a árvore com menos arcos, ou seja, que ocupa o menor número de elementos da rede. O interesse nesse problema foi introduzido na sequência de um projeto com PT Inovação.

Neste caso o problema que propomos resolver é o problema de Steiner em grafos com 2 custos associados a cada arco. Adiciona-se ao problema de minimização anterior o problema da minimização do número de arcos da solução. Isso equivale a associar a cada arco um segundo custo com valor unitário. Assim, o problema deixa de ser mono-critério e torna-se num problema bi-critério em que há duas funções objetivo a ter em conta.

3.4.1 Formulação do Problema

Existindo então 2 custos não negativos, $c^1(i,j)$ e $c^2(i,j)$, associados a cada arco, o problema de Steiner bi-critério pode ser formulado da seguinte forma, considerando $\mathcal{T}$ o conjunto de todas as árvores T possíveis que interligam os nós terminais S:

$$\min_{T \in \mathcal{T}} c^1(T) = \sum_{(i,j) \in T} c^1(i,j) \tag{3.2}$$

$$\min_{T \in \mathcal{T}} c^2(T) = \sum_{(i,j) \in T} c^2(i,j) \tag{3.3}$$

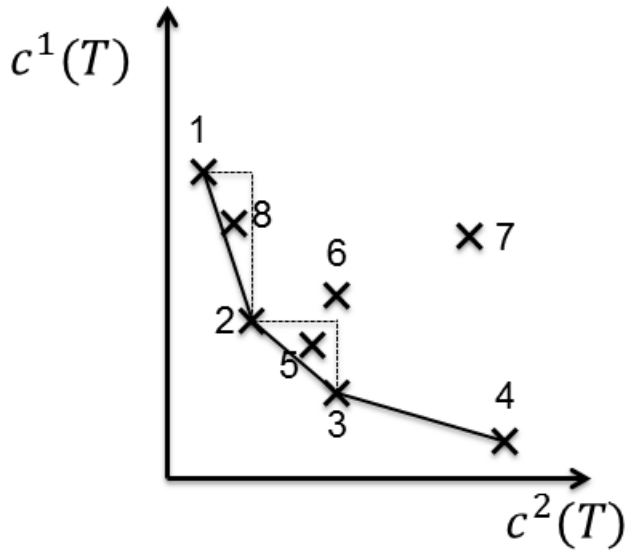

Figura 3.2: Soluções dominadas $(6, 7)$, soluções não dominadas suportadas $(1, \ldots, 4)$ e soluções não dominadas não suportadas $(5, 8)$.

Num problema multi-objetivo não existe, geralmente, uma solução ótima. Normalmente para se melhorar uma das funções objetivo piora-se a outra. Então, neste problema, não existe uma solução ótima mas sim um conjunto de soluções não dominadas.

Uma árvore T^1 domina uma árvore T^2 se e só se $c^1(T^1) \leq c^1(T^2)$ e $c^2(T^1) \leq c^2(T^2)$, sendo que pelo menos uma das desigualdades tem de ser estrita.

3.4.2 Soluções iniciais

Uma abordagem para a resolução de problemas multiobjetivo consiste na redução desse problema num bem mais simples, com uma única função objetivo, que resulta da combinação linear convexa das várias funções objetivo envolvidas na formulação inicial. A solução ótima para este novo problema é uma solução não dominada do problema multiobjetivo. Outras soluções não dominadas podem ainda ser encontradas com outros pesos na combinação linear das funções objetivo, designadas por soluções não dominadas suportadas, que pertencem à fronteira do chamado invólucro convexo. Existem outras soluções não dominadas que não podem ser encontradas desta forma, designadas por soluções não dominadas não suportadas, que se encontram no interior do invólucro convexo. Soluções dominadas, não dominadas suportadas e não dominadas não suportadas, estão representadas na figura 3.2.

Esta abordagem vai ser usada para obtenção das soluções iniciais da meta-heurística com *tabu search* para a resolução do problema de Steiner bi-critério, embora neste caso, como não

há garantia de que a solução ótima é encontrada no caso do problema mono-critério, não há garantia de que as soluções não dominadas para o problema bi-critério sejam na realidade não dominadas, dado que podem existir outras soluções, que não são encontradas, que as dominem.

A primeira aproximação para o conjunto de soluções do problema bi-critério de Steiner pode então ser encontrada minimizando a função escalar $c'(T)$:

$$\min_{T \in \mathcal{T}} c'(T) = \sum_{(i,j) \in T} \sum_{l=1}^{2} \epsilon_l c^l(i,j), \quad l = 1,2 \tag{3.4}$$

$$\overline{\epsilon} = (\epsilon_1, \epsilon_2) \in \varepsilon = \{\overline{\epsilon} : \epsilon_l \geq 0 \wedge \sum_{l=1}^{2} \epsilon_l = 1\} \tag{3.5}$$

Neste problema, a combinação linear convexa associa a cada arco da rede um novo custo que é uma soma pesada não negativa dos custos que correspondem a cada uma das métricas do problema.

Neste trabalho o objetivo não foi calcular todo o conjunto de soluções não dominadas suportadas (ou uma aproximação a esse conjunto) mas apenas as soluções que resultam do uso de três combinações diferentes de pesos. Os casos extremos, ou seja, os pesos que permitem calcular o ótimo de cada uma das funções objetivo separadamente, que definem os extremos do espaço das soluções não dominadas, e ainda uma combinação em que se tenta dar igual importância a cada uma das métricas, como se apresenta a seguir.

1. $\epsilon_1 = 1 \wedge \epsilon_2 = 0$
2. $\epsilon_1 = 0 \wedge \epsilon_2 = 1$
3. Dando igual importância a cada uma das métricas, os pesos obtidos são:
   $$\begin{cases} \overline{c^1}(i,j)\epsilon_1 = \overline{c^2}(i,j)\epsilon_2 \\ \epsilon_1 + \epsilon_2 = 1 \end{cases} \equiv \begin{cases} \epsilon_1 = \frac{1}{\overline{c^1}(i,j)} + 1 \\ \epsilon_2 = 1 - \epsilon_1 \end{cases}$$
   sendo $\overline{c^1}(i,j)$ a média do primeiro custo para todos os arcos da rede. Da mesma forma $\overline{c^2}(i,j) = 1$.

Neste problema bi-critério as soluções iniciais voltam a ser calculadas pelo algoritmo de Takahashi. O algoritmo de Takahashi é um algoritmo mono-critério, sendo usado no contexto do problema bi-critério com o terceiro custo que resulta da combinação linear referida anteriormente. No entanto, para ser usado com cada uma das funções objetivo em separado,

i. e. para a obtenção das soluções nos dois extremos (para $\epsilon_1 = 1 \wedge \epsilon_2 = 0$ e $\epsilon_1 = 0 \wedge \epsilon_2 = 1$), foi desenvolvida uma nova versão lexicográfica do algoritmo de Takahashi descrita no algoritmo 7, onde l_1 é uma das métricas e l_2 a outra métrica. A diferença relativamente ao algoritmo original é que, quando existem múltiplos nós terminais ainda não ligados à árvore construída até esse momento, à mesma distância do ponto de vista do primeiro custo que está a ser considerado, o nó terminal que é ligado à árvore é aquele cujo caminho que o liga tem o menor segundo custo. É de notar que o primeiro custo do ponto de vista do algoritmo lexicográfico tanto pode ser $c^1(i,j)$ como $c^2(i,j)$, sendo o segundo custo $c^2(i,j)$ e $c^1(i,j)$, respetivamente.

Como já foi referido, o algoritmo 7 é usado para obter as soluções iniciais nos extremos do espaço das soluções não dominadas, ou seja, dando peso 1 a uma das métricas e peso 0 à outra. Isso significa que o parâmetro l_1 que aparece no algoritmo representa a métrica com peso 1 enquanto que l_2 representa a outra métrica. Este algoritmo verifica se existe mais do que um caminho, para nós terminais ainda não ligados, com igual custo mínimo. Para isso é necessário retirar pelo menos 2 caminhos da *heap*. Vão-se retirando caminhos até se encontrar um caminho com maior custo para a métrica l_1 ou até não haver mais nenhum caminho na *heap*. Portanto, para todos os caminhos com igual custo para a métrica l_1 vai ser escolhido o caminho com menor custo para a métrica l_2. É de notar que se voltam a colocar na *heap* todos os outros caminhos que foram de lá retirados.

É ainda importante referir que, tal como no caso mono-critério, as soluções são obtidas considerando como nó inicial todos os nós terminais e ainda aplicando o algoritmo de Prim ao grafo induzido por cada solução, sendo de seguida eliminados os nós folha que não sejam nós terminais.

3.4.3 Meta-heurística para a resolução do BSTP

Nesta abordagem bi-critério, depois do cálculo das soluções iniciais, entra-se na pesquisa tabu para as duas métricas, ou seja, procuram-se as soluções não dominadas na vizinhança de cada uma das soluções iniciais, guardando-se todas as soluções não dominadas encontradas. Após isso, as melhores soluções para cada uma das métricas ainda serão alvo de um processo cujo objetivo é procurar melhores soluções na vizinhança dessas soluções para essa mesma métrica.

```
P_conj = ∅ // conjunto de caminhos
J = S − {s};
T ← (s, 0, 0);
P_{s,J} ← Dijkstra(s, J);
create(H);
for j ∈ J do
    insert(p_sj, H);
end
while J ≠ ∅ do
    find − min(p_ab, H);
    delete − min(p_ab, H);
    if b ∈ J then
        indice = 1;
        p_vy ← p_ab;
        while indice > 0 do
            indice = 0;
            if H ≠ ∅ then
                find − min(p_zx, H);
                delete − min(p_zx, H);
                if c^{l1}(p_vy) = c^{l1}(p_zx) ∧ x ∈ J then
                    indice = 1;
                    if c^{l2}(p_zx) < c^{l2}(p_vy) then
                        P_conj ← P_conj ∪ {p_vy};
                        p_vy ← p_zx;
                    end
                    else
                        P_conj ← P_conj ∪ {p_zx};
                    end
                end
                else
                    P_conj ← P_conj ∪ {p_zx};
                end
            end
        end
        insert(P_conj, H);
        P_conj = ∅;
        J = J − {y};
        for each n ∈ N_vy do
            P_{n,J} = Dijkstra(n, J);
            for j ∈ J do
                insert(p_nj, H);
            end
        end
        T ← T ∪ p_vy;
    end
end
```

Algoritmo 7: Algoritmo de Takahashi lexicográfico.

Dá-se o nome de *Bicriteria Steiner Tree Problem – Tabu* (BSTP - Tabu) à meta-heurística criada que é composta pelos seguintes passos:

1. Cálculo das soluções iniciais, tal como referido anteriormente, limitando-se o número de soluções a 75;

2. Aplicação do procedimento tabu bi-critério (algoritmo 8) para cada uma das métricas l, com $l = 1, 2$, a cada solução inicial, guardando-se todas as soluções não dominadas;

3. Aplicação do procedimento tabu bi-critério (algoritmo 8) à melhor solução encontrada para cada uma das métricas no passo anterior, limitando a variável *iteração* ao dobro do número de nós do grafo, intensificando-se assim a pesquisa nas regiões extremas no espaço das funções objetivo.

$iteração = 0$;
while $iteração < 20$ **do**
  $P = \infty$;
  for *cada nó de Steiner* j **do**
    $T' \leftarrow adicionar\_ou\_remover\_nó(j, T)$;
    if $((c^l(T') + penalização(j) < P)$ ***e*** $(Tabu(j) \leq iteração$ ***ou*** $c^l(T') < c^l(T_l^*)))$
    then
      $P = c^l(T')$;
      $w = j$;
    end
    if $naodominada(T')$ **then**
      $ND \leftarrow ND \cup \{T'\} \setminus D(T')$;
    end
  end
  $T = adicionar\_ou\_remover\_nó(w, T)$;
  $Tabu(w) = iteração + Random(x, y) + 1$;
  if $c^l(T) < c^l(T_l^*)$ **then**
    $T_l^* \leftarrow T$;
    $iteração = -1$;
  end
  $iteração = iteração + 1$;
end

Algoritmo 8: Procedimento tabu bi-critério.

No algoritmo 8, ND é o conjunto de soluções não dominadas, $naodominada(T)$ é um método que verifica se a árvore T corresponde a uma solução não dominada, devolvendo verdadeiro ou falso, e $D(T) \in ND$ é o conjunto de soluções dominadas pela nova solução T.

É de notar que sempre que uma solução não dominada é encontrada, é preciso verificar se alguma das soluções que já estavam no conjunto de soluções não dominadas é dominada pela nova solução encontrada.

Tal como já foi referido, as soluções não dominadas que vão sendo encontradas podem ser dominadas por outras soluções que podem ainda vir a ser descobertas, ou não. Não existe,

portanto, a garantia de que as soluções aqui denominadas como soluções não dominadas sejam mesmo soluções não dominadas do problema.

Nesta abordagem não foi considerada a diversificação de caminhos, apresentada em [4] e usada na resolução do problema mono-critério, embora tenha sido testada, como se pode ver no capítulo 4, onde se apresenta a análise de desempenho da meta-heurística bi-critério.

Como pode ser visto no capítulo seguinte, a diversificação de caminhos para o problema mono-critério não conduziu a melhorias significativas das soluções, sendo a qualidade das soluções encontradas maioritariamente definida pelas soluções iniciais. Assim, neste problema bi-critério, a diversificação do espaço de pesquisa, para além da diversificação contínua que é mantida, foi baseada na forma como foram obtidas as soluções iniciais. É de notar que, tal como foi referido no passo 2 da descrição do BSTP-Tabu, cada solução obtida entra duas vezes no algoritmo 8, numa das vezes procurando-se melhorar o custo da árvore, e da outra vez tentando diminuir o número de arcos na árvore.

3.4.4 Reduções

Na secção 3.1 foram apresentadas as reduções de grafos possíveis de aplicar no contexto do problema de Steiner mono-critério. Neste problema bi-critério algumas dessas reduções já não podem ser aplicadas pois poderiam impedir encontrarem-se algumas soluções de menor número de arcos. Por exemplo, uma redução (b) da secção 3.1 levaria a que o grafo reduzido representasse dois arcos apenas por um, o que poderia conduzir à existência de arcos paralelos, o que obrigava a repensar alguns algoritmos usados.

Neste problema bi-critério, apenas podem ser realizadas as reduções do tipo (a), algumas do tipo (b) e ainda outras do tipo (d). Podem-se fazer as reduções do tipo (b) quando estas dão origem a um arco paralelo de maior custo. Neste caso elimina-se o nó de Steiner de grau 2 e os respetivos arcos incidentes. A redução (d) pode ser aplicada quando o nó vizinho mais próximo de um nó terminal também é nó terminal. Em todos os outros casos estas reduções não podem ser aplicadas.

Sendo assim, as reduções para o problema bi-critério são as seguintes:

(a) Se i tem grau 1 e existe o arco (i, j) então tanto o arco como o nó i podem ser retirados do grafo. Se i for nó terminal então o nó j passa a ser nó terminal, se ainda não o for, e o nó i e o arco (i, j) são adicionados à solução.

(b) Sendo i nó de Steiner de grau 2 e existindo os arcos (j, i), (i, l) e (j, l), então i, (j, i) e (i, l) são removidos do grafo se $c^1(j, l) \leq (c^1(j, i) + c^1(i, l))$.

(c) Seja i um nó terminal e sejam j e l os 2 nós vizinhos mais próximos de i, respetivamente, sendo j um nó terminal. Se $c^1(i,j) \leq c^1(i,l)$, então o arco (i,j) faz parte da solução e o grafo pode ser contraído ao longo deste arco. Ou seja, o nó j é removido, os arcos incidentes em j ficarão incidentes em i e caso i fique com um par de arcos paralelos incidentes, o arco com maior custo é removido. No caso do nó l também ser nó terminal, a condição passa para $c^1(i,j) < c^1(i,l)$.

Segue-se uma sequência de realização de reduções, semelhante à anteriormente referida para o problema de Steiner mono-critério, ou seja, neste caso: b-c-a.

É de notar que as reduções aqui apresentadas são as possíveis para o caso do problema bi-critério apresentado. Estas reduções poderiam, no entanto, ser facilmente generalizadas para o caso geral do problema bi-critério de Steiner em que ambas as métricas são aditivas sem que nenhuma seja unitária.

Capítulo 4

Resultados Obtidos

De forma a comprovar a correção e eficácia das heurísticas e meta-heurísticas desenvolvidas são agora apresentados os resultados. Os grafos utilizados no problema mono-critério foram os grafos C e D da biblioteca existente em [6] para os quais os valores ótimos são conhecidos. Os resultados são ainda comparados com os apresentados em [4]. Para o problema bi-critério as redes usadas foram as redes B, C, I80 e I640 da mesma biblioteca, para as quais existem resultados para este mesmo problema em [9], com os quais os resultados agora obtidos podem ser comparados.

4.1 Meta-heurística de *Tabu Search* mono-critério – Análise de Desempenho

Foram aplicadas as técnicas já referidas de reduções de grafos previstas para o problema mono-critério e os resultados obtidos estão descritos na tabela 4.1, sendo que Grafos Artigo refere-se aos grafos obtidos em [4].

É de notar que algumas das reduções obtidas neste trabalho são iguais às obtidas em [4], pelo menos em termos do número de nós, nós terminais e arcos, enquanto que outras são diferentes. Isto acontece porque nalguns casos depende de por onde se começam a calcular as reduções e, portanto, a solução obtida não é única, existindo um número finito de soluções possíveis.

Na tabela 4.2 são agora apresentados os resultados dos cálculos das soluções do problema de Steiner para cada um dos grafos pelos algoritmos de Takahashi, P-Tabu e F-Tabu à semelhança dos resultados apresentados em [4]. Os resultados apresentados são na forma de percentagem de erro relativo ao valor ótimo, sendo o erro a diferença entre o resultado

	Grafos Originais			Grafos Artigo [4]			Meus Grafos		
Problema	|N|	|S|	|A|	|N|	|S|	|A|	|N|	|S|	|A|
C1	500	5	625	145	5	263	145	5	263
C2	500	10	625	130	8	239	130	8	239
C3	500	83	625	125	39	237	122	36	235
C4	500	125	625	116	42	233	107	36	221
C5	500	250	625	47	24	117	56	30	142
C6	500	5	1000	369	5	847	369	5	847
C7	500	10	1000	382	9	869	382	9	869
C8	500	83	1000	335	53	817	335	52	817
C9	500	125	1000	351	80	834	350	78	837
C10	500	250	1000	205	68	603	216	74	636
C11	500	5	2500	499	5	2184	499	5	2184
C12	500	10	2500	498	9	2236	498	9	2238
C13	500	83	2500	458	60	2066	456	58	2117
C14	500	125	2500	414	68	1886	422	73	2003
C15	500	250	2500	275	70	1276	304	89	1620
C16	500	5	12500	500	5	4740	500	5	4740
C17	500	10	12500	500	10	4704	498	8	4694
C18	500	83	12500	483	67	4637	479	63	4672
C19	500	125	12500	433	58	3431	451	76	4440
C20	500	250	12500	262	19	1692	344	98	3736
D1	1000	5	1250	274	5	510	274	5	510
D2	1000	10	1250	285	10	523	285	10	523
D3	1000	167	1250	228	70	445	243	74	473
D4	1000	250	1250	180	81	376	172	67	376
D5	1000	500	1250	118	67	113	140	61	282
D6	1000	5	2000	761	5	1741	761	5	1741
D7	1000	10	2000	754	10	1735	754	10	1735
D8	1000	167	2000	732	124	1711	731	123	1710
D9	1000	250	2000	660	157	1620	663	158	1631
D10	1000	500	2000	407	136	1282	410	137	1284
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	915	113	4461
D14	1000	250	5000	840	147	4048	850	146	4355
D15	1000	500	5000	525	135	2655	567	160	3408
D16	1000	5	25000	1000	5	10595	1000	5	10595
D17	1000	10	25000	999	9	10531	1000	10	10542
D18	1000	167	25000	944	111	9331	953	120	10078
D19	1000	250	25000	874	129	7943	909	161	9712
D20	1000	500	25000	532	42	3341	679	187	8259

Tabela 4.1: Reduções.

obtido e esse valor ótimo. Para o F-Tabu foram realizados 3 testes para cada problema, tal como em [4], devido à aleatoriedade existente ao longo do processo de pesquisa do algoritmo. O número de testes realizados está abaixo do número normal de testes usado para avaliar resultados obtidos através de algoritmos com estas características, devido ao grande esforço computacional exigido pelo F-Tabu, com tempos de execução muito elevados. Na tabela 4.2 encontra-se o melhor resultado, o pior e a média dos 3.

Problema	Ótimo	Tak.$_{artigo}$ [4]	Tak.	P-Tabu$_{artigo}$ [4]	P-Tabu	F-Tabu$_{artigo}$ [4]			F-Tabu		
						média	melhor	pior	média	melhor	pior
C1	85	0	0	0	0	0	0	0	0	0	0
C2	144	0	0	0	0	0	0	0	0	0	0
C3	754	0	0.13	0	0	0	0	0	0	0	0
C4	1079	0.09	0.09	0	0	0	0	0	0	0	0
C5	1579	0	0	0	0	0	0	0	0	0	0
C6	55	0	0	0	0	0	0	0	0	0	0
C7	102	0	0	0	0	0	0	0	0	0	0
C8	509	0	0	0	0	0	0	0	0	0	0
C9	707	0.99	0.85	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14
C10	1093	0.09	0	0	0	0	0	0	0	0	0
C11	32	0	6.25	0	6.25	0	0	0	3.13	3.13	3.13
C12	46	0	0	0	0	0	0	0	0	0	0
C13	258	0.78	1.55	0	0.39	0	0	0	0.78	0.78	0.39
C14	323	1.24	0.93	0.31	0.31	0.31	0.31	0.31	0.31	0.31	0.31
C15	556	0.18	0	0	0	0	0	0	0	0	0
C16	11	0	0	0	0	0	0	0	0	0	0
C17	18	0	5.56	0	0	0	0	0	0	0	0
C18	113	4.24	5.31	0.88	1.77	0	0	0	0.88	0.88	0.88
C19	146	4.79	4.11	0.68	2.74	0	0	0	2.05	2.05	2.05
C20	267	0.37	0	0	0	0	0	0	0	0	0
Média		0.64	1.43	0.10	0.58	0.02	0.02	0.02	0.35	0.35	0.35
D1	106	0	0	0	0	0	0	0	0	0	0
D2	220	0	0	0	0	0	0	0	0	0	0
D3	1565	0.77	0.58	0.26	0.32	0.19	0.06	0.26	0.26	0.13	0.32
D4	1935	0.16	0	0	0	0	0	0	0	0	0
D5	3250	0.06	0.12	0	0.09	0	0	0	0.02	0	0.06
D6	67	0	5.97	0	4.48	0	0	0	4.48	4.48	4.48
D7	103	0	0	0	0	0	0	0	0	0	0
D8	1072	1.70	1.68	0.47	1.03	0.37	0.37	0.37	0.84	0.84	0.84
D9	1448	0.83	0.76	0.41	0.55	0.21	0.21	0.21	0.48	0.48	0.48
D10	2110	0.38	0.33	0	0.05	0	0	0	0.05	0.05	0.05
D11	29	0	3.45	0	3.45	0	0	0	3.45	3.45	3.45
D12	42	0	0	0	0	0	0	0	0	0	0
D13	500	2.00	1.40	0	1.20	0	0	0	1.20	1.20	1.20
D14	667	0.75	0.90	0.15	0.45	0.15	0.15	0.15	0.45	0.45	0.45
D15	1116	0.36	0.27	0	0.09	0	0	0	0.09	0.09	0.09
D16	13	0	0	0	0	0	0	0	0	0	0
D17	23	0	0	0	0	0	0	0	0	0	0
D18	223	6.28	6.73	1.35	5.38	0.90	0.90	0.90	5.38	5.38	5.38
D19	310	5.81	4.19	0.65	3.55	0.43	0.32	0.65	2.90	2.90	2.90
D20	537	0.19	0.19	0	0	0	0	0	0	0	0
Média		0.96	1.33	0.16	1.03	0.11	0.10	0.13	0.98	0.97	0.99

Tabela 4.2: Resultados obtidos com o algoritmo de Takahashi, P-Tabu e F-Tabu.

O F-Tabu é a abordagem que apresenta melhores resultados. Comparando com os resultados do artigo nota-se a extrema importância da solução inicial. Devido às diferenças nas reduções e, principalmente, nos resultados obtidos pelo algoritmo de Takahashi, para os quais não há nenhuma garantia de serem iguais aos calculado em [4], os resultados obtidos por este algoritmo podem perder alguma qualidade e isso influencia todos os outros resultados.

nó	740	747	34	359	999
740	0	13	15	9	4
747	13	0	10	16	17
34	15	10	0	18	19
359	9	16	18	0	13
999	4	17	19	13	0

Tabela 4.3: Distância entre os nós terminais na solução.

nó	740	747	34	359	999
740	0	10	10	9	4
747	10	0	10	12	14
34	10	10	0	10	9
359	9	12	10	0	11
999	4	14	9	11	0

Tabela 4.4: Distância mínima entre os nós terminais na rede D11, dada pelo algoritmo de Dijkstra.

Existem situações, como no caso da rede D11, em que o algoritmo de Takahashi usado tem a possibilidade de encontrar a solução ótima, mas devido aos empates, resultantes da existência de mais do que um caminho de igual custo, a melhor que encontra é uma solução próxima da ótima, em termos de custo da árvore, mas em termos da árvore obtida esta é bastante distinta da solução ótima. Como consequência, nem o P-Tabu nem o F-Tabu conseguem melhorar esse resultado, e isso leva a um aumento da média da percentagem de erro. Recorreu-se à implementação de um algoritmo de Takahashi modificado baseado apenas no algoritmo de Dijkstra que considera em cada iteração todos os caminhos de igual custo mínimo para nós diferentes tendo sido encontradas 42 soluções, sendo que apenas uma dessas era a ótima.

Para clarificar a forma como o algoritmo de Takahashi pode chegar a diferentes soluções, será de seguida explicado com mais detalhe o caso da rede D11. Na figura 4.1 pode ser vista a solução ótima encontrada, sendo que os nós terminais são o 34, 747, 999, 740 e 359. A distância entre cada par de nós terminais na solução é mostrada na tabela 4.3, como se pode verificar na figura 4.1. Na tabela 4.4 estão representadas as distâncias mínimas entre os nós terminais no grafo completo dadas pelo algoritmo de Dijkstra e usadas pelo algoritmo de Takahashi.

O algoritmo de Takahashi, tal como foi referido na secção 3.2, tem como ponto de partida uma das linhas da tabela 4.4, consoante o nó terminal por onde começou. Como todos os nós terminais vão ser usados como o primeiro nó da árvore no cálculo da solução, o algoritmo vai começar por cada uma das linhas e, portanto, escolher para cada uma delas o nó terminal, que corresponde à coluna, cujo caminho tem menor custo. Esse vai ser o segundo nó terminal

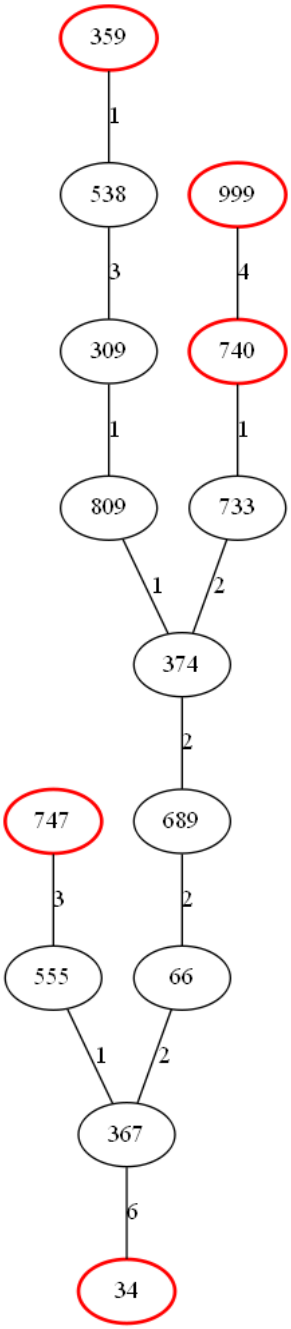

Figura 4.1: Solução ótima para a rede D11.

da árvore a ser ligado. Da análise da tabela pode concluir-se que o nó 34, como nó inicial, nunca poderá conduzir à solução ótima pois o caminho mais curto para cada um dos outros nós terminais é o caminho que o liga ao nó 999, que é um caminho totalmente distinto daquele que está presente entre esses dois nós na solução ótima. Deixamos então essa opção de fora e vamos analisar as outras 4.

A opção de iniciar com o nó 740 é igual àquela em que o nó inicial é o 999, porque em ambos os casos o nó inicial vai ligar-se ao outro nó. Após isso, existem 2 possibilidades distintas: ligar o nó 359 ou o nó 34 à árvore, que estão ambos à mesma distância. Ligar o nó 34 está fora de questão pelas razões já apresentadas. Sobra então a ligação ao nó 359, que é exatamente o caminho representado na figura 4.1 entre o nó 740 e o nó 359. Falta apenas ligar o nó 34 e o nó 747. A ligação a esses nós na solução é feita pelo nó 374 (que não é nó terminal) mas aqui isso não é possível pois sabe-se, pelo algoritmo de Dijkstra, que o caminho mais curto de 374 a esses nós tem custo 10 (para o nó 34) e o caminho mais curto entre os nós presentes na árvore atual e os nós terminais ainda por ligar tem custo 9 (para o nó 34).

Sobra então apenas o nó 747 como nó inicial. Neste caso existe logo à partida um empate, entre os dois caminhos de custo 10 para os nós 34 e 740. Olhando para a solução excluímos o caminho para o nó 740. Assim liga-se o nó 747 ao nó 34 pelo caminho que consiste nos nós 555 e 367, tal como na solução. Neste momento volta a aparecer um empate por existir um caminho de custo 9 que liga o nó 34 ao nó 999 e um caminho de custo 9 entre o nó 367 (que não é nó terminal) e 740, não existindo mais nenhum caminho com menor custo que esses. O único desses caminhos que poderia levar à solução seria o caminho que juntava à árvore o nó 740. Fazendo isso surge depois naturalmente o resto da solução sem problemas. Primeiro o nó terminal mais próximo é o nó 999 à distância de 4 do nó 740 e depois junta-se finalmente à árvore o nó 359 através do caminho de custo 6 que o liga ao nó 374.

Esta rede serve então de exemplo para os casos em que a solução ótima está ao alcance do algoritmo de Takahashi mas isso não garante que o algoritmo a encontre a não ser que teste todas as possibilidades. Experimentalmente, usando o algoritmo de Takahashi modificado, encontram-se 12 soluções diferentes usando como nó inicial o nó 740, 6 soluções para o nó inicial 747, 4 para o nó 34, 8 para o nó 359 e 12 soluções diferentes para o nó inicial 999.

Não se faz referência à possibilidade de melhorar cada uma das outras 41 soluções para esta rede, pela aplicação de um algoritmo de determinação da árvore mínima abrangente ao grafo induzido por cada solução obtida pelo algoritmo de Takahashi, pois só seria possível encontrar a solução ótima no caso de todos os nós dessa solução estarem presentes em soluções obtidas por esse algoritmo, e isso nunca acontece para estas 41 soluções. Para além disso,

esta análise foi feita supondo que existe apenas uma solução ótima mas podem existir outras soluções ótimas alternativas que este algoritmo de Takahashi não encontrou pois apenas testa todos os caminhos alternativos para nós diferentes com o mesmo custo mínimo e não testa todos os caminhos alternativos com o mesmo custo entre o mesmo par de nós. Por conseguinte, é possível encontrar ainda mais soluções com o algoritmo de Takahashi.

A comparação com os resultados do artigo leva a crer que os resultados apresentados são fruto de vários testes e que são usados os melhores resultados possíveis, em termos das soluções iniciais, para mais facilmente se comprovar a eficácia da meta-heurística, pois existem casos, como a rede D06 em que o algoritmo de Takahashi modificado não encontra a solução ótima, mas ela é encontrada pelo algoritmo de Takahashi em [4].

Nas redes D os resultados são bem piores que os anteriores, quando comparados com os resultados do artigo, devido à maior complexidade destas redes, com o consequente aumento do espaço de pesquisa, o que evidencia a dependência de uma boa solução inicial. Embora em média os resultados obtidos pela nova implementação do algoritmo de Takahashi, em termos relativos, não sejam piores do que os obtidos para as redes C, a ideia com que se fica é que as soluções obtidas pelos autores em [4] foram escolhidas de forma a otimizar os resultados obtidos pelo P-Tabu. Os resultados apresentados nesta dissertação, obtidos pela nova implementação da meta-heurística para a resolução do problema mono-critério, não tiveram este tipo de tratamento.

Conclui-se que esta meta-heurística de *tabu search* obtém melhores resultados que o algoritmo de Takahashi, mas está muito dependente da solução inicial para encontrar soluções muito próximas ou iguais às ótimas com o P-Tabu, concluindo-se ainda que o mecanismo de diversificação de caminhos do F-Tabu é pouco eficiente.

4.2 Meta-heurística de *Tabu Search* bi-critério – Análise de Desempenho

Fazendo as reduções dos grafos propostas para este problema bi-critério obtém-se os resultados apresentados na tabela 4.5. Mostram-se os resultados obtidos para as redes B e C para se poder comparar com os resultados das reduções aplicadas no problema mono-critério, por um lado, e, por outro lado, para se analisarem os efeitos das reduções em redes mais pequenas (as redes B). As redes I80 e I640 também foram reduzidas através dos mesmos métodos, mas os resultados não são apresentados porque cada um desses conjunto de redes é constituído

por 100 redes e não seriam facilmente aqui apresentadas.

Podem-se avaliar facilmente os resultados obtidos para a primeira métrica pois o valor ótimo é conhecido e está disponível em [6]. No entanto isso não é assim para a segunda métrica, ou seja, não se conhece o valor ótimo para o número de arcos da solução. Uma opção seria comparar os resultados com o número mínimo possível de arcos, que é $|S| - 1$ para um conjunto S de nós terminais, mas esse valor só é possível de alcançar se a rede o permitir pois nem sempre existem ligações diretas entre os nós terminais.

Sendo assim, os resultados serão acrescentados à comparação de resultados existente em [9], onde um outro algoritmo para este mesmo problema de Steiner bi-critério foi apresentado. Em [9] são ainda apresentados os resultados de mais dois algoritmos. Por conseguinte, os novos resultados serão comparados com os resultados de 3 outros algoritmos que nas figuras seguintes são designados por: BCSTP(1-1), BCSTP(2-10) e BCSTP. Os novos resultados obtidos com *tabu search* estão identificados nestas figuras por BSTP-Tabu.

Esses algoritmos com os quais o BSTP-Tabu vai ser comparado são baseados no algoritmo de Kou [14] que é um algoritmo de referência para o problema mono-critério.

O algoritmo apresentado por Kou é realmente muito simples e começa pelo cálculo dos caminhos mais curtos entre cada par de nós terminais. Assim, é construído um grafo constituído apenas pelos nós terminais e pelos caminhos calculados anteriormente, sendo que esses caminhos são representados no novo grafo como se se tratassem apenas de um arco, cujo custo corresponde ao custo total do caminho. Após a obtenção desse grafo, calcula-se a árvore mínima abrangente e, no final, substituem-se os arcos pelos caminhos iniciais, tal como estavam no grafo original. Conclui-se o algoritmo eliminando eventuais ciclos, bem como qualquer nó não terminal que tenha ficado como nó folha (nó de grau 1).

Os algoritmos BCSTP(1-1), BCSTP(2-10) e o BCSTP baseiam-se na mesma estratégia inicial do algoritmo de Kou, i. e., partem da construção de um sub-grafo que interliga todos os nós terminais, e recorrem a algoritmos exatos bi-critério para determinação de todas as árvores abrangentes que são soluções não dominadas suportadas nesse sub-grafo.

De seguida, são apresentados os resultados obtidos pelo BSTP-Tabu comparando-os com os obtidos pelos outros algoritmos. Nas figuras 4.2 e 4.3 encontram-se os resultados obtidos para o segundo custo, ou seja, para o número mínimo de arcos da solução. Esses resultados são apresentados considerando como ótimo para cada rede o menor resultado encontrado entre os 4 algoritmos.

Observando as figuras nota-se que os resultados do BSTP-Tabu para o número mínimo de arcos na solução são melhores que os resultados obtidos pelos outros algoritmos. Nas

	Grafos Originais			Meus Grafos		
Problema	|N|	|S|	|A|	|N|	|S|	|A|
C1	500	5	625	307	5	432
C2	500	10	625	305	8	428
C3	500	83	625	281	57	404
C4	500	125	625	277	70	402
C5	500	250	625	197	88	319
C6	500	5	1000	470	5	969
C7	500	10	1000	460	9	960
C8	500	83	1000	451	65	950
C9	500	125	1000	446	96	945
C10	500	250	1000	364	136	859
C11	500	5	2500	500	5	2500
C12	500	10	2500	498	9	2498
C13	500	83	2500	485	69	2480
C14	500	125	2500	470	95	2466
C15	500	250	2500	389	139	2343
C16	500	5	12500	500	5	12500
C17	500	10	12500	498	8	12485
C18	500	83	12500	482	65	12363
C19	500	125	12500	446	71	11875
C20	500	250	12500	338	88	9403
B1	50	9	63	18	4	26
B2	50	13	63	32	8	44
B3	50	25	63	21	8	32
B4	50	9	100	45	8	95
B5	50	13	100	42	11	91
B6	50	25	100	38	16	84
B7	75	13	94	46	11	64
B8	75	19	94	39	12	56
B9	75	38	94	26	11	43
B10	75	13	150	67	11	142
B11	75	19	150	66	15	140
B12	75	38	150	51	18	121
B13	100	17	125	59	12	83
B14	100	25	125	54	15	79
B15	100	50	125	42	19	62
B16	100	17	200	89	13	188
B17	100	25	200	90	22	190
B18	100	50	200	66	25	158

Tabela 4.5: Reduções bi-critério.

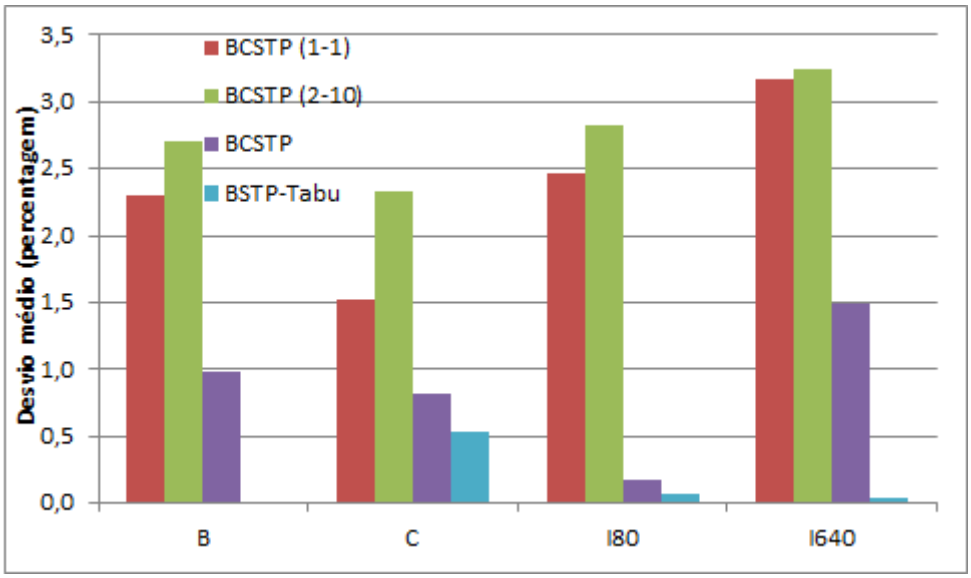

Figura 4.2: Desvio médio relativo ao mínimo de *hop count*.

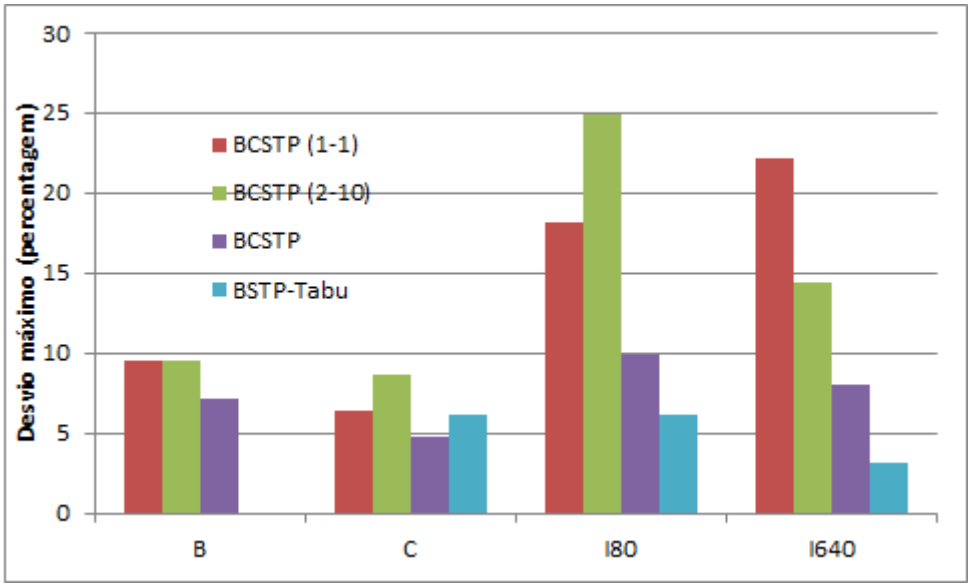

Figura 4.3: Desvio máximo relativo ao mínimo de *hop count*.

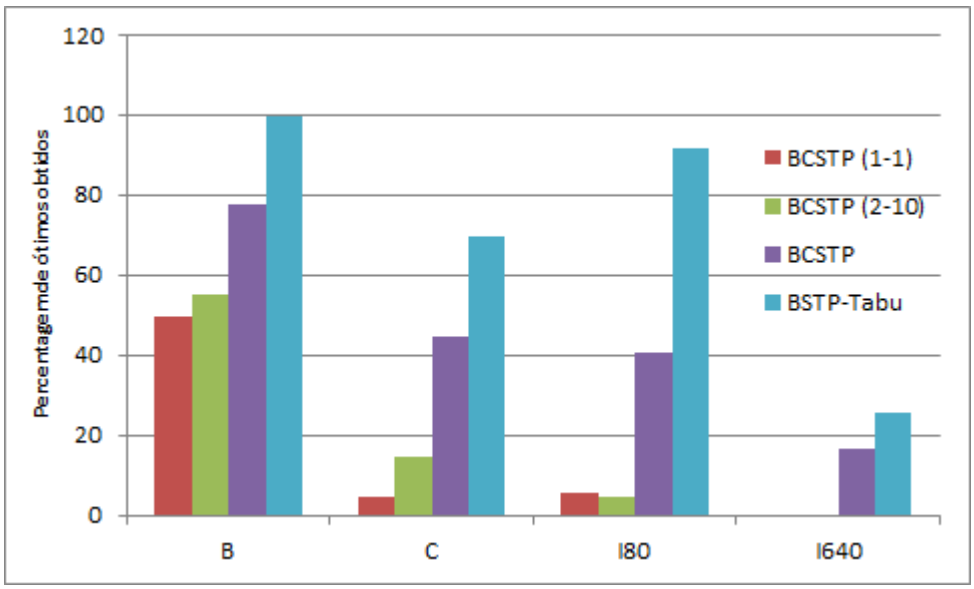

Figura 4.4: Percentagem de ótimos obtidos.

redes B, não há sequer nenhum caso em que a melhor solução para esta métrica não seja obtida pelo BSTP-Tabu. É nas redes C que os resultados são piores, apesar de se ter obtido a solução ótima em 18 das 20 redes. Nesses 2 casos em que não são obtidas as soluções ótimas, consegue até ficar pior, em termos percentuais, que o pior resultado obtido pelo BCSTP para as redes C, sendo esse conjunto de redes o único dos 4 conjuntos em que isso acontece, como se pode ver pela figura 4.3. A diferença em relação ao ótimo nem é muito grande, no pior caso a diferença é de 17 para 16 arcos, mas uma vez que essas 2 redes têm poucos nós terminais e poucos arcos a ligá-los, o desvio médio é considerável em termos percentuais.

Nas redes I80 as diferenças são pequenas, pois apesar do BSTP-Tabu ser o melhor, ele é pouco melhor que o BCSTP, o que leva a crer que os resultados obtidos por ambos os algoritmos já estarão muito próximos dos menores possíveis.

Nas figuras 4.4, 4.5 e 4.6 encontram-se os melhores resultados obtidos para a primeira métrica. Para esta métrica o ótimo é conhecido e, então, para além do desvio médio e máximo em relação ao ótimo também é apresentada a percentagem de ótimos obtidos por cada algoritmo.

Nesta métrica o algoritmo BSTP-Tabu volta a ser muito melhor que os outros. A percentagem de ótimos obtidos é sempre muito alta e muito superior à dos outros algoritmos, exceto no conjunto de redes I640. Mas mesmo nesse conjunto de redes, onde a percentagem de ótimos é muito mais baixa, pouco acima de 20%, continua a ser o algoritmo com maior percentagem de ótimos.

No desvio médio relativo ao custo ótimo da primeira métrica, o BSTP-Tabu é o que obtém

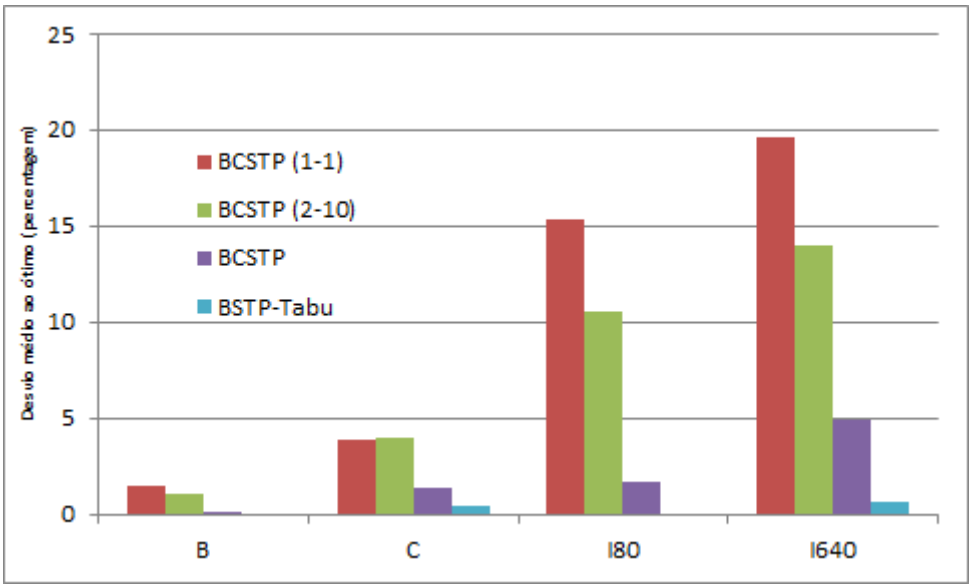

Figura 4.5: Desvio médio ao ótimo da primeira métrica.

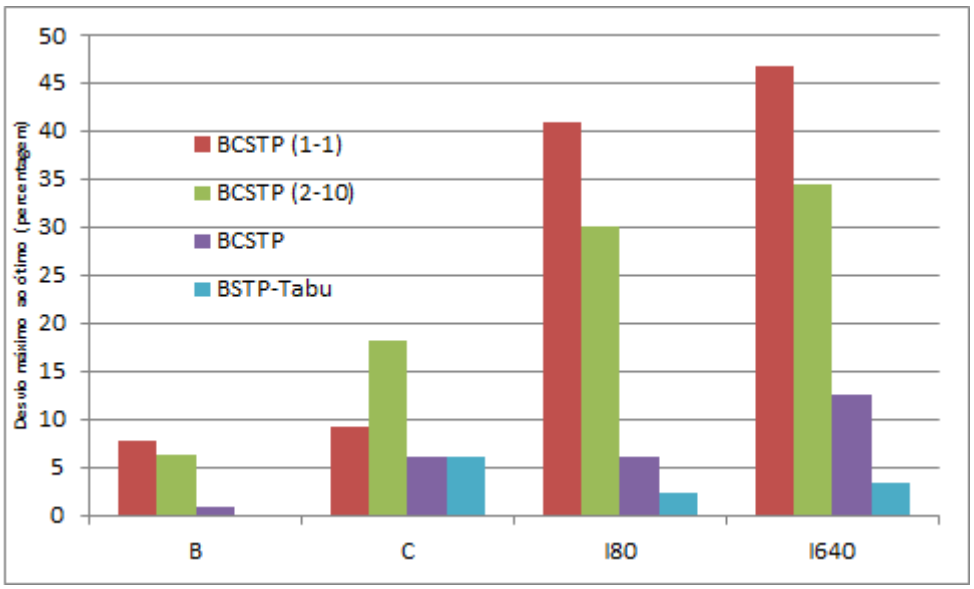

Figura 4.6: Desvio máximo ao ótimo da primeira métrica.

os melhores resultados, como se pode ver na figura 4.5. No desvio máximo, no geral, os resultados obtidos por este algoritmo, apresentados na figura 4.6, são superiores aos obtidos pelos restantes algoritmos, sendo apenas semelhantes aos obtidos pelo algoritmo BCSTP no caso das redes C. Repare-se que no conjunto de redes I640, o o BSTP-Tabu tem desvio máximo relativo inferior ao desvio médio relativo do algoritmo BCSTP, que é o segundo algoritmo com os melhores resultados.

Os resultados apresentados anteriormente para o problema de Steiner bi-critério são os melhores resultados obtidos para os extremos do espaço de soluções não dominadas. Isto significa que os resultados apresentados poderiam ser obtidos para cada um dos problemas mono-critério considerando cada função objetivo em separado. O facto destes resultados já serem melhores para o BSTP-Tabu indicia que o conjunto de soluções não dominadas também o seja. Nas figuras 4.7, 4.8[1] e 4.9[2] estão representados os conjuntos de soluções 'não dominadas' obtidas por cada algoritmo para as redes B18, I640-315 e I640-335, respetivamente.

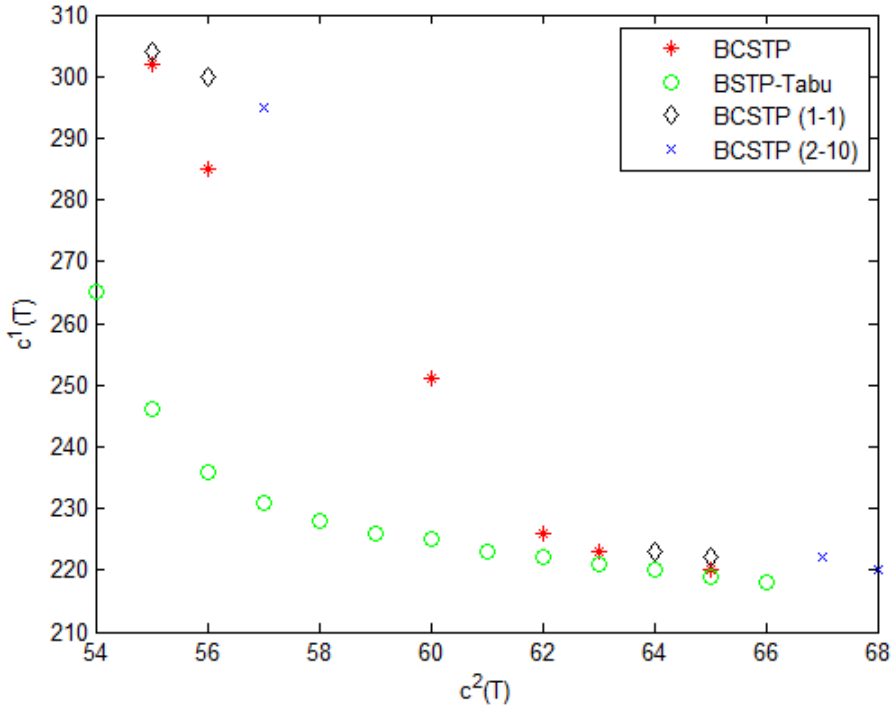

Figura 4.7: Aproximação ao conjunto de soluções não dominadas obtidas por cada algoritmo para a rede B18.

[1]Por indisponibilidade dos dados, nesta figura estão representadas apenas as duas soluções extremas, do espaço das soluções 'não dominadas', para o BCSTP(1-1) e o BCSTP(2-10), embora o BCSTP(1-1) encontre 14 soluções enquanto que o BCSTP(2-10) encontra 20 soluções 'não dominadas' para esta rede. Os conjuntos de soluções 'não dominadas' para o BCSTP e para o BSTP-Tabu estão completos.

[2]À semelhança da figura anterior, embora o BCSTP(1-1) e o BCSTP(2-10) tenham apenas duas soluções representadas, são encontradas para estas redes, por estes algoritmos, 28 soluções e 29 soluções 'não dominadas', respetivamente.

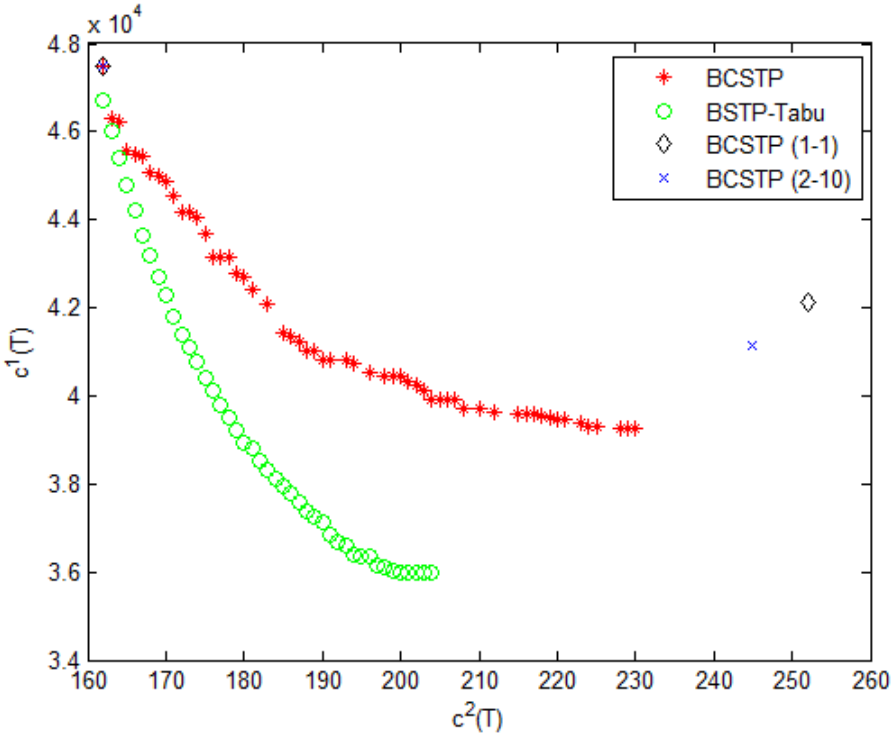

Figura 4.8: Aproximação ao conjunto de soluções não dominadas obtidas por cada algoritmo para a rede I640-315.

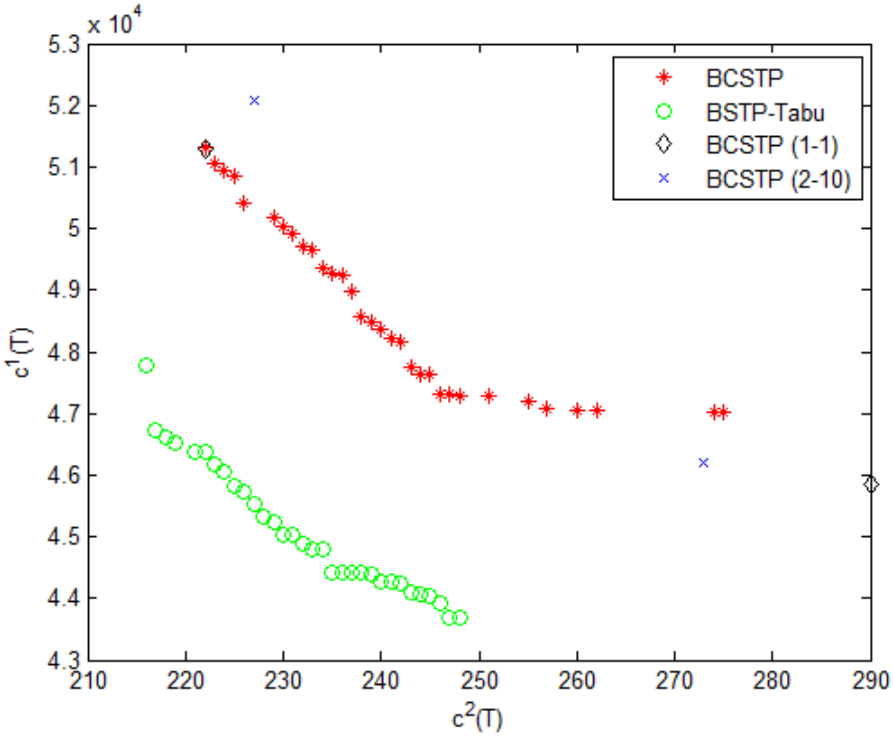

Figura 4.9: Aproximação ao conjunto de soluções não dominadas obtidas por cada algoritmo para a rede I640-335.

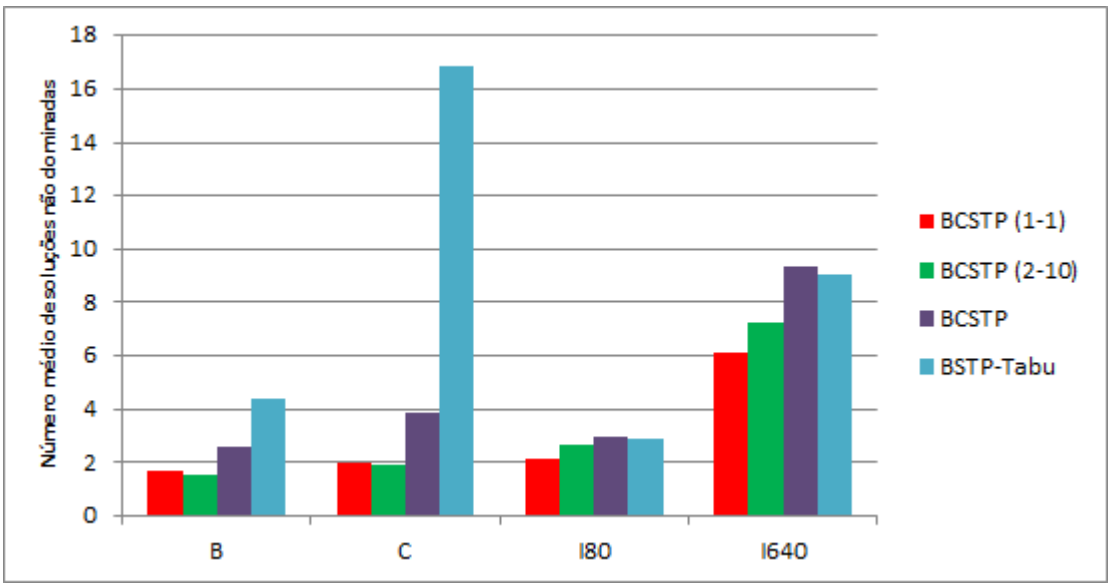

Figura 4.10: Número médio de soluções não dominadas obtidas pelas heurísticas.

Como se vê nas figuras anteriores, as soluções não dominadas obtidas pelo BSTP-Tabu domina as obtidas pelos outros algoritmos.

Para concluir, apresenta-se ainda o número médio de soluções não dominadas obtidas por cada algoritmo. Esses resultados podem ser vistos na figura 4.10, onde o algoritmo que tende a encontrar o maior número de soluções não dominadas é o BSTP-Tabu para o caso dos conjuntos de redes B e C, tendendo a aproximar-se do BCSTP nas restantes redes.

4.2.1 Análise da diversificação do espaço de pesquisa

Os resultados obtidos pelo BSTP-Tabu, apresentados anteriormente, foram aqueles que usaram a diversificação proposta no âmbito deste trabalho, na secção 3.4.2 para a meta-heurística de *tabu search* para o problema de Steiner bi-critério. A diversificação proposta substitui então a diversificação de caminhos do F-Tabu proposta em [4] para o problema mono-critério.

De modo a ilustrar o percurso seguido para se chegar a esta forma de diversificação são mostrados a seguir resultados obtidos com as diferentes formas de diversificação apresentadas na secção 3.4.2, estudadas no âmbito desta dissertação.

A diversificação de caminhos, como já foi referido, não consegue melhorar significativamente as soluções iniciais obtidas pelo P-Tabu. Para comparar com esta forma de diversificação, foram consideradas mais duas novas formas de diversificação do espaço de pesquisa que foram testadas separadamente para as redes B, C e I80.

Os resultados apresentados de seguida foram então obtidos por três abordagens diferentes.

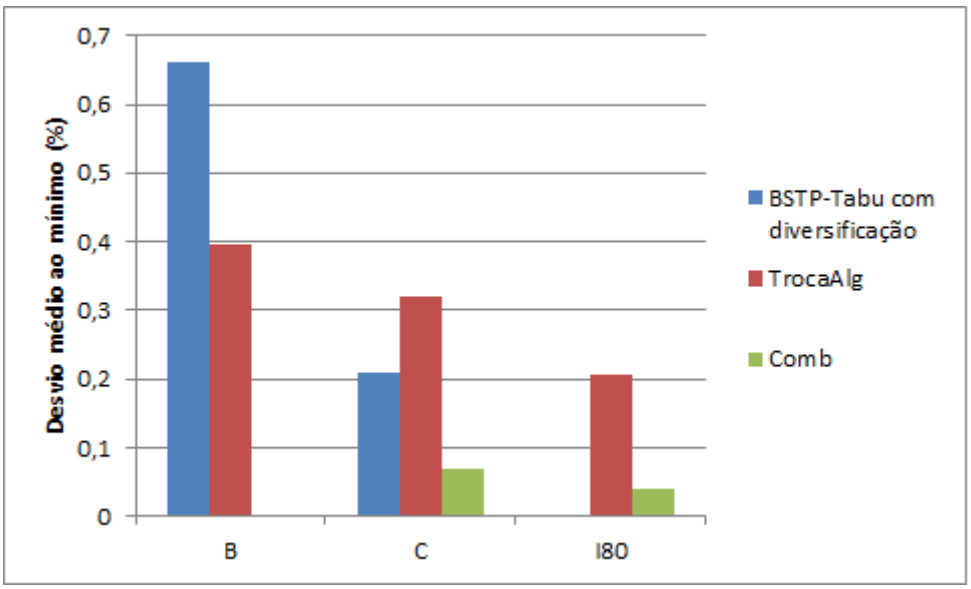

Figura 4.11: Desvio médio ao mínimo de *hop count*.

BSTP-Tabu com diversificação refere-se ao algoritmo que obtém soluções iniciais apenas para os extremos do espaço das funções objetivo, i. e., as soluções iniciais são as que minimizam separadamente cada uma das funções objetivo, recorrendo-se depois ao procedimento tabu bi-critério (algoritmo 8) apenas com o objetivo de encontrar novas soluções na vizinhança da solução encontrada tentando apenas melhorar o custo da solução encontrada pelo algoritmo de Takahashi. Neste algoritmo é ainda usada na última fase a diversificação de caminhos tal como no caso do problema mono-critério. *TrocaAlg* refere-se ao método que calcula as mesmas soluções iniciais que o algoritmo anterior mas para as quais o tabu bi-critério procura soluções na vizinhança dessa solução do ponto de vista do outro custo da solução (que não o usado pelo algoritmo de Takahashi). *Comb* refere-se à obtenção das soluções iniciais através da combinação linear das duas métricas, tal como está exposto na secção 3.4.2. Neste caso o tabu bi-critério, para cada solução inicial, procura encontrar soluções que melhorem cada um dos custos das soluções iniciais separadamente, ou seja, o algoritmo 8 é usado com $l = 1$ e $l = 2$, para cada solução inicial.

É de notar que os resultados obtidos pelo BSTP-Tabu, apresentados na secção anterior, resultam da combinação destas três abordagens, com exceção da diversificação de caminhos.

Na figura 4.11 apresenta-se o desvio médio relativamente ao mínimo de *hop count*, obtido pelo conjunto dos três algoritmos. Nas figuras 4.12 e 4.13 apresentam-se o desvio médio relativo ao ótimo da primeira métrica e a percentagem de ótimos obtidos, respetivamente.

Mais do que obter melhores resultados em termos de percentagens de ótimos e do desvio médio ao ótimo, interessa é saber se estas duas novas abordagens conseguem obter melhores

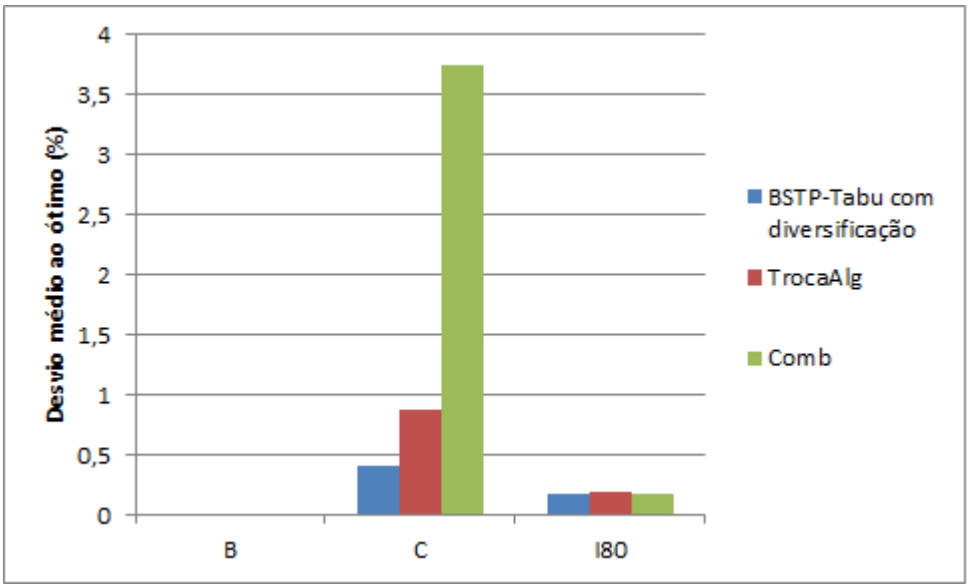

Figura 4.12: Desvio médio ao ótimo da 1ª métrica.

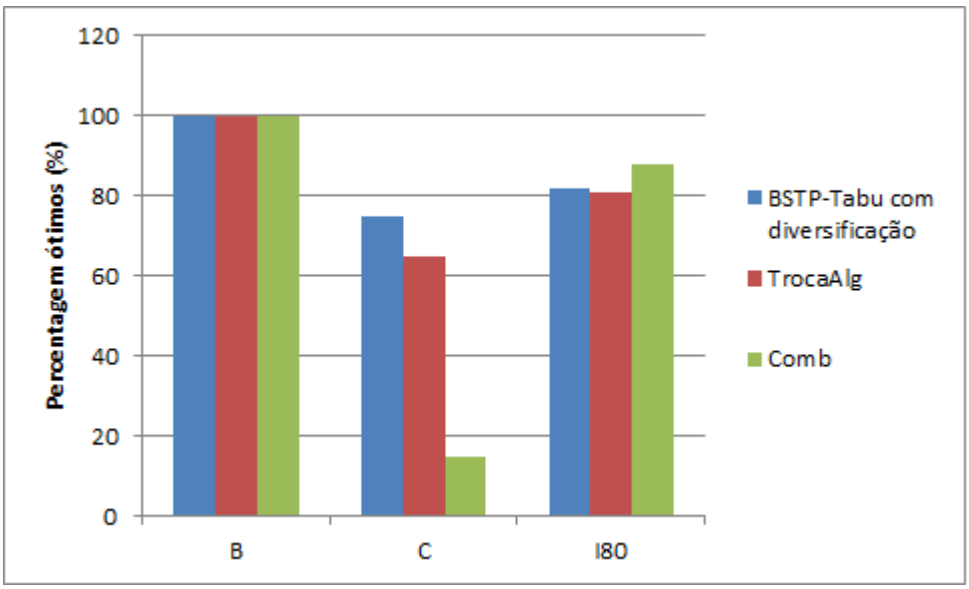

Figura 4.13: Percentagem de ótimos obtidos.

	TrocaAlg		Comb	
rede	primeira métrica	*hop count*	primeira métrica	*hop count*
B	0	1	0	2
C	0	0	0	1
I80	10	0	15	0

Tabela 4.6: Número de vezes em que são obtidas melhores soluções relativamente ao *BSTP-Tabu com diversificação*.

resultados, em algumas redes, para qualquer uma das métricas. Esse era o objetivo, descobrir se alguma das abordagens, ou ambas, poderiam substituir a diversificação de caminhos ao permitirem procurar soluções não dominadas em regiões inexploradas do espaço de soluções e isso é o que se pretende mostrar nesta secção. Na tabela 4.6 encontram-se o número de redes em que se obteve melhores resultados, relativamente ao *BSTP-Tabu com diversificação*, para cada uma das métricas e por cada um dos algoritmos.

É de referir ainda que os resultados obtidos pelo *BSTP-Tabu com diversificação* não seriam muito diferentes dos apresentados, se a diversificação de caminhos tivesse sido excluída.

Capítulo 5

Conclusão

Nesta dissertação de mestrado foi estudado o problema de Steiner como primeiro passo para a obtenção de ligações multiponto-multiponto em redes de transporte. Assim foram estudadas heurísticas de referência e uma meta-heurística, apresentada em [4], muito eficiente que usa *tabu search* para a resolução deste problema.

Formulou-se um problema bi-critério de Steiner com o objetivo de se encontrar ligações multiponto-multiponto, não só de custo mínimo, mas que utilizem também o mínimo de recursos da rede. Este problema de Steiner bi-critério foi resolvido recorrendo a uma extensão da meta-heurística referida para o problema bi-critério. Foram identificados os aspetos mais críticos da meta-heurística, sobretudo relacionados com a diversificação de caminhos e com a obtenção das soluções iniciais, e propôs-se uma nova forma de diversificação para o problema bi-critério, para a qual se apresentam os primeiros resultados.

Os resultados apresentados são superiores aos anteriormente obtidos por outras abordagens para a resolução do mesmo problema bi-critério de Steiner, o que é bastante encorajador na prossecução deste trabalho.

Na sequência da colaboração com a PT Inovação, apresenta-se um esboço do problema de Steiner com restrições, que poderá ser estendido a curto prazo a uma formulação multi-critério, tirando partido dos resultados agora obtidos recorrendo a *tabu search*.

O trabalho foi desenvolvido em linguagem JAVA.

Bibliografia

[1] P. Winter and J. MacGregor Smith, Path-distance heuristics for the Steiner problem in undirected networks, Algorithmica 7 (1992), 309–327.

[2] V.J. Rayward-Smith and A. Clare, On finding Steiner vertices, Networks 16 (1986), 283–294.

[3] John Hopcroft and Robert Tarjan, Efficient Algorithms for Graph Manipulation [H], Comunnications of the ACM (1973), 372-378.

[4] Michel Gendreau, Jean-Francois Larochelle, and Brunilde Sanso, A Tabu Search Heuristic for the Steiner Tree Problem, Networks 34 (1999), 162-172.

[5] H. Esbensen, Computing near-optimal solutions to the Steiner Problem in a graph using a genetic algorithm, Networks 26 (1995), 173-185.

[6] `http://steinlib.zib.de/steinlib.php`

[7] P. Soriano and M. Gendreau, Diversification strategies in tabu search algorithms for the maximum clique problem, Annals of Operations Research 63 (1996), 189–207.

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

[9] 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.

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

[11] C. Ribeiro and M. Souza, Tabu search for the Steiner Problem in Graphs, Networks 36 (2000), 138–146.

[12] L. L. Pinto and G. Laporte, An efficient algorithm for the Steiner Tree Problem with revenue, bottleneck and hop objective functions, European Journal of Operational Research 207 (2010), 45–49.

[13] Jens Vygen, Faster algorithm for optimum Steiner trees, Information Processing Letters 111 (2011), 1075–1079.

[14] L. Kou, G. Markowsky and L. Berman, A fast algorithm for Steiner trees, Acta Informatica 15 (1981), 141-145.

[15] Bharat Krishna, C.Y. Roger Chen and Naresh K. Sehgal, A Novel Ultra-Fast Heuristic for VLSI CAD Steiner Trees, GLSVLSI'03, April 28-29, 2003, Washington, DC, USA.

[16] Samira Noferesti and Hamed Shah-Hosseini, A Hybrid Algorithm for Solving Steiner Tree Problem, International Journal of Computer Applications (2012), 0975 – 8887.

[17] Sarah Steiner and Tomasz Radzik, Computing all efficient solutions of the biobjective minimum spanning tree problem, Computers Operations Research 35 (2008), 198 – 211.

Apêndice A

Problema de Steiner sugerido pela PT Inovação

A heurística de Takahashi adapta-se muito facilmente para a resolução de um problema de Steiner com restrições que foi sugerido pela PT Inovação. Neste problema pretende-se obter uma árvore de Steiner para interligação de um conjunto S de nós terminais e, para além disso, pretende-se ainda que um conjunto de nós e/ou arcos adicionais também pertençam à solução.

Como se pretende obter uma árvore, é necessário verificar se o conjunto de elementos adicionais na solução é ou não coerente com o problema que se quer resolver. Isso porque os arcos poderiam criar ciclos na solução e então deixava-se de ter uma árvore. Para se evitar ciclos é necessário um pré-processamento dos elementos adicionais. Após se verificar que todos os elementos adicionais pertencem ao grafo original, começa-se por aplicar o algoritmo apresentado na secção 2.5, de modo a obter os diferentes sub-grafos conexos B_i, que possam eventualmente existir no conjunto de elementos adicionais. Após a identificação dos sub-grafos, é necessário eliminar possíveis ciclos, que são facilmente eliminados aplicando o algoritmo de Prim a cada sub-grafo.

Após esse pré-processamento podiam-se ligar os diferentes B_i através dos caminhos mais curtos entre eles. Depois disso, seria preciso apenas aplicar o algoritmo de Takahashi, apresentado na secção 3.2, partindo da árvore já construída com os diferentes B_i e com as suas interligações, acrescentando de seguida os restantes nós terminais.

No entanto, em vez de se ligarem esses sub-grafos B_i entre si, optou-se por adicionar todos os nós pertencentes aos diferentes B_i ao conjunto de nós terminais. Neste caso, ao resolver o algoritmo de Takahashi é necessário, ao adicionar um novo caminho da solução atual para

um nó pertencente a um dos B_i, que se adicione também à solução todo o sub-grafo. De seguida, os nós desse sub-grafo são retirados do conjunto de nós terminais ainda por ligar. Esta nova forma de ligar os elementos adicionais conduz a melhores resultados.

Os passos do algoritmo são os seguintes:

1. Verificar se os elementos adicionais existem no grafo original G;
2. Calcular os sub-grafos conexos B_i pelo algoritmo proposto por Hopcroft e Tarjan [3], com $i = 1, \ldots, I$, sendo I o número total de sub-grafos identificados;
3. Aplicar o algoritmo de Prim a cada um dos B_i de modo a eliminar eventuais ciclos nesses sub-grafos. Note-se que B_i passa a ser uma árvore;
4. Considerando que cada sub-grafo $B_i = (N_i, A_i)$, em que N_i é o conjunto de nós do sub-grafo e A_i o conjunto de arcos, resolver o algoritmo de Takahashi modificado (algoritmo 9), tendo como conjunto de nós terminais $S' = S \cup N_1 \cup N_2 \cup \cdots \cup N_I$.

Relembra-se que N_{ab} é o conjunto de nós pertencentes ao caminho p_{ab}, excetuando o nó a.

```
J = S' − {s};
T_T ← (s, 0, 0);
P_{s,J} ← Dijkstra(s, J);
create(H);
for j ∈ J do
    insert(p_{sj}, H);
end
while J ≠ ∅ do
    find − min(p_{ab}, H);
    delete − min(p_{ab}, H);
    if b ∈ J then
        J = J − b;
        T_T ← T_T ∪ p_{ab};
        for f = 1, 2, ..., I do
            if b ∈ B_f then
                J = J \ N_f;
                T_T ← T_T ∪ B_f;
                break;
            end
        end
        for each n ∈ (N_{ab} ∪ N_f) do
            P_{n,J} = Dijkstra(n, J);
            for j ∈ J do
                insert(p_{nj}, H);
            end
        end
    end
end
```

Algoritmo 9: Algoritmo de Takahashi modificado.

Apêndice B

Artigo

Apresenta-se de seguida um artigo resultante deste trabalho, já submetido a uma conferência internacional.

Tabu Search for Bicriteria Multipoint-to-Multipoint Virtual Connections

Rui Barbosa
Department of Electrical and Computer Engineering
Faculty of Sciencies and Technology, University of Coimbra
Email: r.ruimiguelbarbosa@gmail.com

Lúcia Martins
Department of Electrical and Computer Engineering
Faculty of Sciencies and Technology, University of Coimbra
Institute for Systems and Computers Engineering at Coimbra
Rua Antero de Quental, 199 3000–033 Coimbra Portugal
Email: lucia@deec.uc.pt

***Abstract*—Multipoint-to-multipoint virtual connections in communication transport networks can be defined through a network management system where they can be computed as a tree which interconnects a given subset of network nodes. This problem can be modelled as a Steiner tree problem is graphs with the aim of obtaining the lowest-cost tree for interconnecting the subset of network nodes involved in each multipoint-to-multipoint virtual connection, where the cost can represent, for instance, the occupancy of each link. In real networks however it may be advantageous not only to obtain the lowest-cost tree but also to obtain the tree with the minimum number of arcs, in order that multipoint-to-multipoint virtual connections might be implemented with the minimum of network resources. This problem can be considered as a bicriteria Steiner tree problem for which a tabu search resolution approach is herein proposed and evaluated.**

I. Introduction

The establishment of multipoint-to-multipoint virtual connections, for instance in a MPLS-TP network, may involve a network management system for the computation of the necessary resources, among the network resources with the available bandwidth to fulfill the Service Level Agreement (SLA) specified for each virtual connection, in accordance with some management optimality criterion. If the computed connection was obtained with the purpose of creating the minimal connectivity between the end nodes, then the obtained connection must be a tree. The problem of finding this tree may be formulated as a Steiner problem in graphs. In the Steiner problem the objective is to find the minimal tree, in terms of the addition of the links costs, to allow the interconnection of a subset of network nodes, named terminal nodes. Eventually some additional nodes, named Steiner nodes, must also to be added into the solution. This problem is NP–complete [1] and several heuristics and meta-heuristics have been presented to solve it as, for instance, in [2], [3], [4] and [5], [6], respectively.

In previous work a bicriteria formulation of the Steiner tree problem was proposed, where a second objective function was added with the purpose of obtaining not only a minimal additive cost tree, as in the classical Steiner problem, but also a tree with a minimum number of arcs. This bicriteria formulation was motivated by practical reasons in the context of transport communication networks where the cost associate with each arc was an additive load cost.

From the network management perspective this bicriteria formulation may allow the choice between a tree with higher cost but with less links and a tree with lower cost but with a larger number of links. Note that if a load cost for the links is used, such as the one proposed in [7], trees with smaller cost values have, on average, less loaded links.

It should be noted that it is possible to find a tree with the lowest cost and simultaneously with the less number of links. In this case exists a unique optimal solution for the problem because there is no conflict between the objective functions. However, in general there is some degree of conflict between the objective functions considered and, instead of a single optimal solution, a set of Pareto optimal solutions, also designated as non-dominated solutions, are obtained. The set of non-dominated solutions is the set of solutions for which there are no other solutions that improve the value of one objective function without degradation of the value of, at least, one of the others objective functions. It should be noted that as the Steiner problem in graphs (single criterion) is NP-complete, with no polynomial time algorithm to solve it in an exact way, the bi-criterion problem is even more complex. As a consequence the obtained Pareto set may be only an approximation to this set. Good surveys for multiobjective optimization are presented in [8], [9], [10].

In [11], [12] two heuristic approaches for this bicriteria problem were presented, based on the the creation of a new fully meshed graph constituted by the set of the terminal nodes and each edge of this new graph corresponds to a path in the original graph between each pair of terminal nodes. This approach was based on the heuristic of Kou et al. [2] and the main reason to use it was the application of exact bicriteria spanning trees algorithms in the obtained graphs. Each spanning tree found in the new graph is then transferred to the original graph, in which it may not be a tree because it may have cycles, requiring an additional procedure for the elimination of those cycles in order that a tree might be obtained.

In this paper a tabu search approach for finding an approximation of the Pareto set for the previously mentioned problem is presented and its performance is compared with the previously proposed heuristics for the same bicriteria problem. Given the complexity of problem under consideration it was decided to begin with the implementation of an efficient tabu search meta-heuristic proposed in [5] to solve the single criterion problem and later extend the basic approach to the bicriteria case.

This paper is organized as follows. In section II the previous work related with this bicriteria Steiner problem is outlined in order to compare the previously obtained results with the results obtained with the new bicriteria tabu search approach. In section III the bicriteria tabu search procedure is presented together with the heuristics used to obtain the initial solutions. The results analysis and the comparison with previous results obtained for this problem are presented in section V. Finally in the last section the main conclusions are presented.

II. Previous work

The network can be represented as an undirected graph $G(N, E)$, where N is a set of nodes $1, 2, \ldots, |N|$, and E is the set of arcs (i, j) (or links) connecting the node i to the node j, with a cost function $c : E \rightarrow \mathbb{R}$ that assigns a cost $c(i, j) \in \mathbb{R}$ to each arc $(i, j) \in E$. The Steiner's tree problem in graphs is defined as the problem of finding the shortest tree T^* spanning all nodes in the set of terminal nodes S, with $S \subset N$, and possibly some Steiner nodes in $N \setminus S$.

In the bicriteria formulation of the Steiner problem there are two costs $\left(c(i, j)^1, c(i, j)^2\right)$ associated with each arc of the graph, being the second one the unitary cost. The problem consists of finding the set of trees $ND = \left\{T, T^2, \ldots T^n\right\} \subset \mathcal{T}$, where $\mathcal{T}$ is the set of all trees in G that may interconnect the sub-set of terminal nodes, which corresponds to the set of non-dominated solutions. The cost of the tree $c(T) = (c(T)^1, c(T)^2)$ is represented as a vector-valued function, such that

$$c(T)^l = \sum_{i,j:(i,j)\in T} c(i, j)^l, \quad l = 1, 2 \tag{1}$$

The set of non-dominated solutions is the set of trees $ND \subset \mathcal{T}$ such that, for every $T \in ND$, $T^{'} \in \mathcal{T}$, $c(T)^l \leq c(T')^l$, with $l = 1, 2$, $T \neq T'$ and $c(T)^l < c(T')^l$ for, at least, one value of l.

As stated before, in previous work two heuristic approaches were proposed for this bicriteria problem, based on the Kou et al. [2] heuristic for the Steiner problem (single criterion). The heuristic of Kou et al. is based on the creation of a new graph between all pairs of terminal nodes. In this new graph the Steiner problem is transform in the computation of a spanning tree for which there are exact polynomial time algorithms such as Kruskal, Prim or Sollin's algorithms [13]. This heuristic can be described through the following steps:

- Step 1: Construct the complete undirected graph $G_1(S, E_1)$ such that E_1 is a set of arcs between each pair of nodes in S and so $|E_1| = |S|(|S| - 1)/2$. The arc $(i, j) \in E_1$ corresponds to the shortest path $p_{i,j}$ in the graph G calculated using the Dijkstra algorithm and the cost of arc (i, j) in graph G_1 is given by $c(p_{i,j}) = \sum_{(k,l)\in p_{i,j}} c(k, l)$;
- Step 2: Find the minimum spanning tree T_1 of G_1, for instance with Kruskal's or Prim's algorithm;
- Step 3: Construct the sub-graph $G_{S'}$ of G by replacing each arc in T_1 by its corresponding shortest path in G;
- Step 4: Find the new minimum spanning tree $T_{S'}$ by removing the cycles in $G_{S'}$;
- Step 5: Construct T_S from $T_{S'}$ by removing unnecessary arcs in order that all leaves in the tree are terminal nodes.

This basic heuristic (single criterion) was then extended for solving the bicriteria problem also because there are exact algorithms to compute bicriteria spanning trees. The number of non-dominated spanning trees in a graph is exponentially large in the number of nodes with a maximum number of $|N|^{|N|-2}$ trees therefore the bicriteria minimum spanning tree problem is intractable and NP-hard [14]. Nevertheless, the number of supported efficient spanning trees, that correspond to breakpoints of the non-dominated frontier, is polynomially bounded by $|E|^2$ for the bicriteria case and can be efficiently computed using the weighted sum method proposed in [15] .

In the first heuristic approach two heuristics were proposed, named as Bicriteria Steiner Tree Probem (1-1), BCSTP (1-1), and BCSTP (2-10), the last one uses up to 10 parallel arcs between each pair of nodes, computed using a k-shortest path algorithm [16]. In the second heuristic approach, named BCSTP, there are only two parallel arcs between each pair of nodes computed using only Dijskstra's algorithm. The comparison of the results obtained by these heuristics with the new results, obtained with bicriteria tabu search approach, are presented in section V.

III. A tabu search approach

After a previous analysis of some heuristics and meta-heuristics to solve the classical Steiner problem in graphs it has been concluded that the Tabu Search approach proposed in [5] is one of the most efficient approaches, and it was chosen to be the base of the new tabu search heuristic for the bicriteria problem.

The tabu search meta-heuristic for the bicriteria problem is based, as in [5] on a set of initial solutions that were calculated through Takahashi and Matsuyama heuristic [4] with some additional modifications proposed by [17] and [18], furthermore the Takahashi and Matsuyama heuristic was modified in order to compute lexicographical solution, as explained next. The Takahashi and Matsuyama heuristic starts the construction of the Steiner tree selecting a terminal node and then connecting it to its closest terminal node. After that, the closest node to the sub-tree already obtained is connected next. The heuristic ends when all the terminal nodes are connected to the tree. The closest node in each iteration of the heuristic is computed using Dijskstra algorithm. Considering the bicriteria problem this heuristic was modified to be more adequate to the bicriteria problem and a lexicographical version of the Takahashi and Matsuyama heuristic is proposed in algorithm 1, where N_{ab} is the node set, with the exception of node a, in the path p_{ab}. The heuristic uses a heap for a more efficient implementation. The main difference, regarding the original heuristic, is that, in each heuristic step, the two costs of each arc are considered and, if there are two nodes at an equal distance to the nodes already in the tree, it chooses to connect the node with the lower second cost. More details about lexicographical approaches can be seen in [8].

```
P_set = ∅ // Set of paths
J = S − {s};
T ← (s, 0, 0);
P_{s,J} ← Dijkstra(s, J);
create_heap(H);
for j ∈ J do
    insert_heap(p_sj, H);
end
while J ≠ ∅ do
    find − min_heap(p_ab, H);
    delete − min_heap(p_ab, H);
    if b ∈ J then
        indice = 1;
        p_vy ← p_ab;
        while indice > 0 do
            indice = 0;
            if H ≠ ∅ then
                find − min_heap(p_zx, H);
                delete − min_heap(p_zx, H);
                if c(p_vy)^l1 = c(p_zx)^l1 then
                    indice = 1;
                    if c(p_vy)^l2 > c(p_zx)^l2 then
                        P_set ← P_set ∪ {p_vy};
                        p_vy ← p_zx;
                    end
                    else
                        P_set ← P_set ∪ {p_zx} ;
                    end
                end
                else
                    P_set ← P_set ∪ {p_zx} ;
                end
            end
        end
        insert_heap(P_set, H) ;
        P_set = ∅;
        J = J − {y};
        for each n ∈ N_vy do
            P_nJ = Dijkstra(n, J);
            for j ∈ J do
                insert_heap(p_nj, H);
            end
        end
        T = T ∪ p_vy;
    end
end
```

Algorithm 1: Takahashi and Matsuyama lexicographic algorithm.

The main bicriteria-tabu procedure is based, as in [5], on the basic operations of adding or removing a given Steiner node from a current solution as explain next. In order to improve the search in the solutions space, the adding or removing a given node may be forbidden, or tabu, for some iterations during the bicriteria-tabu procedure. Also, in order to improve even more the search of the solution space, two different diversifications were considered: the continuous diversification proposed in [5] and a new bicriteria diversification in order to better exploit the solution space in the bicriteria sense. As such the $penalty(j)$ function, mentioned in the algorithm 2, is part of the continuous diversification and consists of a positive or negative cost that is added to the cost of the current solution, as described in [5] .

The basic tabu procedure for the bicriteria Steiner problem is presented in the algorithm 2, where $D(T) \in ND$ is the set of solutions in ND that became dominated by a new T solution. The procedure $adding\_or\_removing(j, T)$ is the cost of a temporary tree which results from adding or removing the node j from the current tree T and $tabu(j)$ is the iteration from which node j can be added or removed from the tree (left to be tabu). The function $random(x, y)$ generates a random number between x and y, where x and y depend on the number of nodes in the graph, as in [5] . Note that this basic bicriteria-tabu procedure is described for each metric l, with $l = 1, 2$, separately.

```
iteration = 0;
while iteration < 20 do
    P = ∞;
    w = −1 ;
    for each Steiner node j do
        T' ← adding_or_removing(j, T) ;
        if ((c(T')^l + penalty(j) < P) and
        (tabu(j) ≤ iteration or c(T')^l < c(T*)^l)) then
            P = c(T')^l;
            w = j;
        end
        if true = non_dominated(T') then
            ND ← ND{T'} \ D(T');
        end
    end
    T = adding_or_removing(w, T);
    tabu(w) = iteration + random(x, y) + 1;
    if c(T)^l < c(T*)^l then
        T* = T;
        iteration = −1;
    end
    iteration = iteration + 1;
end
```

Algorithm 2: Bicriteria-tabu search procedure.

The paths diversification proposed in [5] leads to some improvement of some solutions but the overall quality of the solutions is much more dependent of the initial solutions than of this diversification technique. Because of that a new diversification is considered based on the calculation of the set of initial solutions. Three different ways of computing the initial solutions were considered using three different costs: the links cost, the hop count and a linear combination of both metrics in Takahashi and Matsuyama heuristic and in the lexicographical adaptation of this heuristic. A linear combination of both metrics, similarly to the weighted sum method for the computation of the supported non-dominated solutions, is a way to find initial solutions that are close to the Pareto set but in another region. Note that the Takahashi and Matsuyama heuristic cannot guarantee the finding of optimal solutions. The three sets of initial solutions are then used by the Bicriteria-tabu search procedure in all possible combination. For instance, an initial solution found with the hop count in the Takahashi and Matsuyama lexicographical heuristic is then used by the bicriteria-tabu search procedure in two different

ways: searching a better solution in terms of hop count and also searching a better solution in terms of links' costs. This diversification technique may be improved but the results obtained with it are already good.

IV. Results analysis

The test networks used for the performance evaluation of the bicriteria tabu search meta-heuristic are from Steinlib Testdata Library [19]. The sets of networks that were used are the B, C, I80 and I640, because they have been used already to test the BCSTP (1-1), BCSTP (2-10) and BCSTP heuristics. Note that in [5] C netwks ser was also used.

The graph reductions presented in [5] for the single criterion Steiner problem were adapted to this bicriteria case. Thus, the reductions were made as follows, considering the nodes in the graph i, j and l :

1) If i has degree 1 and edge (i, j) exists, then both the edge and the node i can be removed from the graph. If i is a terminal node then node j becomes a terminal node, if it is not, and the node i and the edge (i, j) are added to the solution.
2) For i Steiner node of degree 2 with the incident edges (j, i), (i, l) and existing the arc (j, l), then i, (j, i) and (i, l) are removed from the graph if $c(j, l) \leq (c(j, i) + c(i, l))$.
3) Let i be a terminal node and j and l the 2 neighboring nodes closer to i, respectively, with j terminal node. If $c(i, j) \leq c(i, l)$, then the edge (i, j) is part of the solution and the graph can be contracted along it. It means that the node j is removed from the graph, the edges incident in j will became incident in i and in the case of parallel incident edges at i, the edge with higher cost will be removed. If l is also a terminal node, the condition change to $c(i, j) < c(i, l)$.

In Table I are represented the results of the reductions applied to the sets of networks B and C. The results of the reductions for networks I80 and I640 are not presented due to the large number of networks belonging to each set that cannot be easily presented. However, these networks were also reduced in order to obtain the results presented below.

The results are shown in compact form for each of the networks sets. The results for the first metric are compared with the optimum value, which is presented in [19]. For the hop count metric the optimum value is not known. The results are thus compared with the minimum result obtained by all of the four algorithms. The results are presented as a percentage of error related to the optimal or minimum value, with the error being the difference between the results obtained and the optimal or the minimum value.

In figure 1 the percentage of optimal results obtained by each algorithm for the first metric is represented. In figure 2 the average deviation from the optimum value is presented and in Figure 3 the maximum deviation is also presented.

The results obtained by BSTP-Tabu are much better than the results obtained by the other algorithms. Only for the set of I640 networks the percentage of optimal solutions is low. Still, it is 5% higher than for the BCSTP algorithm, which is the best algorithm without considering BSTP-Tabu. In the other sets the difference is always above 20 $\%$. In C networks the BSTP-Tabu obtains 75% of optimal values, in B 100% and in I80 almost 100%.

	Original graphs			Graphs reducted		
Problem	—N—	—S—	—A—	—N—	—S—	—A—
C1	500	5	625	307	5	432
C2	500	10	625	305	8	428
C3	500	83	625	281	57	404
C4	500	125	625	277	70	402
C5	500	250	625	197	88	319
C6	500	5	1000	470	5	969
C7	500	10	1000	460	9	960
C8	500	83	1000	451	65	950
C9	500	125	1000	446	96	945
C10	500	250	1000	364	136	859
C11	500	5	2500	500	5	2500
C12	500	10	2500	498	9	2498
C13	500	83	2500	485	69	2480
C14	500	125	2500	470	95	2466
C15	500	250	2500	389	139	2343
C16	500	5	12500	500	5	12500
C17	500	10	12500	498	8	12485
C18	500	83	12500	482	65	12363
C19	500	125	12500	446	71	11875
C20	500	250	12500	338	88	9403
B1	50	9	63	18	4	26
B2	50	13	63	32	8	44
B3	50	25	63	21	8	32
B4	50	9	100	45	8	95
B5	50	13	100	42	11	91
B6	50	25	100	38	16	84
B7	75	13	94	46	11	64
B8	75	19	94	39	12	56
B9	75	38	94	26	11	43
B10	75	13	150	67	11	142
B11	75	19	150	66	15	140
B12	75	38	150	51	18	121
B13	100	17	125	59	12	83
B14	100	25	125	54	15	79
B15	100	50	125	42	19	62
B16	100	17	200	89	13	188
B17	100	25	200	90	22	190
B18	100	50	200	66	25	158

TABLE I. Bi-criterion graph reductions for B and C networks sets.

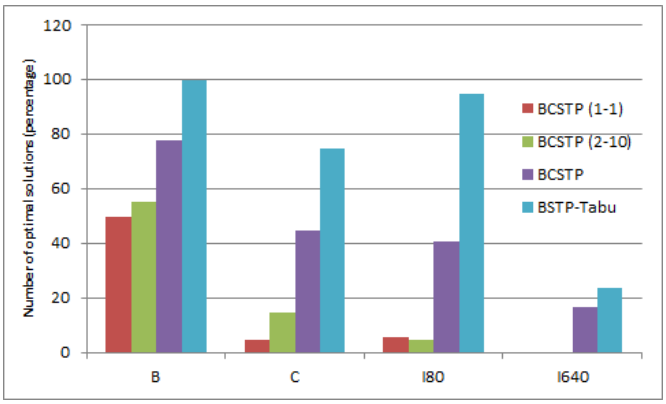

Fig. 1. Percentage of optimal solutions.

The average deviation from the optimal for BSTP-Tabu is almost null in all sets, even for I640 networks. The difference for the other algorithms is very large. Only for the maximum deviation from the optimum the results are similar for both BSTP-Tabu and BCSTP. Yet the difference between these algorithms is high, except in C networks where the results are similar for both algorithms. However, the small average deviation shows that the cases that lead to higher values for

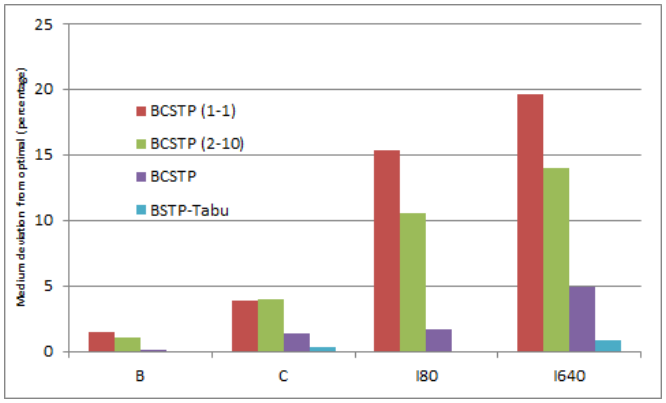

Fig. 2. Average relative deviation from the optimum.

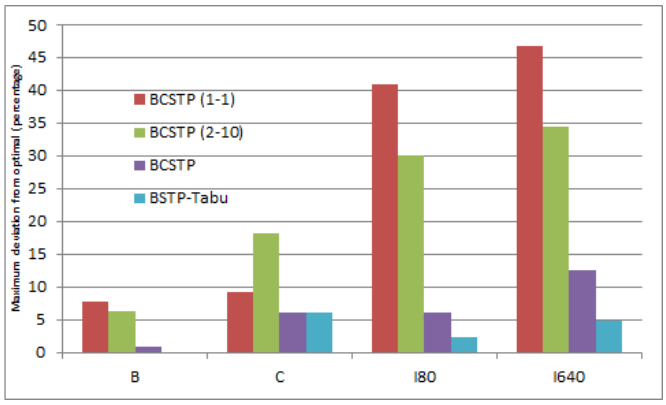

Fig. 3. Maximum relative deviation from the optimum.

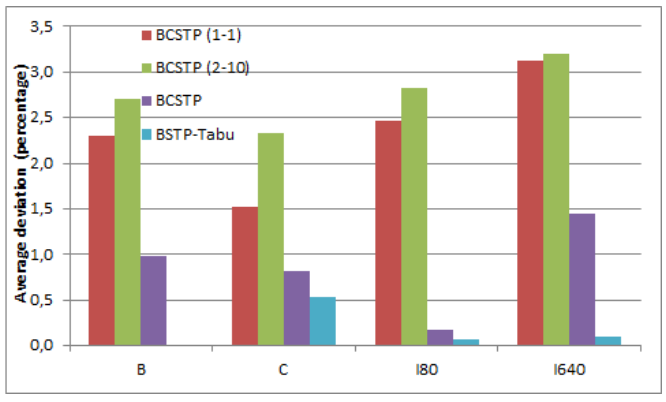

Fig. 4. Average relative deviation from the minimum obtained hop count.

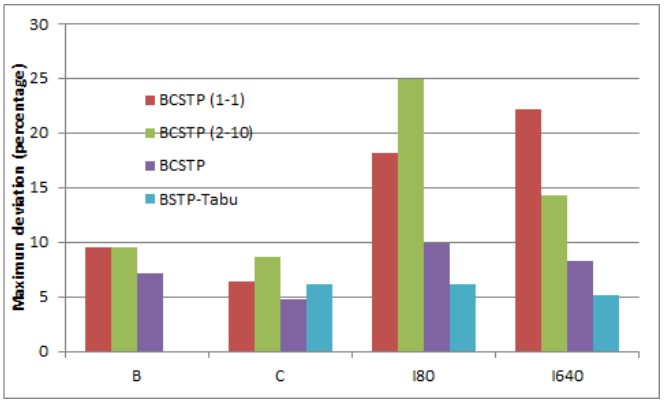

Fig. 5. Maximum relative deviation from the minimum obtained hop count.

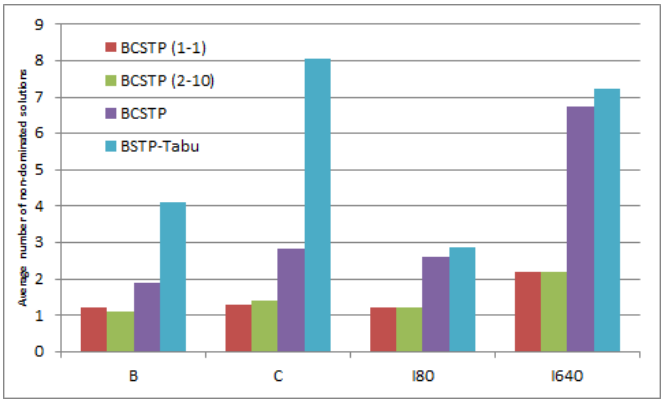

Fig. 6. Average number of non-dominated solutions.

the maximum deviation are a small number.

In figures 4 and 5 the average and the maximum deviations for the minimum hop count are presented, respectively.

For this metric the results obtained by BSTP-Tabu are again very good as compared with the results obtained by the other heuristics. Looking at figure 4 we note that the average deviation obtained by BSTP-Tabu is almost non-existent because it is the algorithm that gets almost all of the minimum values for this metric, except for C networks. The worst results are obtained for those networks. There is only one case for which the BSTP-Tabu is not the best algorithm. It is the maximum deviation to the minimum of the hop count for C networks. In that case the difference is between 25 and 23 edges, being the later value obtained by BCSTP algorithm. But that difference corresponds to a relatively high maximum deviation that makes an increasing of the average deviation for the C networks, although 18 of the 20 minimum values for this metric are obtained with BSTP-Tabu for this set of networks.

In figure 6 the average number of non-dominated solutions obtained by each algorithm for each set of networks is presented.

The BSTP-Tabu algorithm is also the algorithm that finds the higher number of non-dominated solutions. Being the algorithm that obtains the best results for the two metrics and that finds more non-dominated solutions, we can say that BSTP-Tabu algorithm obtains the best approximation of the Pareto set.

V. Conclusion

In order to be applied in the context of communication transport networks for the computation of multipoint-to-multipoint virtual connections, a bicriteria Steiner tree problem was previously formulated. A tabu search approach to solve this bicriteria Steiner problems in graphs was herein present. This new approach for the considered bicriteria problem was

initially based on an extension a previously presented tabu search meta-heuristic which presents very good results for the classical Steiner problem in graphs. However the bicriteria nature of the problem has led to the development of an extension of the classical meta-heuristic in order to improve the quality of the solutions found for this new problem with higher complexity. Besides the consideration of a set of non-dominated solution all over the meta-heuristic, instead of a single optimal solution, other improvements were considered. The first is related with the reductions that can be done in the graph in this bicriteria case for the reduction of the problem size an the other is related with the initial set of solutions for which a lexicographic version of a classical heuristic was developed. The diversification based on the initial solution is an important aspect for the bicriteria problem. This aspect can be even improved as well as other forms of diversification can also be integrated for treating higher dimensions problems.

The performance analysis of the proposed meta-heuristic shows that this approach can find very good solutions regarding both metrics as compared with previously developed heuristics.

ACKNOWLEDGMENT

This work has been partially supported by project QREN 23301 PANORAMA II, co-financed by European Union's FEDER through "Programa Operacional Factores de Competitividade" (POFC) of QREN (FCOMP-01-0202-FEDER-023301) and by the Portuguese Foundation for Science and Technology under project grant PEst-OE/EEI/UI308/2014.

REFERENCES

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability – A Guide to the Theory of NP-Completeness*, V. Klee, Ed. Bell Telephone Laboratories, Incorporated, 1979.

[2] L. Kou, G. Markowsky, and L. Berman, "A fast algorithm for steiner trees," *Acta Informatica*, vol. 15, no. 2, pp. 141–145, June 1981.

[3] K. Mehlhorn, "A faster approximation algorithm for the steiner problem in graphs," *Information Processing Letters*, vol. 27, no. 3, pp. 125 – 128, 1988.

[4] A. M. H. Takahashi, "An approximate solution for the steiner problem in graphs," *Math. Japon*, vol. 24, pp. 573–577, 1980.

[5] M. Gendreau, J. F. Larochelle, and B. Sansó, "A tabu search heuristic for the steiner tree problem," *Networks*, vol. 34, no. 2, pp. 162–172, 1999.

[6] H. Esbensen, "Computing near-optimal solutions to the steiner problem in a graph using a genetic algorithm," *Networks*, vol. 26, pp. 173–185, 1995.

[7] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 4, pp. 756 – 767, May 2002.

[8] R. Marler and J. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[9] J. Clímaco and M. Pascoal, "Multicriteria path and tree problems: discussion on exact algorithms and applications," *International Transactions in Operational Research*, vol. 19, no. 1-2, pp. 63–98, 2012.

[10] J. Clímaco, J. Craveirinha, and M. Pascoal, "Multicriteria routing models in telecommunication networks – overview and a case study," in *Advances in multiple criteria decision making and human systems management: knowledge and wisdom*, Y. Shi, D. Olson, and A. Stam, Eds. IOS Press, 2007, vol. edited in honor of Milan Zeleny, ch. 1, pp. 17–46.

[11] L. Martins, N. Ferreira, and J. Craveirinha, "A bi-criteria algorithm for multipoint-to-multipoint virtual connections in transport networks," in *CNSM 2011, 7th International Conference on Network and Service Management*, Paris, Oct. 2011, pp. 1–5.

[12] L. Martins and N. G. Ferreira, "A bi-criteria approach for steiner's tree problems in communication networks," in *Proceedings of the 2011 International Workshop on Modeling, Analysis, and Control of Complex Networks – Cnet '11*. San Francisco, California: International Teletraffic Congress, 2011, pp. 37–44.

[13] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows - Theory, Algorithms and Applications*. Prentice-Hall, Inc., 1993.

[14] S. Ruzika and H. W. Hamacher, "A survey on multiple objective minimum spanning tree problems," in *Algorithmics of Large and Complex Networks*, ser. Lecture Notes in Computer Science, J. Lerner, D. Wagner, and K. A. Zweig, Eds., vol. 5515. Springer, 2009, pp. 104–116.

[15] H. Hamacher and G. Ruhe, "On spanning tree problems with multiple objectives," *Annals of Operations Research*, vol. 52, pp. 209–230, 1994.

[16] E. Q. V. Martins, M. M. B. Pascoal, and J. L. E. Santos, "Deviation algorithms for ranking shortest paths," *International Journal of Foundations of Computer Science*, vol. 10, pp. 247–263, 1999.

[17] P. Winter and J. M. Smith, "Path-distance heuristics for the steiner problem in undirected networks," *Algorithmica*, vol. 7, pp. 309–327, 1992.

[18] V. Rayward-Smith and A. Clare, "The computation of nearly minimal on finding steiner vertices," *Networks*, vol. 16, no. 283–294, 1986.

[19] T. Koch, A. Martin, and S. Voß, "SteinLib: An updated library on steiner tree problems in graphs," Konrad-Zuse-Zentrum für Informationstechnik Berlin, http://elib.zib.de/steinlib, Tech. Rep. ZIB-Report 00-37, 2000.