ALGORITMOS PARA O PROBLEMA DA ÁRVORE GERADORA MÍNIMA PROBABILÍSTICA

RAFAEL FERREIRA BARRA DE SOUZA

ALGORITMOS PARA O PROBLEMA DA ÁRVORE GERADORA MÍNIMA PROBABILÍSTICA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Sebastián Alberto Urrutia

Belo Horizonte

Maio de 2010

Souza, Rafael Ferreira Barra de

S729a Algoritmos para o problema da árvore geradora mínima probabilística / Rafael Ferreira Barra de Souza. — Belo Horizonte, 2010
xxiv, 49 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais. Departamento de Ciência da Computação.

Orientador: Prof. Sebastián Alberto Urrutia

1. Teoria dos Grafos - Tese. 2. Heurísticas - Tese. 3. Otimização a priori - Tese. 4. Árvore Geradora Mínima Probabilística - Tese. 5. Programação Inteira - Tese.
I. Orientador. II. Título.

CDU 519.6*62

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Algoritmos para o problema da árvore geradora mínima probabilística

RAFAEL FERREIRA BARRA DE SOUZA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. SEBASTIÁN ALBERTO URRUTIA - Orientador
Departamento de Ciência da Computação - UFMG

PROFA. SIMONE DE LIMA MARTINS
Departamento de Ciência da Computação - UFF

PROF. GERALDO ROBSON MATEUS
Departamento de Ciência da Computação - UFMG

PROF. MAURÍCIO CARDOSO DE SOUZA
Departamento de Engenharia de Produção - EE- UFMG

Belo Horizonte, 31 de maio de 2010.

Ao meu avô Luiz, meu maior incentivador e entusiasta.

Agradecimentos

É uma satisfação imensa concluir este projeto e perceber que muitos contribuíram para que eu chegasse até aqui. Tenho muita gratidão por todos aqueles que doaram parte de seu tempo, fosse influenciando diretamente o meu trabalho, fosse para me incentivar nesses dias tão atarefados dos últimos dois anos. Ter muitas pessoas para agradecer é sinal de que não estive sozinho em momento algum e é mais um motivo de certeza de que tudo valeu a pena.

Agradeço em primeiro lugar a Deus, por mais uma conquista. Por me dar saúde e disposição para viver e permitir que eu me apaixone por tudo que faço.

Esse trabalho não teria sequer começado se não fosse a dedicação dos meus avós, Luiz e Elaine. Dedico este trabalho à luta dos dois em sempre cuidar de mim e me apoiar em tudo na vida. Agradeço à minha mãe, Eliane, pelo amor em todos os momentos, por mais apertados que tenham sido, do início ao fim da dissertação. Agradeço ao meu pai, Osvaldo, e à sua esposa Astrid, pelas orações e por toda a motivação. Agradeço também o carinho de minha tia Dori. Sua presença no dia da defesa teve tamanho significado que não sou capaz de descrever. Agradeço à Juliana, minha namorada, por todo o amor e companheirismo durante o mestrado. Obrigado por me animar, ficar acordada ao meu lado por diversas noites e me ajudar a acreditar que esse trabalho seria concluído com sucesso. A sua revisão do texto foi fundamental. Agradeço à minha sogra Maria José por sempre torcer por mim e participar dessa etapa tão importante.

Agradeço aos professores e funcionários do departamento de Ciência da Computação da Universidade Federal de Minas Gerais pela competência e compromisso em buscar a excelência do Programa de Pós-Graduação. Agradeço ao meu orientador, professor Sebastián, por toda a atenção e profissionalismo. Sua compreensão foi fundamental para orientar alguém que precisava dividir o tempo entre a pesquisa e o trabalho. A minha admiração pelas suas pesquisas só aumentou desde que começamos a trabalhar juntos. Foi bastante desafiador trabalhar com um problema tão interessante e ter a oportunidade de aprender tanto. Aos professores da banca examinadora, agradeço por terem aceitado o convite de participar da minha defesa e por terem contribuído

com sugestões valiosas, ajudando a enriquecer este trabalho. Agradeço especialmente ao professor Robson pelas diversas orientações casuais e também pela confiança no meu trabalho profissional, desenvolvido no Synergia.

Aos meus colegas do Laboratório de Pesquisa Operacional (LaPO), agradeço pelos momentos de trabalho intenso, no dia a dia, na dissertação e nos congressos, sempre muito produtivos e ao mesmo tempo alegres, descontraídos. Agradeço aos meus amigos do Synergia por todo o apoio, em todos os instantes. Ter tantas pessoas legais por perto e sempre dispostas a ajudar é um privilégio. Agradeço especialmente aos amigos que contribuíram bastante na minha vida acadêmica. Dárlinton, por ter me apresentado a um orientador tão bom. Adriano Pereira, pela pesquisa interessantíssima que desenvolvemos juntos. Renato Cunha, por todo o auxílio com LaTeX. Alair, pela ajuda na prévia da apresentação.

Agradeço a tantas outras pessoas, vocês sabem quem são, pela amizade e pelo companheirismo, que ajudaram a tornar essa conquista mais brilhante e amenizar as dificuldades ao longo do desenvolvimento deste trabalho.

"Nunca ande pelo caminho traçado,
pois ele conduz somente até onde os outros já foram."
(Alexander Graham Bell)

Resumo

O Problema da Árvore Geradora Mínima Probabilística é uma generalização do problema clássico da Árvore Geradora Mínima em que se considera a situação na qual nem todos os nós estão deterministicamente presentes, mas estão presentes conforme uma determinada probabilidade.

Dado um grafo, $G = (V, E)$, que possui um custo associado a cada aresta em E e uma probabilidade de cada vértice em V estar ativo, pretende-se construir uma árvore geradora T em G *a priori*, tal que o custo esperado de T para um momento futuro seja mínimo. Este problema é provado como NP-Difícil em seu caso geral.

Nesta dissertação, a versão homogênea do problema, situação em que todos os nós têm a mesma probabilidade de estarem ativos, é descrita, analisada e resolvida através de algoritmos de busca local. Apresenta-se uma heurística construtiva capaz de encontrar soluções viáveis para o problema. A partir de uma técnica que avalia os custos de soluções vizinhas de maneira eficiente, propõe-se a incorporação de algoritmos de busca local a um algoritmo de Busca Tabu, capaz de gerar soluções de melhor qualidade para o problema. Propõe-se, também, uma modelagem que permite a resolução do problema por Programação Inteira. A análise dos resultados revela que os algoritmos, quando comparados à resolução do modelo exato, mostraram ser uma ferramenta bastante eficiente para lidar com um problema computacionalmente difícil.

Palavras-chave: Teoria dos Grafos, Heurísticas, Otimização *a priori*, Árvore Geradora Mínima Probabilística, Programação Inteira.

Abstract

The Probabilistic Minimum Spanning Tree Problem is a generalization of the classical Minimum Spanning Tree problem, addressing the assumption that arise when not all nodes are deterministically present but, rather, nodes are active with known probabilities.

Given a graph, $G = (V, E)$, where there is a cost associated with every edge in E and a probability of each node in V to be active, the objective is to build a sub-tree T in G *a priori*, where the expected cost of T is minimum. This problem is proved to be NP-Hard in the general case.

In this dissertation, the homogeneous case of the problem, when all nodes have the same probability of being active, is described, analyzed and solved through local search algorithms. A constructive heuristic is proposed in order to find feasible solutions for the problem. Starting through a technique that efficiently evaluates the costs of neighboring solutions, it is proposed the embedding of local search algorithms into a Tabu Search metaheuristic, capable of yielding better quality solutions for the problem. It is also proposed a model that can be solved through Integer Programming. The analysis of the results shows that the algorithms, when compared to the resolution of the exact model, proved to be an efficient tool to deal with a computationally difficult problem.

Keywords: Graph Theory, Heuristics, *A priori* Optimization, Probabilistic Minimum Spanning Tree Problem, Integer Programming.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Sumário

Capítulo 1

Introdução

A Otimização *a priori* representa um subconjunto dos problemas de Otimização Combinatória caracterizados pela presença de elementos probabilísticos em sua definição. Esses problemas são chamados de Problemas Probabilísticos de Otimização Combinatória (PCOPs[1]) [Bertsimas et al., 1989]. Dentre as diversas motivações para se trabalhar com esse tipo de problema, destaca-se a adequação de modelos mais realistas a aplicações práticas, em que a aleatoriedade é um dos conceitos principais e não somente uma característica do cenário estudado. Há, também, o interesse em se investigar a robustez de soluções ótimas para problemas determinísticos quando as instâncias de problemas já resolvidos são modificadas. Esse aspecto é, talvez, um dos mais importantes da otimização *a priori*, por caracterizar um cenário em que se tem uma instância já resolvida e é necessário que se resolva novamente instâncias do mesmo problema, que são apenas uma variação da instância original. Ao invés de se tentar reotimizar cada instância provável, procura-se encontrar uma solução *a priori* para o problema original, para que tal solução seja facilmente adaptável a cada variação aplicada ou atenda satisfatoriamente, em média, cada variação em particular.

Problemas de otimização *a priori* podem ser utilizados para modelar diversas aplicações práticas, como nas áreas de roteamento, desenho de circuitos integrados, logística e telecomunicações. Exemplos de problemas encontrados na literatura que utilizam tal estratégia de otimização são o Problema do Caixeiro Viajante Probabilístico [Jaillet, 1985], o Problema de Roteamento de Veículos Probabilístico [Jaillet & Odoni, 1988] e o Problema da Árvore Geradora Mínima Probabilística [Bertsimas, 1988a] sendo, este último, foco do estudo deste trabalho.

[1]Do inglês *Probabilistic Combinatorial Optimization Problems*.

1.1 Contextualização

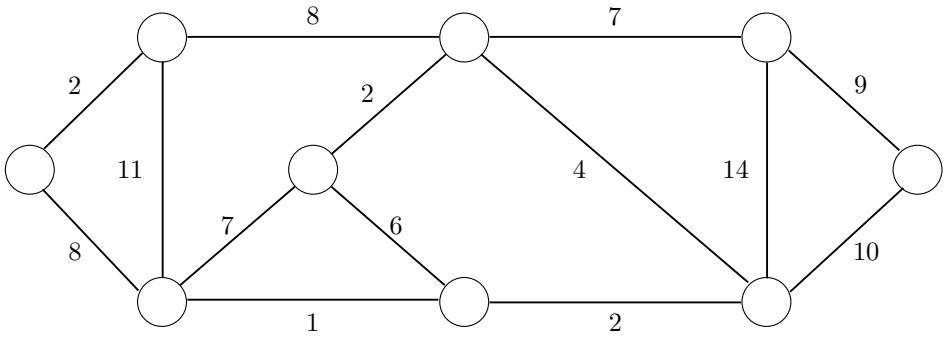
Alguns problemas de otimização combinatória podem ser formulados através de um tipo de grafo específico conhecido como árvore. Uma árvore é um grafo conexo (existe caminho entre qualquer par de seus vértices) e acíclico (que não possui ciclos). Maiores detalhes sobre Teoria dos Grafos, árvores e suas aplicações podem ser encontradas na literatura, como em Bondy & Murty [1976].

Dado um grafo conexo não direcionado, uma árvore geradora desse grafo é um sub-grafo representado por uma árvore que conecta todos os seus vértices. Um único grafo pode possuir várias árvores geradoras. É possível, ainda, associar um custo a cada aresta do grafo, determinando-se o custo de uma árvore geradora através da soma dos custos das arestas que possui. Eis que surge o problema da Árvore Geradora Mínima (MST[2]), que consiste em se encontrar uma árvore geradora de custo mínimo em um grafo conexo através de arestas com custos para conectar seus vértices. A árvore encontrada é aquela com o menor custo possível dentre todas as árvores geradoras possíveis para um determinado grafo.

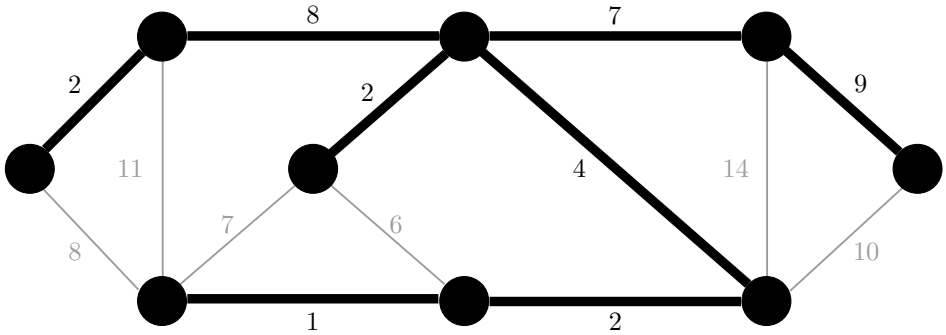
Um exemplo simples de aplicação para uma árvore geradora mínima seria uma empresa de televisão a cabo com interesse em realizar o cabeamento de uma vizinhança de clientes. Dentre as diversas possibilidades de se conectar uma residência a outra, alguns caminhos podem ser de custo maior, por necessitarem de cabos de comprimento mais longo, devido a algum tipo de obstáculo que deve ser contornado, por exemplo; esses caminhos podem ser representados por arestas com custos mais elevados. Uma árvore geradora para o grafo que interliga a vizinhança seria um subconjunto dos caminhos possíveis que não possui ciclos, mas garante a conexão entre todas as residências. A árvore geradora mínima que conecta todas as residências seria aquela que garantiria o menor custo possível para realizar todas as conexões. Um grafo conexo e sua respectiva árvore geradora mínima são exemplificados na figura 1.1.

O primeiro algoritmo para se encontrar uma árvore geradora mínima foi desenvolvido pelo cientista Otakar Borüvka em 1926 [Borüvka, 1926], que tinha como objetivo conectar a rede elétrica de Moravia, cidade localizada no leste da República Tcheca. Posteriormente, surgiram outros algoritmos capazes de resolver esse tipo de problema, tal como Kruskal [1956] e Prim [1957], que, assim como Borüvka [1926], resolveram o problema em tempo polinomial. Conforme o aumento e a diversidade da demanda em situações práticas envolvendo árvores geradoras mínimas, foram surgindo variações do problema que consistem do problema fundamental acrescido de restrições. Dentre estas variações, é possível destacar alguns exemplos:

[2]Do inglês *Minimum Spanning Tree*.



(a) Grafo.



(b) Árvore geradora mínima, de custo 35.

Figura 1.1. Grafo conexo e sua respectiva árvore geradora mínima.

- **Árvore geradora mínima com restrição de grau:** proposta em Narula & Ho [1980], consiste em encontrar uma árvore geradora mínima para um grafo em que o maior grau de cada vértice é limitado a um valor constante.
- **Árvore geradora estocástica:** definida em Ishii et al. [1981], pretende encontrar uma árvore geradora de custo mínimo em um grafo onde os custos de suas arestas não são constantes, mas variáveis aleatórias.
- **Árvore geradora mínima quadrática:** proposta por Xu & Gen [1995], almeja encontrar uma árvore geradora mínima para um grafo completo em que os custos das arestas possuem dependência e, portanto, a função que os define é quadrática.
- **Árvore geradora mínima probabilística:** detalhado inicialmente em Bertsimas [1988a]. Dado um grafo com um custo associado a cada aresta e uma probabilidade associada a cada vértice, conhecidos *a priori*, deseja-se construir uma árvore geradora para esse grafo tal que seu custo esperado *a posteriori* seja o menor possível.

As extensões do problema da árvore geradora mínima são modelagens muito importantes na resolução de problemas de otimização combinatória. O foco deste

trabalho é o problema da árvore geradora mínima probabilística, especificado nas seções que se seguem.

1.2 O Problema da Árvore Geradora Mínima Probabilística

O problema da Árvore Geradora Mínima Probabilística (PMST[3]) se aplica em circunstâncias nas quais os nós estão presentes conforme uma determinada probabilidade. Mesmo que alguns casos específicos do problema possam ser resolvidos em tempo polinomial, esse problema é NP-Difícil, como demonstrado no trabalho de Bertsimas [1988b]. O problema se mantém NP-Difícil em duas situações: quando o grafo não é completo e os custos das arestas são iguais, ou se o grafo é completo e os custos podem assumir dois valores distintos. Destaca-se, inclusive, a dificuldade do PMST quando a probabilidade dos vértices é idêntica para todos eles. Por outro lado, sempre que uma árvore geradora mínima possui uma topologia em estrela, o problema da árvore geradora mínima probabilística se reduz ao problema clássico da Árvore Geradora Mínima podendo, então, ser resolvido em tempo polinomial por diversos algoritmos já conhecidos. Em instâncias cujas probabilidades dos nós estarem presentes são muito baixas, conforme Bertsimas [1990], o problema é praticamente equivalente ao problema de Modelagem de Redes, apresentado e provado como sendo NP-Difícil em Garey & Johnson [1979].

Modelado como uma variação do problema clássico da árvore geradora mínima, o custo da árvore no contexto do problema, denominado custo ativo esperado, é definido como a soma dos custos das arestas que conectam nós ativos em determinado momento. O objetivo do PMST é encontrar uma árvore geradora *a priori*, cujo custo ativo esperado seja o menor possível. No PMST são tratados dois diferentes casos das probabilidades dos nós estarem ativos. No primeiro, a probabilidade é a mesma para qualquer nó, sendo denominado homogêneo. No segundo caso, chamado heterogêneo, a probabilidade dos nós varia de forma independente.

1.3 Objetivo do Trabalho

O objetivo deste trabalho é propor algoritmos de busca local para o Problema da Árvore Geradora Mínima Probabilística em seu caso homogêneo, incorporados a uma heurística de Busca Tabu. A exploração do problema de tal forma só é possível devido a uma

[3]De *Probabilistic MST*.

estratégia eficiente de avaliação de custos das soluções vizinhas, quando o algoritmo é executado. Esta talvez seja a maior contribuição deste trabalho, posto que o cálculo do custo ativo esperado da árvore é uma das tarefas com maior custo computacional no âmbito do problema. Propõe-se, também, um modelo em Programação Linear Inteira, tratando de instâncias pequenas devido à sua complexidade. É perceptível, através dos testes com o modelo, a dificuldade em se resolver o problema em sua modelagem exata no que se diz respeito a recursos computacionais.

1.4 Organização da Dissertação

Para atingir os objetivos pretendidos, este trabalho está organizado em cinco capítulos, sendo o primeiro esta introdução. No Capítulo 2 detalha-se o PMST, seguido das motivações para se trabalhar com o problema e uma revisão dos principais trabalhos relacionados. Apresenta-se no Capítulo 3 uma formulação matemática por Programação Linear Inteira para o caso homogêneo do problema, com os testes empíricos e respectivos resultados computacionais. Em seguida, o Capítulo 4 descreve o procedimento de construção de soluções iniciais, seguido dos algoritmos de busca local associados a uma Busca Tabu, juntamente com os testes e resultados encontrados. Finalmente, no Capítulo 5 são apresentadas as principais contribuições deste trabalho, as conclusões e propostas para atividades futuras.

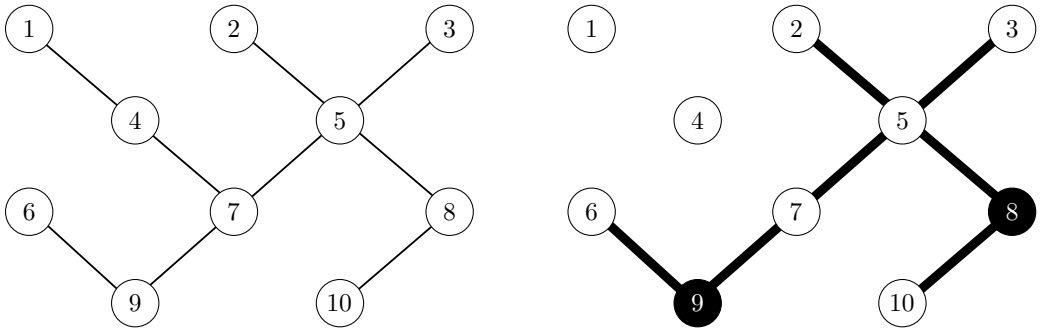
Capítulo 2

O Problema

2.1 Definição

Como forma de ilustrar o problema, pode-se levar em consideração um grafo qualquer com dez nós e uma árvore geradora, T, ilustrada na figura 2.1a. Suponha-se que em um momento futuro, o conjunto de nós presentes será representado por $S = \{2, 3, 5, 6, 7, 10\}$. A partir desse ponto, definimos $T(S)$ como a mínima sub-árvore de T necessária para possibilitar a conexão de todos os nós de S. $T(S)$ é ilustrada na figura 2.1b. As arestas $(1, 4)$ e $(4, 7)$ não estão conectadas em $T(S)$, pois os vértices 1 e 4 não estão ativos. Porém, as arestas $(6, 9)$ e $(7, 9)$, bem com as arestas $(5, 8)$ e $(8, 10)$, permanecem presentes em $T(S)$. Isso acontece pois existem nós ativos, 6 e 10, que dependem dos nós 8 e 9 para estarem conectados à sub-árvore $T(S)$.

O PMST define uma estratégia eficiente para se atualizar soluções de árvore geradora quando instâncias de um problema específico são afetadas probabilisticamente, em resposta à ausência de determinados vértices do grafo. A partir de uma árvore



(a) Árvore Geradora Mínima T. (b) Sub-Árvore $T(S)$.

Figura 2.1. Árvore geradora mínima e sua sub-árvore em um momento futuro.

geradora T conhecida *a priori*, definimos o custo ativo de T na configuração S, $L_T(S)$, como sendo o custo da menor sub-árvore de T que conecta os nós do conjunto S. Na figura 2.1a, se $S = \{2, 3, 5, 6, 7, 10\}$, $L_T(S)$ é o custo da árvore apresentada na figura 2.1b.

Dado um grafo não direcionado $G = (V, E)$, não necessariamente completo, uma função de custo $c : E \to \Re^+$, e uma função de probabilidade $P : 2^V \to [0, 1]$, que mapeia cada subconjunto de nós com uma probabilidade, o objetivo do problema é encontrar uma árvore geradora T, que minimize o custo ativo esperado $E[L_T]$, onde:

$$E[L_T] = \sum_{S \subseteq V} P_S \; L_T(S). \tag{2.1}$$

É importante notar que, nesse nível de generalização, pode-se modelar dependências entre as probabilidades de presença dos conjuntos de vértices. Com essa formulação seria necessário um esforço de $O(2^n)$ multiplicado pelo custo de se calcular $L_T(S)$, onde $n = |V|$, para calcular o custo ativo esperado ($E[L_T]$) de uma árvore T. Se um nó i está ativo com uma probabilidade p_i, sendo a ativação dos nós eventos independentes, $E[L_T]$ para uma árvore geradora mínima é dado pela equação [Bertsimas, 1990]:

$$E[L_T] = \sum_{e \in T} c_e \left\{ 1 - \prod_{i \in K_e} (1 - p_i) \right\} \times \left\{ 1 - \prod_{i \in (T \setminus K_e)} (1 - p_i) \right\}. \tag{2.2}$$

A operação de somatório é feita sobre todas as arestas existentes na árvore T. A remoção de uma aresta e divide T em duas sub-árvores, cujos conjuntos de nós são K_e e $T \setminus K_e$. Na equação 2.2 o custo da aresta é multiplicado pela probabilidade da aresta estar ativa em um determinado momento. Calcula-se a probabilidade de uma aresta ser ativa através da probabilidade de se ter pelo menos um nó ativo em cada uma das componentes conexas geradas pela eliminação da aresta da árvore. Isso se deve ao fato de que, em uma árvore, entre todo par de nós existe apenas um caminho. Como as probabilidades são independentes, a probabilidade da aresta ser ativa é calculada através da multiplicação das probabilidades de pelo menos um nó estar presente em cada uma das componentes conexas, ou seja, uma unidade subtraída da probabilidade de nenhum nó estar presente.

Para o caso homogêneo do problema, objeto dessa dissertação, considera-se que para todos os nós $p_i = p$. Sendo que k_e representa a cardinalidade do conjunto K_e, pode-se desenvolver a equação 2.2 para:

$$E[L_T] = \sum_{e \in T} c_e \left\{ 1 - (1 - p)^{k_e} \right\} \times \left\{ 1 - (1 - p)^{n - k_e} \right\}. \tag{2.3}$$

A critério de simplificação, considerando uma variável q tal que $q = 1 - p$, a equação 2.3 pode ser reescrita como:

$$E[L_T] = \sum_{e \in T} c_e(1 - q^{k_e})(1 - q^{n-k_e}). \tag{2.4}$$

Em Bertsimas [1988b] é provado que o PMST é NP-Completo, com base na exploração da função de probabilidade P_S da equação 2.1 e na redução do PMST ao Problema de Cobertura Exata por Triplas (do inglês *Exact Cover By 3-Sets*) conforme o trabalho de Garey & Johnson [1979].

Algumas aplicações práticas podem ser modeladas através de uma árvore geradora em um grafo cujos nós possuam determinada probabilidade de estarem presentes. Tais aplicações são citadas em Abuali et al. [1994, 1995] e exemplificadas em Bertsimas [1988a,b,c] e Bertsimas et al. [1989]. Nesse âmbito em que algum tipo de fator aleatório existe, esse problema pode ser mais apropriado do que fazer uso, por exemplo, da modelagem clássica da Árvore Geradora Mínima, e ter que utilizar mecanismos de reotimização que atuam posteriormente a cada modificação na instância trabalhada. A característica essencial do PMST é justamente ser um problema mais global, realista, do que o problema da árvore geradora mínima em determinadas situações.

2.2 O PMST e o Problema da Árvore de Steiner

Existe certo interesse em comparar o PMST com o Problema da Árvore de Steiner em Grafos [Hwang et al., 1992], pois no PMST deseja-se atualizar eficientemente soluções de árvore geradora mínima, quando instâncias são modificadas probabilisticamente devido à ausência de determinados nós de um grafo. No problema da Árvore de Steiner, dado um grafo $G = (V, E, w)$, composto por um conjunto V de vértices e E de arestas, tem-se como objetivo conectar com custo mínimo um sub-conjunto w de nós terminais pertencente a V.

Bertsimas [1988b] analisa a utilização de otimização *a priori* no contexto do PMST comparada à utilização de diferentes estratégias de reotimização, aplicadas a cada modificação de certa instância ao longo do tempo. Uma das estratégias descritas é justamente a de se utilizar conceitos da Árvore de Steiner. A análise realizada na literatura é a que se segue.

Supõe-se que para um determinado conjunto de nós, em que apenas um conjunto S está ativo, uma estratégia de otimização *a priori* tem como solução uma árvore $T_P(S)$ de custo $L_{T_P}(S)$. Esse seria o custo da árvore que conecta os nós ativos do conjunto

S utilizando partes da árvore T_P. Uma estratégia de reotimização utilizando Steiner a cada modificação do conjunto S de nós, corresponderia a um custo $L_{STEINER}(S)$.

Considerando a estratégia de reotimização por Steiner, é evidente que $L_{STEINER}(S) < L_{T_P}(S)$, pois a árvore que conecta o conjunto S utilizando apenas partes da árvore T_P é também uma solução para o problema da Árvore de Steiner em S. A desvantagem da estratégia de Steiner é que a cada modificação da instância, deve-se resolver um problema NP-Difícil. Isso seria viável apenas quando as instâncias fossem pequenas. Ainda, com otimização *a priori*, desenvolve-se uma solução que atende a diversas variações de uma determinada instância ao longo do tempo, ao passo que a estratégia de reotimização com Steiner exige o conhecimento de todos os nós ativos a cada variação. Sendo assim, a Árvore de Steiner pode ser considerada um caso especial do PMST, onde os nós terminais têm probabilidade de ativação igual a 1 e os nós não terminais ($V \backslash w$) têm probabilidade de ativação igual a 0.

2.3 Trabalhos Relacionados

O interesse no estudo de problemas de otimização combinatória cujas instâncias são modificadas probabilisticamente começou com o Problema do Caixeiro Viajante Probabilístico (PTSP[1]) [Jaillet, 1985]. Jaillet [1985] examina as propriedades combinatórias deste problema no qual o número de localidades a serem visitadas em cada instância varia aleatoriamente. Bertsimas [1988a] descreve e estuda alguns resultados do PTSP e inclui alguns resultados do Problema de Roteamento de Veículos Probabilístico, definido por Jaillet & Odoni [1988], além de problemas probabilísticos de localização de facilidades.

O PMST foi proposto pela primeira vez na literatura em Bertsimas [1988a] e posteriormente, teve suas propriedades combinatórias analisadas em dois volumes [Bertsimas, 1988b,c]. No primeiro, o PMST é abordado como uma variação do problema clássico da Árvore Geradora Mínima, discutindo suas aplicações e provendo formulações para a medição do custo ativo esperado de uma árvore geradora mínima probabilística. Discutem-se, ainda, as propriedades combinatórias do problema, estabelecendo relações entre o PMST, aspectos do problema da Árvore Geradora Mínima e do problema de Modelagem de Redes [Garey & Johnson, 1979]. O segundo volume trata de análises probabilísticas do problema, aproximando-o em formulação do problema da Árvore Geradora Mínima e da Árvore de Steiner em Grafos [Hwang et al., 1992]. As formulações do PMST propostas fornecem insumos para se comparar a sua resolução

[1]Do inglês *Probabilistic Traveling Salesman Problem*.

com problemas semelhantes, porém resolvidos através de estratégias de reotimização. Como resultado, evidencia-se a utilidade e praticidade de se utilizar estratégias *a priori* para resolver problemas de otimização combinatória em instâncias que variam ao longo do tempo.

Ao contrário do que se encontra sobre trabalhos relacionados ao MST ou mesmo suas especializações, tal como a árvore geradora mínima com Restrição de Grau, Estocástica etc, existem poucos trabalhos que abordam o PMST. Os trabalhos existentes na literatura abordam o problema através de algoritmos genéticos, tal como se observa em Abuali et al. [1994], ou algoritmos genéticos associados a diferentes esquemas de codificação, em Abuali et al. [1995]. No primeiro, o uso de algoritmos genéticos auxilia no desenho de redes de telecomunicações, com enfoque em estratégias específicas do algoritmo, como mutação e cruzamento, tratando de instâncias de até 20 nós. A segunda referência compara o uso das mesmas heurísticas, porém associadas a diferentes tipos de codificação, ampliando as instâncias para até 30 nós. Ambos os trabalhos tratam apenas do caso homogêneo do problema. Dada a complexidade das equações nele envolvidas, observou-se que a utilização de simulação pode ser utilizada como uma ferramenta interessante para a avaliação de árvores geradoras propostas através de heurísticas construtivas, sugeridas por Souza & Urrutia [2008].

No trabalho de Balaprakash et al. [2007], que trata de algoritmos de busca local estocástica, é apresentado um estudo de caso de busca local baseada em estimativas para o problema do caixeiro viajante probabilístico. As ideias que sustentam o estudo consistem de técnicas de *speed-up* para melhorar os algoritmos e também da utilização de estimativas empíricas para avaliar movimentos no espaço de busca. Um entendimento em alto nível deste problema também se enquadraria no PMST, pois se trata de um problema que consiste de uma otimização *a priori*, que define uma solução com o intuito de minimizar a esperança do custo de soluções conhecidas apenas *a posteriori*. Os resultados positivos do trabalho, que superaram as soluções previamente conhecidas para o problema, serviram como motivação para se calcular o custo das árvores geradoras trabalhadas no PMST fazendo o uso de dados experimentais, ou seja, através de estimativas empíricas. O uso de simulação, portanto, mostrou ter papel interessante como forma de validar os resultados encontrados através das heurísticas construtivas.

No momento, o PMST é um problema pouco estudado e de difícil resolução. Em suas abordagens não existem técnicas para trabalhar com instâncias muito grandes e, até onde se sabe, não há uma proposição de modelagem exata, que não a de Souza & Urrutia [2010], tampouco uma abordagem que envolva busca local.

Capítulo 3

Um Modelo de Programação Inteira para o PMST

3.1 O Modelo

Até o presente momento, é desconhecida na literatura a existência de qualquer modelo que permita a resolução exata do PMST. Nesta seção é proposta uma modelagem por programação linear inteira que pode ser utilizada para resolver o problema em sua abordagem homogênea, tendo-se $p_i = p$, onde $0 < p \leq 1$.

Para este problema, tem-se variáveis de decisão $x_{(i,j)k}$ que assumem valor 1 se existe uma aresta (i, j) tal que, quando removida, divide a árvore em duas sub-árvores de k e $n - k$ nós, e 0 caso contrário. De maneira auxiliar, recorre-se ao uso das variáveis y_{ij}, que determinam o valor de k para uma aresta (i, j). As variáveis s_{ij} são tais que se tiverem valor 1, significa que, ao se eliminar uma aresta (i, j), a sub-árvore remanescente que possui i contém menos nós do que a sub-árvore que contém j. Caso contrário, s_{ij} adota valor 0. Em outras palavras, k terá o tamanho da sub-árvore que contém i se $s_{ij} = 1$ ou o tamanho da sub-árvore que contém j, caso contrário.

Algumas considerações são necessárias para a apresentação do modelo:

$$x_{(i,j)k} \in \{0, 1\}, \quad \forall (i, j) \in E, \quad k = 1, 2, \ldots, \frac{n}{2}, \quad i < j \tag{3.1}$$

$$y_{ij} \in \mathbb{Z}_+, \quad \forall (i, j) \in E, \quad i < j \tag{3.2}$$

$$s_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E, \quad i < j \tag{3.3}$$

Utiliza-se uma função de custo de probabilidade Pc[1], que representa o quanto uma aresta contribui para o custo total da árvore, ou seja, a probabilidade de ativação de uma aresta, probabilidade esta determinada pelos tamanhos dos conjuntos de nós que ela interliga, multiplicada pelo seu custo.

$$Pc_{(i,j)k} = c_{ij} \cdot (1 - q^k)(1 - q^{n-k}), \quad q = 1 - p, \quad 0 < p < 1 \tag{3.4}$$

A formulação do problema é apresentada a seguir.

$$minf = \sum_{(i,j)\in E} \sum_{k=1}^{\frac{n}{2}} Pc_{(i,j)k}\ x_{(i,j)k} \tag{3.5}$$

Sujeito a:

$$\sum_{(i,j)\in E} \sum_{k=1}^{\frac{n}{2}} x_{(i,j)k} \ = \ n - 1 \tag{3.6}$$

$$\sum_{k=1}^{\frac{n}{2}} x_{(i,j)k} \ \leq \ 1, \quad \forall (i,j) \in \ E, \quad i < j \tag{3.7}$$

$$\sum_{k=1}^{\frac{n}{2}} k x_{(i,j)k} \ = \ y_{ij}, \quad \forall (i,j) \in \ E, \quad i < j \tag{3.8}$$

$$y_{ij} \ + \ n(1 - s_{ij}) \ + \ n \left(1 \ - \ \sum_{k=1}^{\frac{n}{2}} x_{(i,j)k} \right) \geq \ 1 + \sum_{\substack{(i,l)\in E \\ l \neq j}} y_{il}, \quad \forall (i,j) \in \ E, \quad i < j \tag{3.9}$$

$$y_{ij} \ + \ n \cdot s_{ij} \ + \ n \left(1 \ - \ \sum_{k=1}^{\frac{n}{2}} x_{(i,j)k} \right) \geq \ 1 + \sum_{\substack{(j,l)\in E \\ l \neq i}} y_{jl}, \quad \forall (i,j) \in \ E, \quad i < j \tag{3.10}$$

A figura 3.1 ajuda na compreensão da formulação proposta. Nela é ilustrada

[1]Do inglês *Probability Cost*.

a remoção de uma aresta e de custo c_e que conecta os vértices 7 e 5 de uma árvore específica, dividindo-a em duas sub-árvores representadas pelo conjunto de vértices K_e e seu conjunto complementar $V \backslash K_e$.

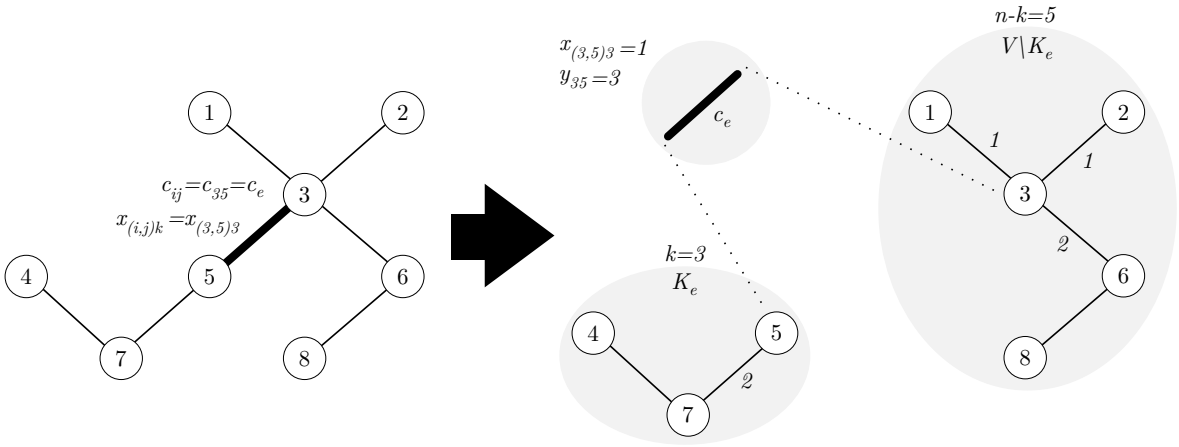


Figura 3.1. Remoção de aresta $e = (3, 5)$, de custo c_e.

A função objetivo na equação 3.5 tem como propósito minimizar o custo ativo esperado ($E[L_T]$) da árvore geradora. $E[L_T]$ é calculado a partir do somatório das funções de custo de probabilidade Pc, multiplicado individualmente pelas variáveis de decisão $x_{(i,j)k}$. Essas variáveis de decisão representam a existência de uma aresta (i, j) que realize a conexão de duas sub-árvores de tamanhos k e $n - k$. Considera-se que k atinge no máximo $\frac{n}{2}$ já que, por simetria, esse é o maior corte possível em uma árvore de n vértices. Cortes com $n > k > \frac{n}{2}$ produzem sub-árvores com os mesmos tamanhos, apenas invertendo qual das sub-árvores possui mais ou menos nós.

Pc, definida na equação 3.4, corresponde à inserção da função de probabilidade apresentada na equação 2.1 e expandida na equação 2.2 para o cálculo do custo ativo esperado da árvore. Supondo-se a remoção de uma aresta e qualquer, como na figura 3.1, a função de probabilidade está relacionada à probabilidade dos nós de cada uma das sub-árvores resultantes estarem ativos e ao custo c_e de e. A aresta terá alto custo ativo à medida que conectar duas sub-árvores que tenham grande probabilidade de ter pelo menos um nó ativo. Uma aresta terá custo ativo baixo caso conecte sub-árvores com baixa probabilidade de pelo menos um nó estar ativo, o que torna o valor da função Pc mais baixo. Quanto mais altos forem os valores das probabilidades dos nós estarem ativos, maior será o valor atribuído a Pc. Na equação 3.4, por questões de simplificação, a probabilidade de um nó estar ativo, denotada por p, é convenientemente substituída por q, que se trata da probabilidade de um nó não estar ativo futuramente.

A equação de restrição 3.6 garante que o número de arestas seja igual a $n - 1$,

quantidade de arestas necessária para a construção de uma árvore geradora de um grafo com n vértices. As restrições estabelecidas na equação 3.7 informam que uma determinada aresta pode ou não ser usada de forma que, se usada, ela possui um único valor de k. As restrições agrupadas na equação 3.8 ligam as variáveis y_{ij} às variáveis $x_{(i,j)k}$ indicando o valor do corte, ou seja, o valor de k para uma aresta (i, j) pertencente à árvore. As equações de restrição 3.9 e 3.10 limitam o valor de y_{ij} para cada aresta (i, j). Esse cálculo é feito a partir dos valores das variáveis y_{il} para as arestas incidentes no vértice i (restrição 3.9) e a partir dos valores das variáveis y_{jl} para as arestas incidentes no vértice j (restrição 3.10). O segundo termo da parte esquerda das inequações torna trivialmente satisfeita uma das duas restrições conforme o valor da variável s_{ij}. O terceiro termo da parte esquerda desativa as restrições quando a aresta (i, j) não faz parte da árvore. Portanto, apenas uma das restrições será analisada para cada aresta. Estando uma das restrições ativa, o seu significado é que o valor de y_{ij} para uma aresta (i, j) deve ser maior ou igual à soma do valor de y_{ij} de todas as arestas incidentes a um mesmo vértice da aresta em consideração, acrescida de uma unidade. Na figura 3.1, y_{35}, que representa y_{ij} para a aresta $(3, 5)$, é igual a 3. A aresta que incide no nó 5 possui $y_{57} = 2$, que acrescido de 1 totaliza 3. Os valores de k para as arestas que incidem no nó 3, somadas de 1 totalizam 5. y_{35} é pelo menos 3. No caso, $y_{35} = 3$, pois, para a variável y_{ij}, considera-se o menor valor entre k e $n - k$ (essa propriedade é verificada através da variável s_{ij}, aliada à definição de que sempre $i < j$).

As equações de restrição 3.9 e 3.10 são também úteis na prevenção de ciclos, o que se demonstra na figura 3.2. Supondo a tentativa de inserção de uma aresta a entre os nós 1 e 2, um ciclo seria formado. Para que uma das duas restrições seja atendida, observando-se o lado direito da desigualdade de cada uma, o valor de k_a deve ser maior ou igual à soma das variáveis k de todas as arestas incidentes em um dos nós conectados por a, acrescida de uma unidade. Supondo que k_a é maior que k_b, atendendo a 3.9 ou 3.10, prossegue-se na avaliação das outras arestas do ciclo. Como $k_b < k_a$, para atender a uma das restrições, k_b deverá ser maior que k_c. Seguindo o mesmo raciocínio, $k_c > k_d$. Por fim, para que k_d não viole as duas restrições, k_d deve ser obrigatoriamente maior que k_a. Porém, k_a é o maior dos k's, já que $k_a > k_b > k_c > k_d$. Por fim, k_d acaba por violar as duas restrições. Sempre que um ciclo se formar, uma das arestas pertencentes a ele fará com que as duas restrições sejam violadas, inviabilizando a solução avaliada.

Discutindo-se a quantidade de variáveis e restrições, sendo E o conjunto de arestas do grafo utilizado e V o seu conjunto de vértices, a modelagem proposta apresenta $O(|V|\times|E|)$ variáveis $x_{(i,j)k}$, $O(|E|)$ variáveis y_{ij}, $O(|E|)$ variáveis s_{ij} e $O(|E|)$ restrições. Essa modelagem só é possível graças à consideração da probabilidade de todos os nós ser a mesma. Em uma abordagem heterogênea exige-se uma modelagem bem mais

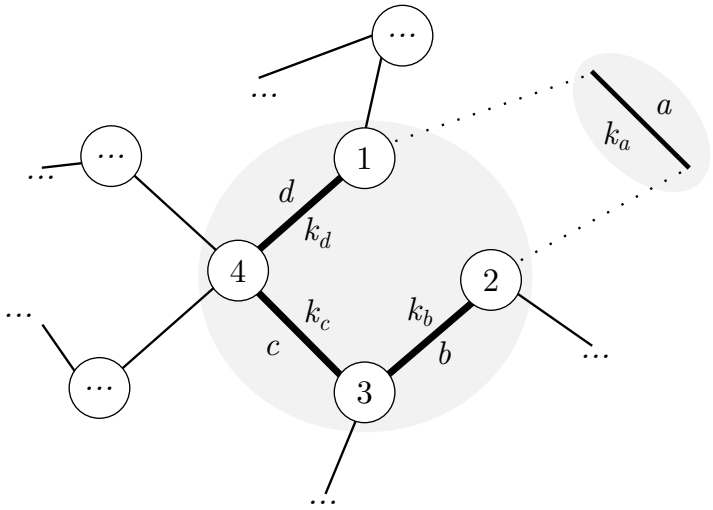


Figura 3.2. Exemplo de situação onde um ciclo poderia ser evitado pelas restrições das equações 3.9 e 3.10.

complexa – e desconhecida até o momento – já que o fator responsável por possibilitar a modelagem apresentada é que basta que se saiba o número de nós existentes de cada lado de uma aresta para determinar a probabilidade de existência de um nó ativo nas sub-árvores conectadas por ela.

3.2 Limite Inferior

O custo ativo esperado de uma árvore geradora T é uma função que depende do custo de suas arestas e da probabilidade de cada uma delas estar ativa. Sendo assim, $E[L_T]$ pode ser descrito a partir de uma função de probabilidade P_e, relativa à probabilidade de uma aresta e estar ativa futuramente:

$$E[L_T] = \sum_{e \in T} c_e P_e \tag{3.11}$$

Conforme explicado anteriormente, P_e depende exclusivamente da quantidade de nós existentes em cada sub-árvore conectada por uma aresta e, cujos tamanhos são respectivamente $|K_e| = k$ e $|V \backslash K_e| = n - k$:

$$P_e = \{1 - (1-p)^k\}\{1 - (1-p)^{n-k}\}. \tag{3.12}$$

Sendo idênticas as probabilidades para todos os nós de T, a menor probabilidade de uma aresta estar ativa é na situação em que um de seus nós incidentes é uma folha

da árvore. Isto implica em $k = 1$, o que simplifica P_e a

$$P_e = p(1 - (1 - p)^{n-1}). \tag{3.13}$$

Supondo uma árvore em que cada aresta está presente conforme sua menor probabilidade possível, *i.e.* uma árvore de topologia em estrela, o custo ativo esperado pode ser limitado inferiormente conforme

$$\sum_{e \in T} c_e P_e \ \geq \sum_{e \in T} c_e p(1 - (1 - p)^{n-1}) \tag{3.14}$$

Como $p(1 - (1 - p)^{n-1})$ é constante, sendo possível retirá-lo do somatório e, sabendo que o limite inferior sobre o custo $\sum_{e \in T} c_e$ de qualquer árvore geradora é o custo da árvore geradora mínima do grafo (L_{AGM}) um limite inferior pode ser determinado para uma árvore T:

$$p(1 - (1 - p)^{n-1}) \ \sum_{e \in T} c_e \ \geq p(1 - (1 - p)^{n-1}) L_{AGM}. \tag{3.15}$$

Sendo assim, tem-se um limite inferior para o PMST. O custo ativo esperado para qualquer solução T é maior ou igual a:

$$p(1 - (1 - p)^{n-1}) L_{AGM}. \tag{3.16}$$

3.3 Experimentos Computacionais

Nesta seção investiga-se a formulação proposta através de experimentos computacionais, a partir dos quais é possível avaliar de maneira empírica a capacidade de um resolvedor (*solver*) de solucionar instâncias do problema por meio do modelo. Como para instâncias de testes para 10 ou mais nós não é possível atingir uma solução ótima em até 2 horas, a execução do algoritmo foi limitada neste tempo.

Os testes são baseados em instâncias para o Problema de Geração de Escalas de Jogos[2][CTT Instances, 2010]. Foram utilizadas instâncias de 4 a 16 nós, de grafos completos, escolhidas arbitrariamente. Considerou-se que todos os nós possuíam a mesma probabilidade de estarem ativos, sendo realizados testes para probabilidades iguais a 0.3, 0.5 e 0.8. Os experimentos foram executados em uma máquina com 3GB de memória (RAM) principal e processador Intel®Core™2 Duo T5550 (2MB Cache, 1.83 GHz, 667 MHz FSB), rodando o sistema operacional Microsoft®Windows

[2]Também conhecido em inglês como *Traveling Tournament Problem*

Vista^{TM}Service Pack 2 (SP2). O modelo de programação linear inteira foi implementado e executado no pacote de otimização ILOG CPLEX [2010] versão 10.2.0 com os parâmetros padrão (*default*).

3.3.1 Análise dos Resultados

A tabela 3.1 apresenta os resultados para a resolução do modelo em Programação Linear Inteira. São apresentados, também, dois limites inferiores, sendo L_{AGM} o limite demonstrado na seção 3.2 e determinado pela equação 3.16. O outro limite inferior utilizado para comparação foi aquele correspondente à Relaxação Linear do modelo. O limite L_{AGM} foi comparado percentualmente com os custos ativos das árvores geradas para cada instância, já que apresentou valores mais altos.

Tabela 3.1. Resultados da resolução do modelo por Programação Linear Inteira, para um determinado grupo de instâncias e três valores diferentes de probabilidade.

Instâncias			Limite Inferior		Modelagem Exata			
			Rel. Linear	L_{AGM}	Resolvedor			
p	instância	n	custo	custo	$E[L_T]$	gap (%)	diferença L_{AGM}(%)	tempo(s)
0.30	NL4	4	157.09	213.26	218.30	0.00	2.36	1.37
	NL6	6	341.67	449.24	574.30	0.00	27.84	3.54
	NL8	8	512.60	592.43	797.96	0.00	34.69	51.04
	NL10	10	615.23	719.16	1020.08	18.85	41.84	7200.00
	NL12	12	823.98	1167.74	1591.76	26.70	36.31	7200.00
	NL14	14	1055.28	1563.01	2468.06	42.83	57.90	7200.00
	NL16	16	1169.82	1586.93	2903.96	50.56	82.99	7200.00
0.50	NL4	4	348.69	473.38	483.38	0.00	2.11	1.51
	NL6	6	663.11	871.88	1075.63	0.00	23.37	3.15
	NL8	8	923.73	1067.59	1351.52	0.00	26.60	26.52
	NL10	10	1066.41	1246.56	1632.21	8.10	30.94	7200.00
	NL12	12	1400.32	1984.53	2490.31	23.61	25.49	7200.00
	NL14	14	1775.78	2630.18	3671.36	35.59	39.59	7200.00
	NL16	16	1958.94	2657.42	4071.57	40.65	53.22	7200.00
0.80	NL4	4	632.50	858.68	868.92	0.00	1.19	2.22
	NL6	6	1094.85	1439.54	1582.30	0.00	9.92	2.36
	NL8	8	1489.58	1721.58	1892.61	0.00	9.93	7.89
	NL10	10	1709.60	1998.40	2217.68	0.00	10.97	4491.30
	NL12	12	2241.60	3176.80	3448.18	18.76	8.54	7200.00
	NL14	14	2841.60	4208.80	4767.82	27.62	13.28	7200.00
	NL16	16	3134.40	4252.00	4875.67	40.63	14.67	7200.00

Conforme é possível perceber, para instâncias de mesmo tamanho mas diferentes probabilidades, a diminuição da probabilidade dificulta a resolução do problema. Isso

é constatado através do tempo de execução, que aumenta para quando a probabilidade é menor, e também através do intervalo entre os limites superior e inferior informados pelo CPLEX, que diminui conforme o aumento da probabilidade dos nós. Esse aspecto, de certa forma, confirma o que é relatado por Bertsimas [1988b]: à medida que a probabilidade dos nós estarem ativos aumenta, o problema se aproxima do Problema da Árvore Geradora Mínima. Tal aproximação também é validada pela observação de que à medida que a probabilidade dos nós aumenta, o limite inferior L_{AGM} é mais próximo percentualmente do valor do custo ativo esperado das soluções encontradas.

Para problemas com variáveis inteiras, o CPLEX utiliza uma abordagem *branch-and-bound*, de forma que o algoritmo de otimização resolve uma série de subproblemas armazenados individualmente em nós de uma estrutura em árvore. Tomando como exemplo a instância de 12 nós, no momento em que a execução foi encerrada (com 2 horas) o tamanho da árvore para $p = 0.80$ era de aproximadamente 1160MB. Para $p = 0.30$ a árvore possuía 1560MB. No mesmo instante, enquanto para a probabilidade maior ainda restavam 3.394.386 nós a serem explorados, para a probabilidade menor restavam 4.655.703 nós. A memória consumida cresce rapidamente na resolução dessa modelagem, restringindo a tentativa de resolução para instâncias maiores.

Estes resultados sinalizam que o caminho de se resolver o PMST através de heurísticas é favorável. Outros resultados relacionados ao modelo foram publicados em Souza & Urrutia [2010].

Capítulo 4

Heurísticas para o PMST

A abordagem através de heurísticas para o PMST se justifica pela dificuldade de se encontrar uma solução exata para o problema [Abuali et al., 1995, 1994; Souza & Urrutia, 2008]. Em Souza & Urrutia [2008] segue-se na direção de propor heurísticas construtivas para gerar soluções iniciais viáveis com um razoável custo ativo esperado. Comparativamente à solução do modelo por Programação Linear Inteira, nota-se, em Souza & Urrutia [2010], que os custos ativos esperados das soluções informadas pelas heurísticas chegam a ser relativamente bons em detrimento do tempo. Em algumas instâncias, para as quais o resolvedor não foi capaz de encontrar o valor ótimo em um limite de tempo específico, as heurísticas construtivas chegaram a soluções bem próximas daquelas obtidas pela resolução exata no limite de tempo pré-estabelecido.

É pertinente dizer que as soluções encontradas pelas heurísticas construtivas ainda possuem condição de serem exploradas, o que justifica o foco em uma maneira ágil de calcular o custo ativo esperado de uma árvore. Esse cálculo, realizado a partir da equação 2.2, é crucial para o bom desempenho de algoritmos de busca local aplicáveis ao problema, porque sempre que uma solução vizinha é encontrada, é preciso realizar um cálculo de custo para comparar a vizinhança explorada com a solução atual. Um bom algoritmo de busca local, além de avançar na abordagem com heurísticas, permite uma posterior integração com metaheurísticas. Neste trabalho utiliza-se Busca Tabu com esta finalidade.

Como forma de apresentar a resolução do problema através de algoritmos de Busca Local integrados a uma metaheurística a seção 4.1 apresenta o algoritmo utilizado para construir soluções viáveis para o problema. A seção 4.2 descreve o cálculo eficiente do custo ativo esperado de uma árvore geradora do caso estudado e como o custo pode ser avaliado em movimentos da busca local dentro do espaço de busca. Na seção 4.3 explica-se como a busca local é possível, suas particularidades e variações.

Na seção 4.4 aborda-se a metaheurística de Busca Tabu e como ela incorpora os algoritmos de busca local. Em seguida, na seção 4.5, são apresentados os experimentos e resultados obtidos.

4.1 Geração de Soluções Iniciais

Em Souza & Urrutia [2008] são propostas heurísticas construtivas para o problema, todas baseadas no algoritmo de Prim [Prim, 1957]. Nesta dissertação, como o objetivo das heurísticas estava limitado apenas a criar soluções iniciais viáveis para uma posterior execução de busca local, decidiu-se por utilizar o algoritmo de Prim em sua forma clássica, descrito no algoritmo 4.1 [veja Cormen et al., 2001, p.572].

Outra justificativa para se gerar soluções iniciais viáveis através do algoritmo de Prim é que foram testadas outras heurísticas construtivas não sendo observada forte interferência na solução final encontrada. Sendo assim, optou-se por esse algoritmo devido à sua rapidez de execução. Com ele, apenas o custo das arestas é considerado para a computação do custo de inserção de uma aresta, ao contrário de outra heurística proposta em Souza & Urrutia [2008], que considera a distância dos caminhos entre os nós já conectados à árvore para realizar as novas inserções.

Algoritmo 4.1: Algoritmo de Prim clássico

Entrada: $G = (V, E)$, $r \in V$
Saída: Árvore Geradora Mínima A tal que $A = (v, \pi[v]) : v \in V \backslash r$

```
1: para cada u ∈ V[G] faça
2:    key[u] ← ∞
3:    π[u] ← nil
4: fim para
5: Q ← V[G]
6: enquanto Q ≠ ∅ faça
7:    u ← ExtrairMínimo(Q)
8:    para cada v ∈ Adj[u] faça
9:       se v ∈ Q e w(u, v) < key[v] então
10:         π[v] ← u
11:         key[u] ← w(u, v)
12:      fim se
13:   fim para
14: fim enquanto
```

A ordem de complexidade para o algoritmo é $O(|E| \log |V|)$, sendo E o conjunto de arestas e V o conjunto de vértices de um grafo $G = (V, E)$. É importante considerar

que esse algoritmo produz uma árvore geradora que é mínima para o MST, mas não necessariamente será mínima para o PMST.

A árvore A começa a ser construída a partir de um nó raiz r. O algoritmo de Prim trata de conectar a r o vértice mais próximo dele (aresta de menor custo), e sempre o nó mais próximo à sub-árvore já construida, sucessivamente, até que todos os vértices do grafo G estejam presentes na árvore – o que acontece quando a fila de prioridades Q se torna vazia. Para cada vértice v, $key[v]$ é o menor custo de qualquer aresta que conecte v a um vértice na árvore. O campo $\pi[v]$ identifica o nó predecessor (pai) de v na árvore. Pressupõe-se que o grafo G é representado através de listas de adjacência (Adj). A implementação de Q é determinante no desempenho do algoritmo, principalmente devido à operação $ExtrairMínimo(Q)$, que seleciona o próximo vértice a participar da árvore. A utilização de listas de adjacência para representar o grafo G foi importante não só para a codificação do algoritmo de Prim, mas também para os outros algoritmos empregados no trabalho.

Por simplicidade, optou-se por implementar um *Heap* Indireto em Q, mas pode-se escolher outros tipos de *Heaps*, como *Heap* Binomial, *Heap* de Fibonacci etc. Conforme Ziviani [2004], através da utilização de um *Heap* Indireto o algoritmo de Prim possui complexidade de $O(|T_{Adj}|log|V|)$, onde T_{Adj} é o tamanho das listas de adjacência utilizadas para representar G.

4.2 Cálculo do Custo Ativo Esperado de uma Árvore Geradora Mínima Probabilística

O custo ativo esperado de uma árvore geradora no contexto do PMST é determinado pela equação 2.2. Para o cálculo, intuitivamente, pode-se utilizar qualquer algoritmo elementar de busca em grafos que seja capaz de percorrer as arestas da árvore e visitar os seus nós. Por exemplo, pode-se utilizar um algoritmo de busca em largura (BFS[1]) ou de busca em profundidade (DFS[2]). Esses algoritmos podem ser executados em tempo razoável, dado que o tempo de execução deles é $O(V + E)$ [Cormen et al., 2001].

A partir da equação 2.4, basta que se saiba em quantos nós a remoção de uma determinada aresta subdivide uma árvore para que seja possível calcular o seu custo probabilístico associado. Não é necessário saber quais nós estão presentes em cada sub-árvore para se ter acesso às probabilidades, mas sim quantos nós existem – já que a

[1]Do inglês *Breadth-First Search*.
[2]Do inglês *Depth-First Search*.

probabilidade para todos eles é a mesma. Essa é a primeira característica que viabiliza o cálculo eficiente do custo ativo esperado.

Conhecendo o valor de k para cada aresta existente na árvore, utiliza-se a equação 2.4 para determinar a contribuição de cada aresta para o custo total. O custo ativo esperado da árvore é facilmente calculado a partir do somatório das contribuições individuais de cada uma das arestas. Conforme abordado no capítulo 2, o valor de k de uma aresta será sempre o tamanho da menor sub-árvore gerada pela sua remoção. Assim, como discutido na seção 3.1, sempre que se remove uma aresta de uma árvore existente, tem-se como resultado duas sub-árvores de tamanhos k e $n-k$, sendo n a quantidade total de nós presentes na árvore.

A partir das considerações realizadas, propõe-se que se utilize um algoritmo de busca em profundidade, descrito no algoritmo 4.2, para determinar os k's das arestas de uma árvore. Esse algoritmo é uma adaptação do DFS proposto em Cormen et al. [2001] e faz uso de uma função recursiva, apresentada no algoritmo 4.3. O primeiro realiza as configurações de variáveis necessárias e o segundo realiza a visita recursiva aos nós da árvore. Como a árvore é totalmente conectada, basta tomar qualquer nó inicial para que todos os nós sejam visitados. Um benefício deste algoritmo é que, além de percorrer a árvore, é possível determinar os predecessores de cada nó, fator fundamental para posteriores atualizações dos valores de k durante a busca local.

A execução do algoritmo 4.3 a partir de um nó u resulta na visita recursiva de cada nó v adjacente a ele, retornando a quantidade de nós existentes a partir de v. Todos os nós visitados a partir desse ponto terão u como predecessor. O valor de k para a aresta (u, v) é determinado por uma função de mínimo que compara o valor retornado com o seu complemento. Esse procedimento é necessário para manter a consistência de k, que varia de 1 a $\frac{n}{2}$. Ao final da execução do algoritmo, k_{in_u} é retornado, sendo composto pela soma dos k's de todas as arestas que nele incidem acrescido de 1, referente ao próprio u. A execução de *Visitar-DFS(u)*, além de visitar todos os nós da árvore, preenche os valores de k para todas as arestas existentes.

Algoritmo 4.2: Determinação dos k's das arestas de uma árvore A

Entrada: Árvore A
Saída: Matriz k, contendo os k's das arestas
1: **para** cada $u \in A$ **faça**
2: $visitado[u] \leftarrow$ FALSO ◁ Indica se u já foi visitado
3: $\pi[u] \leftarrow nil$ ◁ Indica o nó pai de u
4: $k \leftarrow Zeros()$ ◁ Matriz k é preenchida com zeros
5: **fim para**
6: **Visitar-DFS**(v) ◁ vértice $v \in V[G]$ qualquer

Algoritmo 4.3: Visitar-DFS(u)

Entrada: Um nó $u \in A$, que se deseja visitar recursivamente
Saída: Valor de k_{in_u} para arestas incidentes em u

1: $visitado[u] \leftarrow$ TRUE
2: $k_{in_u} \leftarrow 1 \triangleleft$ Um vértice u acaba de ser visitado
3: **para** cada $v \in Adj[u]$ **faça**
4: **se** $visitado[v] =$ FALSO **então**
5: $\pi[v] \leftarrow u$
6: $k_{in_v} \leftarrow$ **Visitar-DFS**(v)
7: $k[u, v] \leftarrow min(k_{in_v}, n - k_{in_v})$
8: $k_{in_u} \leftarrow k_{in_u} + k_{in_v}$
9: **fim se**
10: **fim para**
11: **retorne** k_{in_u}

Uma vez que os k's das arestas são definidos, o custo ativo esperado da árvore será determinado pela soma de cada um dos custos multiplicados individualmente por $\{1 - (1 - p)^k\}\{1 - (1 - p)^{n-k}\}$. Para que o processo de cálculo do custo tenha um desempenho ainda melhor, armazena-se, inicialmente, o custo total de cada aresta para cada k que ela possa assumir[3]. Quando for preciso calcular o custo total de uma aresta basta recuperar o valor de seu custo para o k desejado.

Não adianta implementar um cálculo de custo baseado em k's se o algoritmo 4.2 tiver que ser executado a cada novo movimento testado em determinada vizinhança. O algoritmo é executado apenas quando se tem uma árvore inicial e a cada troca de aresta realizada. Na primeira vez, o procedimento determina os k's das arestas presentes na árvore, o que não é feito a cada troca, quando apenas alguns k's são atualizados. Em todas as execuções a árvore é enraizada a partir de um nó arbitrário, o que permite calcular rapidamente o único caminho entre dois nós na árvore. Desenvolveu-se uma técnica especial descrita na seção 4.2.1 para descrever e explicar como essa atualização é possível.

4.2.1 Atualização eficiente dos k's

Os procedimentos de busca local a serem descritos funcionam de forma semelhante. De modo geral, consistem na remoção de uma aresta da árvore e na posterior tentativa de se reconectar as sub-árvores resultantes com uma aresta que reduza o custo ativo esperado da árvore. Esse procedimento é ilustrado na figura 4.1.

[3]Conforme seção 3.1, k pode variar de 1 a $\frac{n}{2}$.

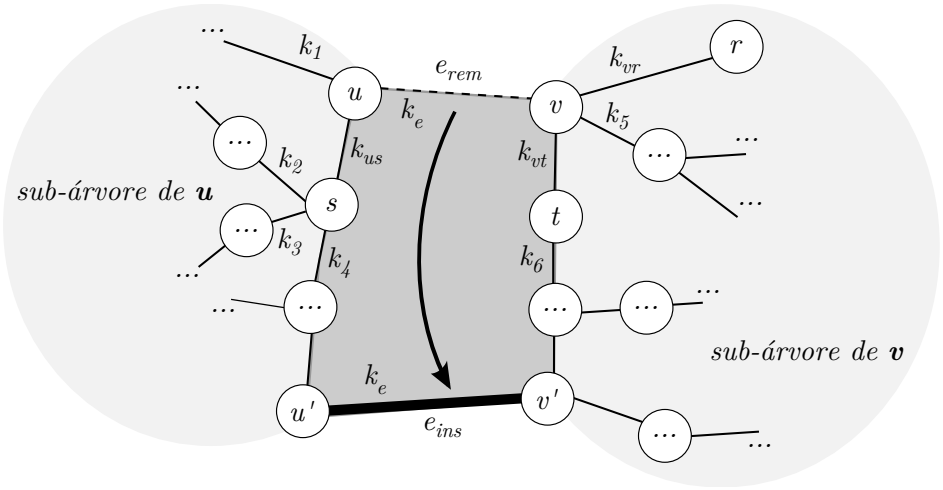


Figura 4.1. Efeito de uma troca de aresta que une duas sub-árvores.

A figura 4.1 ilustra a remoção de uma aresta e_{rem} qualquer, que conecta um par de nós (u, v), e a posterior união das sub-árvores que contêm u e v através de uma aresta (u', v') representada por e_{ins}. Qualquer que seja a aresta e_{rem}, o valor de k determinará o tamanho da menor das duas sub-árvores resultantes da sua remoção. O valor de k para a aresta inserida e_{ins} será o mesmo de e_{rem}, k_e, pois sua remoção resulta nas mesmas duas sub-árvores. Ou seja, sempre que uma aresta é removida, a aresta inserida detém o seu valor de k.

A atualização dos k's deve ser feita para as arestas envolvidas na operação de troca de e_{rem} por e_{ins}. Arestas envolvidas serão todas aquelas que terão o seu k alterado pela troca e pertencentes ao ciclo formado por u, u', v' e v, sendo possível identificá-las na figura 4.1. Para toda aresta que não está no ciclo, a operação de troca acontece entre nós que estão em uma das duas sub-árvores que conecta. Por exemplo, supondo a aresta (v, r), que possui $k = k_{vr}$, a troca de e_{rem} por e_{ins} acontece à sua esquerda, não interferindo na quantidade de nós que possui à esquerda ou à direita – apenas a configuração deles é modificada. Portanto, seu k continuará igual a k_{vr} após a troca. Essa prerrogativa é válida para todas as arestas não pertencentes ao ciclo.

Quando as arestas pertencentes ao ciclo são identificadas, também toma-se conhecimento de qual sub-árvore cada uma pertence, podendo ser a sub-árvore com mais ou menos nós. Este procedimento é realizado através da execução do algoritmo de busca em profundidade em uma das sub-árvores resultantes da remoção de e_{rem} (a partir de u, por exemplo). A chamada do algoritmo nesse ponto não prejudica o desempenho da busca local por ser realizada apenas uma vez a cada remoção de aresta. Por convenção, supõe-se que a sub-árvore à qual pertence u seja a maior. O trajeto de u até u' é constituído de arestas da sub-árvore maior e o de v' a v de arestas da sub-árvore

menor. Define-se essa separação pois as sub-árvores podem ser atualizadas de forma diferente, conforme discutido a seguir.

Para atualizar os k's das arestas no trajeto de u até u' deve-se basear nos k's das arestas adjacentes a cada um dos nós da aresta. Recorrendo à figura 4.1, partindo de u, o novo valor de k_{us} será determinado por[4] $min(k_1+1, k_2+k_3+k_4+1+k_e)$. Em seguida, k_4 será calculado com base no novo k_{us}, através de $min(k_{us}+k_2+k_3+1, k_{...}+1+k_e)$, onde $k_{...}$ representa a soma dos k's incidentes no nó posterior a s em direção a u'. Assim é realizado sucessivamente para todas arestas no trajeto, até a última aresta conectada a u'.

A sub-árvore menor também pode ser atualizada através dos k's das arestas adjacentes, mas como toda sub-árvore menor sempre possui tamanho igual a k_e, a sua atualização pode ser simplificada. Antes da troca de e_{rem} por e_{ins}, k_{vt} era definido pela quantidade de nós da sub-árvore que continha t, a princípio menor que a sub-árvore que continha v – sub-árvores resultantes de uma suposta remoção da aresta (v, t). Com a remoção de e_{rem}, k_{vt} passa a depender da sub-árvore que contém v, identificada pela soma $k_{vr} + k_5 + 1$, ou então $k_e - k_{vt}$. k_6 é redefinido por $k_6 \leftarrow k_e - k_6$, e assim por diante, para as arestas no caminho até v'.

A figura 4.2 exemplifica a situação de troca de arestas com atualização de k's de maneira mais clara. Na figura 4.2a, temos uma árvore inicial A, com os valores de k para cada aresta. Na figura 4.2b, indicamos a remoção de uma aresta (e, h) e a reconexão das sub-árvores resultantes através de uma aresta (b, c). Seguindo o critério de atualização, a atualização do trajeto de h a b é dado por:

- $k_{hd} = min(k_{hg}+k_{hk}+k_{hl}+1, k_{da}+k_{db}+k_{bc}+1) = min(1+1+1+1, 1+1+5+1) = 4.$
- $k_{db} = min(k_{da} + k_{dh} + 1, k_{bc} + 1) = min(4 + 1 + 1, 5 + 1) = 6.$

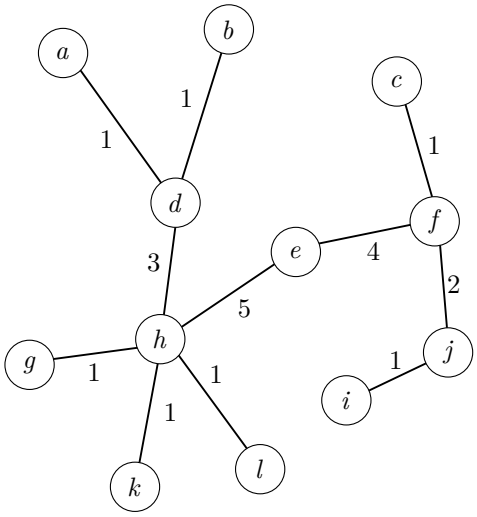
A atualização da sub-árvore menor, no trajeto de c a e é, por sua vez, dada por:

- $k_{ef} = k_{bc} - k_{ef} = 5 - 4 = 1$
- $k_{fc} = k_{bc} - k_{fc} = 5 - 1 = 4$

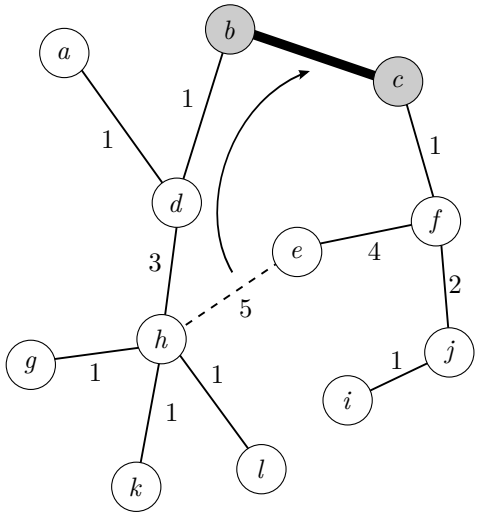
Caso fosse desejado atualizar as arestas participantes do ciclo no lado menor tal como as do lado maior foram atualizadas, teria-se:

- $k_{ef} = min(1, k_{fj} + k_{fc} + k_{bc} + 1) = min(1, 2 + 1 + 5 + 1) = 1$
- $k_{fc} = min(k_{ef} + k_{fj} + 1, k_{bc} + 1) = min(1 + 2 + 1, 5 + 1) = 4$

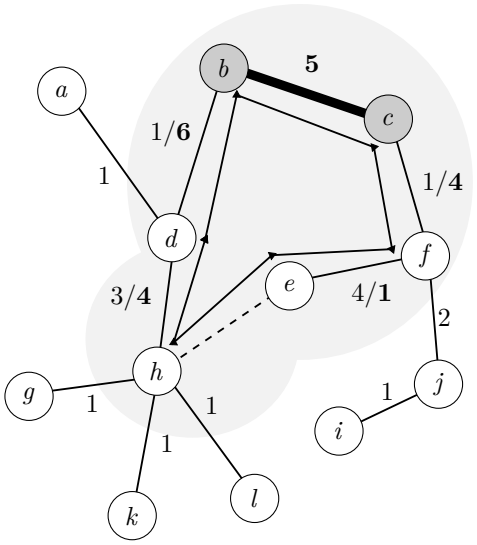
[4] $min(a, b)$ é uma função que retorna o menor valor entre dois números a e b.



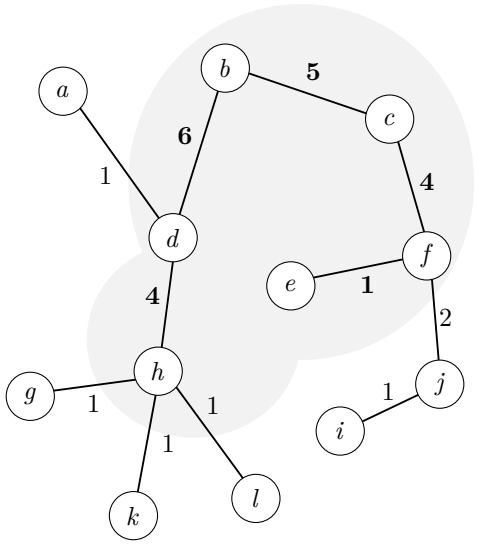
(a) Árvore inicial.



(b) Remoção de aresta (e, h) e inserção de aresta (b, c).



(c) Atualização dos k's no caminho das arestas envolvidas.



(d) Árvore final, com k's atualizados.

Figura 4.2. Exemplo de atualização de k's em uma troca de arestas.

A figura 4.2c mostra as arestas pertencentes ao ciclo definido por h, d, b, c, f, e e as respectivas atualizações de k's. Finalmente, a figura 4.2d ilustra a árvore A' obtida, com os k's devidamente atualizados.

4.3 Busca Local

O princípio de um algoritmo de busca local consiste em se partir de uma solução qualquer e percorrer sistematicamente uma vizinhança em busca de uma solução melhor

do que a corrente, até que um ótimo local seja encontrado. Definem-se, então, algumas questões fundamentais para os algoritmos de busca local propostos para o PMST. Considera-se que uma árvore A' é solução vizinha de uma árvore A se elas se diferem uma da outra por uma aresta. Sendo assim, a estratégia de busca a ser utilizada consiste em primeiro lugar na remoção de uma aresta e_{rem} qualquer de A, o que origina duas sub-árvores, e em segundo lugar, na tentativa de se encontrar uma aresta e_{ins} que as reconecte, objetivando uma melhoria do custo ativo esperado da solução corrente.

O primeiro algoritmo de busca local proposto é chamado de **Melhor Aprimorante** e ilustrado na figura 4.3. Nesta versão, a cada iteração toda a vizinhança é pesquisada (para todas as arestas e suas trocas possíveis) e a melhor solução vizinha A' encontrada se torna a solução corrente A. Supõe-se um conjunto de arestas E_A enumeradas de a a j que compõem uma árvore A. Primeiramente, remove-se a aresta a, tenta-se todas as possibilidades possíveis de reconectar as duas sub-árvores resultantes, sempre avaliando o custo das soluções vizinhas através da técnica de atualização eficiente dos k's, mas não se encontra uma aresta que melhore a solução inicial A. Em seguida, continua-se o processo para a aresta b, sem sucesso, e então é encontrada uma possibilidade de melhoria do custo ativo esperado da árvore através da remoção da aresta c. A possibilidade de troca é armazenada, mas as outras arestas ainda são verificadas. Quando a aresta g é removida, encontra-se uma aresta s que melhora a solução corrente A, com um custo ainda menor que aquele se a troca da aresta c fosse realizada. A possibilidade de troca de g é então considerada a melhor até o momento e prossegue-se com a busca para as outras arestas da árvore. Quando a aresta j é testada e não há mais nenhuma solução vizinha a ser explorada, realiza-se a troca de g por s, por ter sido a solução que melhor aprimora a solução corrente A. A árvore A' resultante da troca de g por s se torna, assim, a solução atual A.

A segunda estratégia de busca local é uma adaptação do primeiro algoritmo, sendo chamada de **Melhor Aprimorante por Aresta**. Nesta estratégia, as trocas de arestas são investigadas seguindo a mesma estratégia, porém é realizada a melhor troca que aprimora a solução atual dentre as opções de troca de uma única aresta, ou seja, nem sempre toda a vizinhança é vasculhada para se encontrar a melhor troca.

Como forma de ilustrar este algoritmo, toma-se o mesmo conjunto E_A, agora ilustrado na figura 4.4. Suponhamos que existe mais de uma aresta que pode substituir a aresta c, reduzindo o custo ativo esperado da solução corrente. Dentre todas as opções de troca de c possíveis, é escolhida a melhor (no caso, m). Pode ser que, como a solução foi modificada, a troca de g por s agora já não melhore a solução corrente tal como aconteceu no exemplo do algoritmo de busca local anterior. Na figura 4.4, é encontrada uma troca de h por p que melhora a solução corrente seguinte. A troca é

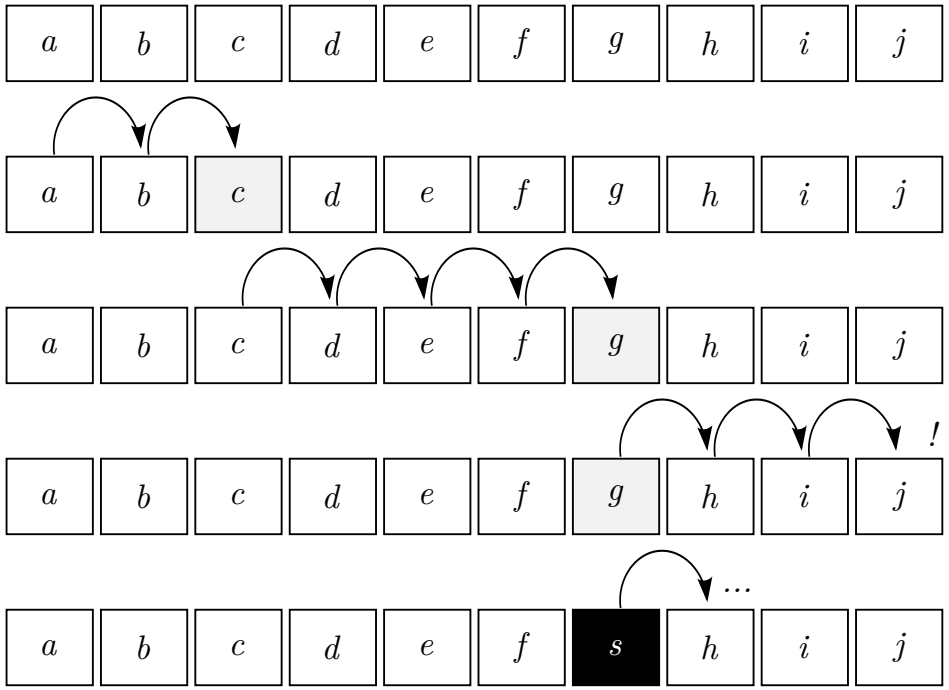


Figura 4.3. Busca local com estratégia *Melhor Aprimorante*, sobre conjunto de arestas E_A

concretizada e prossegue-se na exploração das vizinhanças originadas da troca de cada uma das arestas. A busca local prossegue a partir de p de maneira circular. Assim que se termina de explorar as trocas possíveis para a aresta j, continua-se a partir de a. Chegando em p novamente sem nenhuma melhoria, um ótimo local foi encontrado.

Propõe-se, finalmente, um terceiro algoritmo de busca local, chamado de **Primeiro Aprimorante**. A cada iteração, assim que se encontra uma aresta que melhora o custo ativo esperado da solução, é realizada uma troca de arestas e essa solução se torna a solução corrente. Ou seja, testa-se para cada aresta do conjunto E_A todas as trocas possíveis até que a primeira troca que reduza o custo da solução seja encontrada.

4.4 Busca Tabu

Nos moldes de busca local descritos na seção anterior, ao se tentar encontrar uma solução S' a partir de uma solução inicial S movendo-se entre soluções vizinhas, o valor da solução vai decrescendo até encontrar um mínimo local. Essa é uma solução a partir da qual não se consegue mais encontrar um vizinho que melhore o custo que se detém no momento. Em tais circustâncias, é necessário encontrar uma forma de sair do mínimo local. A estratégia proposta neste trabalho como forma de contornar esse

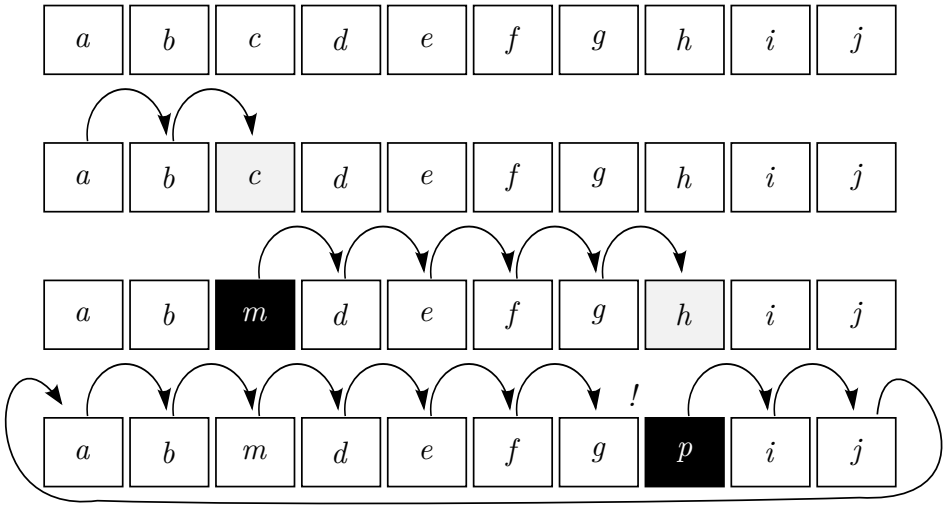


Figura 4.4. Busca local com estratégias de *Primeiro Aprimorante* ou *Melhor Aprimorante por Aresta*, sobre conjunto de arestas E_A. Para a primeira estratégia, m representa a primeira aresta encontrada que substitui c e melhora o custo ativo esperado da solução. Para a segunda estratégia, m representa a aresta que melhor substitui c, dentre todas as possíveis arestas que possam reconectar as sub-árvores conectadas por c.

problema, no âmbito do PMST, consiste da utilização de uma heurística de Busca Tabu. A primeira definição de busca Tabu foi feita em Glover [1986], podendo também ser consultada em seus desdobramentos e mecanismos adicionais em Glover [1989, 1990].

A Busca Tabu permite que se explorem soluções que não reduzam o custo ativo esperado da árvore geradora. Isto se torna possível, pois é mantido um registro das últimas soluções encontradas em termos das trocas de arestas realizadas para transformar uma solução em outra – seria computacionalmente inviável manter um registro de todas as soluções geradas. Sempre que uma operação de troca é realizada, por exemplo, considera-se que o retorno da aresta removida para a solução é Tabu por um certo número T de iterações. Quando uma troca é considerada Tabu, qualquer movimento na vizinhança que a realize é proibido enquanto este número de iterações não é alcançado. A este parâmetro T dá-se o nome de tamanho da lista[5] Tabu.

Quando qualquer um dos algoritmos de busca local sugeridos na seção 4.3 é adaptado ao algoritmo de Busca Tabu, é permitido que se realize uma troca que resulte em um aumento do custo da solução atual, caso todas as arestas sejam testadas e nenhuma delas seja capaz de aprimorar a solução corrente. Para que o efeito de piora na solução seja mínimo, sempre que a remoção de uma aresta é testada, mesmo que não se encontre uma outra aresta que quando inserida melhore a solução, armazena-se aquela aresta que resulta no menor aumento do custo ativo esperado dentre todas as

[5]Também designado como *Tabu tenure*.

possíveis trocas de arestas. Após todos os testes de remoção e inserção, caso nenhuma troca que melhore a solução seja encontrada, a aresta é utilizada. Este procedimento de aceitar uma solução pior é fundamental para que se amplie o espaço de busca, sendo implementado de maneira semelhante para os três algoritmos de busca local. A principal diferença entre os métodos está justamente no critério de aceitação de uma solução melhor.

Definidos esses conceitos, parte-se para a explicação do algoritmo 4.4 que descreve uma busca Tabu adaptada ao problema da árvore geradora mínima probabilística.

Algoritmo 4.4: Busca Tabu adaptada ao PMST

Entrada: Árvore inicial A, gerada pela heurística construtiva
Saída: Árvore A_{melhor}, caso encontrada
1: $A_{melhor} \leftarrow A$
2: **configura** $ListaTabu_{rem}$ ◁ arestas que não podem voltar para solução A
3: **configura** $ListaTabu_{ins}$ ◁ arestas que não podem sair da solução A
4: $A' \leftarrow$ **PesquisarVizinhança**(A) ◁ uma das três estratégias da busca local
5: $A \leftarrow A'$ ◁ A' não necessariamente é melhor que A
6: **se** $custoAtivoEsperado(A') < custoAtivoEsperado(A_{melhor})$ **então**
7: $\quad A_{melhor} \leftarrow A'$ ◁ melhor solução encontrada
8: **fim se**

O algoritmo recebe como entrada uma árvore inicial A e espera-se que ele seja capaz de retornar uma árvore de custo ativo esperado menor, a melhor que conseguir encontrar dentro dos critérios de parada definidos, denominada A_{melhor}. Neste trabalho foram estabelecidos dois critérios, o primeiro é tempo e o segundo é a quantidade de iterações da busca Tabu que são realizados. No algoritmo apresentado, utilizamos duas listas Tabu. A primeira delas, $ListaTabu_{rem}$, mantém um registro de arestas que saíram da solução e da iteração correspondente. A segunda lista, $ListaTabu_{ins}$, memoriza as arestas que entraram na solução e em qual iteração esta operação ocorreu. Foi implementada, também, uma versão que utiliza apenas uma lista, a $ListaTabu_{rem}$. Os resultados experimentais foram devidamente documentados e são apresentados na seção 4.5.1.

Uma configuração fundamental para as listas Tabu é o seu tamanho T. O tamanho deve ser escolhido com cuidado, pois se o valor de T for muito pequeno, a ocorrência de ciclos é mais susceptível, podendo, com frequência, prender o algoritmo em mínimos locais. Uma lista mais restritiva com um valor de T muito grande, por sua vez, pode impedir a exploração de uma grande parte do espaço de busca. Com isto, visitam-se poucas soluções e podem-se obter, frequentemente, soluções de baixa qualidade. Esse conceito se torna ainda mais importante quando se avalia que

tanto os algoritmos de busca local propostos quanto a busca Tabu apresentada para o PMST são determinísticos. Portanto, é crucial que se tenha um valor de T bem configurado para se evitar ciclos ou recorrência de mínimos locais. Quando se utiliza duas listas Tabu deve-se ter atenção ao escolher o tamanho das listas. Um valor de T para $ListaTabu_{rem}$ geralmente é maior que o valor de T para $ListaTabu_{ins}$. Um valor de T muito alto (ou próximo de n) para a segunda lista (arestas da árvore temporariamente proibidas de sair da solução) restringe demais as opções de troca de arestas pertencentes à solução, o que tende a prendê-la em um mínimo local. Já a quantidade de arestas fora de uma determinada solução é muito maior, o que permite um valor de T maior para $ListaTabu_{rem}$.

Inicia-se a busca local, conforme um dos três algoritmos propostos, sempre armazenando a melhor árvore encontrada até o momento. Para entender melhor o procedimento de busca local a partir de uma árvore A, recorre-se ao algoritmo 4.5, que descreve um procedimento de pesquisa de vizinhança através do algoritmo *Melhor Aprimorante*.

Algoritmo 4.5: PesquisarVizinhança(A), estratégia *Melhor Aprimorante*

Entrada: Árvore A
Saída: Árvore A', vizinha de A

```
1:  para cada aresta e_rem ∈ A tal que e_rem ∉ ListaTabu_rem faça
2:    A' ← A − e_rem ◁ A é dividida em duas sub-árvores
3:    para cada aresta e_ins que reconecte as duas sub-árvores faça
4:      se e_ins ∉ ListaTabu_ins então
5:        A' ← A' + e_ins
6:        se melhor forma de reconectar as sub-árvores foi encontrada então
7:          e_rem_melhor ← e_rem
8:          e_ins_melhor ← e_ins
9:        fim se
10:       A' ← A' − e_ins ◁ para tentar outras reconexões
11:     fim se
12:   fim para
13: fim para
14: ListaTabu_rem ← ListaTabu_rem + e_rem_melhor ◁ Conforme critério
15: ListaTabu_ins ← ListaTabu_ins + e_ins_melhor ◁ Conforme critério
16: A' ← A − e_rem_melhor
17: A' ← A' + e_ins_melhor
18: retorne A'
```

No algoritmo 4.5, entre as linhas 1 e 13, testa-se a remoção individual de cada uma das arestas e_{rem} pertencentes a A e as prováveis substituições por e_{ins}. Arestas pertencentes às listas Tabu não são consideradas na pesquisa de vizinhança. A me-

lhor troca possível é identificada por $e_{rem_{melhor}}$ e $e_{ins_{melhor}}$ e é realizada quando toda a vizinhança é pesquisada. Nesse ponto, outro critério de configuração da lista Tabu é pertinente, destacado nas linhas 14 e 15 do algoritmo. Este critério define quando uma aresta deve passar a pertencer às listas Tabu. Se a lista é considerada como um mecanismo para não criar ciclos, pode-se considerar que apenas elementos de soluções encontradas em movimentos não aprimorantes devem entrar na lista Tabu. Se o objetivo da lista Tabu é guiar a busca local por regiões diferentes do espaço de busca, pode-se considerar que qualquer que seja o movimento realizado, seja na direção de melhorar ou de piorar a solução, este movimento deveria entrar na lista Tabu.

Um aspecto importante a respeito da implementação da busca Tabu diz respeito à maneira que a lista é codificada e acessada. Apesar de ser chamada de lista, a estrutura de dados empregada é uma matriz $n \times n$, tal que uma posição $[i, j]$ indica em qual iteração determinada aresta (i, j) passou a integrar ou foi removida da solução (de acordo com a lista utilizada). Supõe-se o exemplo da figura 4.3, no momento em que se testaria a remoção da aresta c e todas as reconexões das duas sub-árvores resultantes que fossem permitidas. Supõe-se, também, que a iteração naquele momento é it, e existe uma aresta z que pertence à lista $ListaTabu_{rem}$, tendo sido retirada da solução (e consequentemente inserida na $ListaTabu_{rem}$), em uma iteração it_z. Uma solução que conte com a troca de c por z só será avaliada se $it - it_z > T$. Essa verificação é praticamente instantânea e dispensa a utilização de uma estrutura de lista de verdade, o que ocasionaria problemas de desempenho. O conceito de lista Tabu, por fim, é basicamente um mecanismo que informa se determinada aresta é ou não Tabu, pergunta respondida com base na iteração atual e em qual iteração a aresta se tornou Tabu.

4.5 Experimentos Computacionais

Devido à variedade de configurações possíveis para a busca Tabu e os tipos de algoritmos de busca local apresentados, foram realizados alguns testes para definir uma combinação de algoritmo de busca local e configurações da busca Tabu, para que fosse possível avaliar o seu desempenho e qualidade como um todo. Vistas as diversas possibilidades, seria impraticável avaliar detalhadamente todo conjunto de configurações possíveis, por isto a escolha de um conjunto de parâmetros.

Os experimentos computacionais sobre um conjunto de 10 instâncias de $n = 22$ a 229 nós do Problema do Caixeiro Viajante [TSPLIB, 2008] foram organizados em duas etapas. Todas as instâncias correspondem a grafos completos. A máquina utilizada

para execução é a mesma definida na seção 3.3.

1. **Primeira etapa:** Escolha da melhor quantidade de listas empregadas na Busca Tabu e do critério de inserção em cada uma das listas.

   - A quantidade de listas pode ser 1 ou 2. No primeiro tipo, uma lista armazena as arestas Tabu que não podem retornar à solução por um determinado número de iterações. No outro, uma segunda lista é acrescentada, com o objetivo de armazenar as arestas Tabu que não podem sair da solução por certa quantidade de iterações.
   - O critério de inserção em cada uma das listas define quando cada aresta se torna Tabu, podendo ser quando uma solução vizinha é piorada ou sempre que uma iteração é concluída.

2. **Segunda etapa:** Escolha do melhor tamanho de lista Tabu a ser empregado (parâmetro T), juntamente com o melhor algoritmo de busca local para a quantidade de listas e critérios selecionados na etapa anterior. As estratégias de busca local, definidas na seção 4.3, são:

   - *Melhor Aprimorante*, que a cada movimento de busca local escolhe a troca de arestas que melhor reduz o valor da solução corrente, dentre todas as arestas existentes na árvore.
   - *Melhor Aprimorante por Aresta*, que a cada movimento de busca local escolhe a melhor aresta que substitui uma aresta específica que se deseja remover e que reduz o valor da solução corrente.
   - *Primeiro Aprimorante*, que a cada movimento de busca local escolhe a primeira aresta que substitui uma aresta específica que se deseja remover e que reduz o valor da solução corrente.

   Para os três algoritmos de busca local, caso não seja encontrada uma solução que melhore a solução atual, é escolhida aquela que resulta no menor acréscimo ao custo da solução corrente.

Uma vez escolhida a melhor combinação de busca local com parâmetros de Busca Tabu, dois tipos de experimentos foram realizados. O primeiro, apresentado na subseção 4.5.3, compara o desempenho do algoritmo selecionado com os resultados apresentados na seção 3.3, referentes à solução do modelo em Programação Linear Inteira por um resolvedor. O segundo avalia a Busca Tabu com relação ao limite inferior da

seção 3.2, para instâncias de até 200 nós. O motivo da escolha de tal limite inferior foi justificado na seção 3.3.

Os testes são baseados em instâncias de grafos completos também disponíveis em TSPLIB [2008]. Foram utilizadas 35 instâncias de 14 a 200 nós, escolhidas arbitrariamente. Considerou-se que todos os nós possuíam a mesma probabilidade de estarem ativos, sendo realizados testes para probabilidades iguais a 0.30, 0.50 e 0.80. A máquina utilizada para os experimentos é a mesma descrita na seção 3.3.

4.5.1 Escolha da Busca Local e das Configurações da Busca Tabu

Nas etapas de teste para escolha de busca local e configurações da busca Tabu empregou-se $p = 0.30$ e tamanho de lista Tabu $T = n$ para $ListaTabu_{rem}$ e $T = \frac{n}{2}$ para $ListaTabu_{ins}$. Os parâmetros, as instâncias e a probabilidade dos nós foram escolhidos arbitrariamente. Foram feitas execuções de 10 minutos para cada algoritmo de busca local.

Escolheu-se o algoritmo de busca local *Primeiro Aprimorante* arbitrariamente para a **primeira etapa** dos testes . O objetivo deste teste é detectar, para um algoritmo qualquer de busca Tabu, se é melhor utilizar uma ou duas listas Tabu e qual o melhor critério de inserção nas listas Tabu. Os critérios, conforme uma ou duas listas, são:

- Critérios para a lista $ListaTabu_{rem}$:
  - Uma aresta e_{rem} é inserida em $ListaTabu_{rem}$ se a troca por e_{ins} resultou em uma variação nula ou positiva do custo ativo esperado da solução corrente. Este critério será identificado por "LTR-MI".
  - Uma aresta e_{rem} é sempre inserida em $ListaTabu_{rem}$ quando substituída por e_{ins}, a despeito da variação do custo ativo esperado da solução corrente. Este critério será identificado por "LTR-S".
- Critérios para a lista $ListaTabu_{ins}$:
  - Uma aresta e_{ins} é inserida em $ListaTabu_{ins}$ se a substituição de e_{rem} resultou em uma variação nula ou positiva do custo ativo esperado da solução corrente. Este critério será identificado por "LTI-MI".
  - Uma aresta e_{ins} é sempre inserida em $ListaTabu_{ins}$ na substituição de e_{rem}, a despeito da variação do custo ativo esperado da solução corrente. Este critério será identificado por "LTI-S".

A combinação desses critérios estabelece dois tipos de critério quando se usa uma lista Tabu, e quatro tipos de critério quando se usam duas listas. O resultado das execuções se encontra na tabela 4.1.

Tabela 4.1. Resultados da primeira etapa de testes, para escolha do critério de inserção nas listas tabu e a quantidade de listas a ser utilizada, que comparam o custo ativo esperado $E[L_T]$ de cada solução. Para cada instância, destaca-se o melhor custo ativo esperado dentre os critérios avaliados. Para cada um dos outros critérios, é apresentada a diferença percentual para o melhor custo.

		Uma lista Tabu		Duas listas Tabu			
		LTR-S	LTR-MI	LTR-S LTI-MI	LTR-MI LTI-S	LTR-MI LTI-MI	LTR-S LTI-S
instância	n	$E[L_T]$	$E[L_T]$	$E[L_T]$	$E[L_T]$	$E[L_T]$	$E[L_T]$
uly22	22	**25.01**	**25.01**	**25.01**	**25.01**	**25.01**	0.06%
eil51	51	1.06%	0.31%	**215.16**	1.23%	0.34%	1.64%
st70	70	1.76%	1.23%	1.81%	2.80%	**362.52**	3.97%
rat99	99	0.68%	**698.57**	2.03%	2.31%	**698.57**	3.44%
pr124	124	0.40%	<0.01%	0.40%	2.45%	**38406.40**	2.81%
gr137	137	1.47%	**391.02**	1.15%	2.41%	0.02%	3.32%
ch150	150	1.30%	0.03%	2.80%	3.37%	**3924.34**	4.09%
u159	159	1.04%	0.20%	1.04%	1.81%	**25494.60**	2.59%
d198	198	1.13%	**7825.28**	1.13%	2.87%	**7825.28**	3.53%
gr229	229	1.38%	**849.10**	1.38%	2.60%	**849.10**	2.97%

A partir da tabela 4.1 avalia-se que melhores resultados (destacados em negrito) são obtidos ao incluir arestas na lista Tabu somente quando se encontra uma solução vizinha que mantém ou aumenta o custo da solução corrente. A princípio, a inclusão de arestas na lista Tabu para soluções não aprimorantes evita a criação de ciclos, proporcionando maior abrangência de soluções vizinhas no espaço de busca. Nos testes realizados para uma ou duas listas Tabu, percebe-se que a utilização de duas listas garantiu mais resultados melhores do que apenas uma lista. Sendo assim, para as próximas etapas de teste, foi considerado que o critério de inserção de arestas em uma lista Tabu se limita à situação na qual o custo ativo esperado da solução vizinha não melhora o custo ativo esperado da solução corrente. A partir dos resultados, determina-se, também, somente o uso de duas listas Tabu.

Passando-se para a **segunda etapa** dos testes, são comparados os algoritmos *Melhor Aprimorante*, *Melhor Aprimorante por Aresta* e *Primeiro Aprimorante*, utilizando-se duas listas Tabu e o critério de inserção nas listas Tabu determinado pela primeira etapa. Para cada instância, dentre 6 pares de parâmetros T_{rem} e T_{ins}, é escolhido o melhor par para cada algoritmo de busca local, baseado no custo ativo esperado das soluções e, em caso de empate, no tempo necessário para atingí-las (limitado a 10

minutos por instância). Os resultados são dispostos nas tabelas 4.2 e 4.3, que respectivamente apresentam os custos ativos esperados e tempos Os parâmetros T_{rem} e T_{ins} indicam os respectivos tamanhos das listas $ListaTabu_{rem}$ e $ListaTabu_{ins}$.

Tabela 4.2. Refinamento da segunda etapa de testes, que compara as melhores configurações de cada algoritmo de busca local em termos de custo ativo esperado ($E[L_T]$) de cada solução encontrada. Apresenta-se o melhor custo ativo dentre todas as configurações para cada instância. Para cada uma das outras configurações, é apresentada a diferença percentual para o melhor custo. Em caso de empate, deve-se recorrer à tabela 4.3, que compara o tempo necessário em cada configuração para se atingir determinada solução.

			$T_{rem}=n$ $T_{ins}=\frac{n}{2}$	$T_{rem}=\frac{n}{2}$ $T_{ins}=\frac{n}{3}$	$T_{rem}=\frac{n}{2}$ $T_{ins}=\frac{n}{6}$	$T_{rem}=n$ $T_{ins}=\frac{n}{3}$	$T_{rem}=n$ $T_{ins}=\frac{n}{9}$	$T_{rem}=\frac{n}{2}$ $T_{ins}=\frac{n}{9}$
alg.	instância	n	$E[L_T]$	$E[L_T]$	$E[L_T]$	$E[L_T]$	$E[L_T]$	$E[L_T]$
MA	uly22	22	0.03%	0.40%	0.43%	**25.01**	**25.01**	0.40%
	eil51	51	0.40%	0.16%	**214.79**	0.40%	0.40%	0.04%
	st70	70	**361.69**	**361.69**	**361.69**	**361.69**	**361.69**	**361.69**
	rat99	99	0.30%	**693.88**	0.30%	0.30%	0.30%	0.30%
	pr124	124	**38375.70**	**38375.70**	**38375.70**	**38375.70**	**38375.70**	**38375.70**
	gr137	137	**389.96**	**389.96**	**389.96**	**389.96**	**389.96**	**389.96**
	ch150	150	**3916.44**	**3916.44**	**3916.44**	**3916.44**	**3916.44**	**3916.44**
	u159	159	**25334.3**	**25334.3**	**25334.3**	**25334.3**	**25334.3**	**25334.3**
	d198	198	**7835.57**	**7835.57**	**7835.57**	**7835.57**	**7835.57**	**7835.57**
	gr229	229	**856.63**	**856.63**	**856.63**	**856.63**	**856.63**	**856.63**
MAA	uly22	22	**25.01**	**25.01**	**25.01**	**25.01**	**25.01**	**25.01**
	eil51	51	0.44%	**214.87**	0.44%	0.28%	0.32%	**214.87**
	st70	70	**361.69**	**361.69**	**361.69**	**361.69**	**361.69**	**361.69**
	rat99	99	1.54%	0.35%	**698.58**	0.93%	1.14%	0.56%
	pr124	124	**38401.70**	**38401.70**	**38401.70**	**38401.70**	**38401.70**	**38401.70**
	gr137	137	**391.42**	**391.42**	**391.42**	**391.42**	**391.42**	**391.42**
	ch150	150	**3913.05**	**3913.05**	**3913.05**	**3913.05**	**3913.05**	**3913.05**
	u159	159	**25453.50**	**25453.50**	**25453.50**	**25453.50**	**25453.50**	**25453.50**
	d198	198	**7821.75**	**7821.75**	**7821.75**	**7821.75**	**7821.75**	**7821.75**
	gr229	229	**850.82**	**850.82**	**850.82**	**850.82**	**850.82**	**850.82**
PA	uly22	22	**25.01**	**25.01**	**25.01**	**25.01**	**25.01**	**25.01**
	eil51	51	0.54%	**214.73**	0.03%	0.40%	0.33%	0.04%
	st70	70	**362.52**	0.68%	0.80%	0.62%	1.23%	0.85%
	rat99	99	**698.57**	**698.57**	**698.57**	**698.57**	**698.57**	**698.57**
	pr124	124	**38406.40**	**38406.40**	**38406.40**	**38406.40**	**38406.40**	**38406.40**
	gr137	137	0.29%	**389.94**	0.14%	0.29%	0.29%	0.25%
	ch150	150	0.23%	0.19%	0.23%	**3915.37**	0.23%	0.23%
	u159	159	**25494.60**	**25494.60**	**25494.60**	**25494.60**	**25494.60**	**25494.60**
	d198	198	**7825.28**	**7825.28**	**7825.28**	**7825.28**	**7825.28**	**7825.28**
	gr229	229	**849.10**	**849.10**	**849.10**	**849.10**	**849.10**	**849.10**

Percebe-se, nas tabelas 4.2 e 4.3, que embora o tamanho das listas não tenha interferido significativamente no custo ativo esperado da solução encontrada por um mesmo algoritmo, a escolha do tamanho das listas afetou o tempo necessário para

Tabela 4.3. Refinamento da segunda etapa de testes, que compara as melhores configurações de cada algoritmo de busca local em termos do tempo necessário, em segundos, para encontrar cada solução correspondente da tabela 4.2. Na avaliação das configurações, o tempo é utilizado como critério de desempate para soluções de mesmo custo ativo ($E[L_T]$). Para cada instância, o melhor tempo é disposto em segundos. Para cada uma das outras configurações, apresenta-se a diferença percentual do tempo encontrado na configuração para o melhor tempo.

			$T_{rem} = n$ $T_{ins} = \frac{n}{2}$	$T_{rem} = \frac{n}{2}$ $T_{ins} = \frac{n}{3}$	$T_{rem} = \frac{n}{2}$ $T_{ins} = \frac{n}{6}$	$T_{rem} = n$ $T_{ins} = \frac{n}{3}$	$T_{rem} = n$ $T_{ins} = \frac{n}{9}$	$T_{rem} = \frac{n}{2}$ $T_{ins} = \frac{n}{9}$
alg.	instância	n	$t(s)$	$t(s)$	$t(s)$	$t(s)$	$t(s)$	$t(s)$
MA	uly22	22	1280.00%	3100.00%	**0.05**	89320.00%	737800.00%	8380.00%
	eil51	51	2.12%	16746.56%	17570.90%	**1.89**	3.17%	31340.74%
	st70	70	2.84%	1.79%	2.54%	**6.70**	1.94%	1.34%
	rat99	99	2.69%	859.31%	0.34%	**26.81**	0.11%	0.63%
	pr124	124	1.56%	0.95%	0.95%	**68.42**	0.01%	0.44%
	gr137	137	2.17%	0.73%	1.05%	0.25%	**173.12**	0.52%
	ch150	150	1.28%	0.06%	0.41%	0.17%	**142.63**	0.45%
	u159	159	1.65%	0.12%	0.39%	**190.71**	0.10%	0.02%
	d198	198	1.27%	0.32%	0.41%	0.12%	**591.62**	0.29%
	gr229	229	1.49%	0.27%	0.33%	**588.93**	0.34%	0.03%
MAA	uly22	22	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**
	eil51	51	**0.89**	7073.03%	50398.88%	22882.02%	1159.55%	33550.56%
	st70	70	1.35%	**0.74**	**0.74**	8.11%	**0.74**	1.35%
	rat99	99	**9.72**	3758.02%	5563.68%	4717.18%	1175.41%	5787.65%
	pr124	124	1.51%	0.46%	**54.31**	1.23%	0.59%	0.94%
	gr137	137	1.11%	**49.39**	0.34%	1.54%	0.59%	3.12%
	ch150	150	0.71%	0.24%	**21.24**	0.47%	0.71%	0.89%
	u159	159	0.62%	0.05%	**81.25**	0.23%	1.01%	1.22%
	d198	198	0.82%	0.03%	**196.16**	0.41%	0.44%	0.41%
	gr229	229	0.94%	**165.77**	0.14%	0.02%	0.44%	0.16%
PA	uly22	22	100.00%	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**
	eil51	51	**1.07**	2495.33%	35978.50%	22523.36%	36206.54%	34935.51%
	st70	70	47096.30%	13658.02%	53232.10%	60746.91%	**0.81**	46898.77%
	rat99	99	**5.43**	1.66%	1.84%	1.29%	1.66%	1.66%
	pr124	124	**62.55**	0.77%	0.59%	1.22%	1.37%	0.77%
	gr137	137	**57.42**	629.69%	526.49%	1.04%	0.92%	575.01%
	ch150	150	**69.98**	455.20%	0.89%	604.50%	1.77%	1.96%
	u159	159	**136.43**	1.03%	0.60%	1.36%	1.57%	1.93%
	d198	198	**240.17**	0.81%	1.57%	1.23%	1.12%	0.91%
	gr229	229	**247.73**	0.16%	0.57%	0.75%	0.75%	0.80%

se encontrar cada solução. Avaliando os resultados dos testes, a melhor configuração encontrada para cada algoritmo foi:

- Melhor Aprimorante: $T_{rem} = n$ e $T_{ins} = \frac{n}{3}$;
- Melhor Aprimorante por Aresta: $T_{rem} = \frac{n}{2}$ e $T_{ins} = \frac{n}{6}$;
- Primeiro Aprimorante: $T_{rem} = n$ e $T_{ins} = \frac{n}{2}$.

Para facilitar a visualização dos resultados, as tabelas 4.4 e 4.5 comparam cada um dos algoritmos com a sua melhor configuração. Nota-se que a busca local *Melhor Aprimorante* é a que detém a maioria dos melhores resultados, sendo, então, o método escolhido para os experimentos computacionais para a busca Tabu. É importante lembrar que, em outos cenários ou outras instâncias, seria possível ter desempenhos diferentes entre os algoritmos de busca local. O intuito destas etapas de teste é escolher um algoritmo para avaliar, através de experimentos empíricos, a utilização de um algoritmo de busca Tabu em comparação à resolução do problema através de uma modelagem exata.

Tabela 4.4. Refinamento da segunda etapa de testes, que compara as melhores configurações de cada algoritmo de busca local em termos de custo ativo esperado ($E[L_T]$) de cada solução encontrada. Em caso de empate, recorre-se à tabela 4.5, que compara os tempos necessários para se atingir cada solução.

		Melhor Aprimorante	Melhor Aprimorante por Aresta	Primeiro Aprimorante
		$T_{rem} = n\ T_{ins} = \frac{n}{3}$	$T_{rem} = \frac{n}{2}\ T_{ins} = \frac{n}{6}$	$T_{rem} = n\ T_{ins} = \frac{n}{2}$
instância	n	$E[L_T]$	$E[L_T]$	$E[L_T]$
uly22	22	**25.01**	**25.01**	**25.01**
eil51	51	**215.65**	0.08%	0.11%
st70	70	**361.69**	**361.69**	0.23%
rat99	99	**695.98**	0.37%	0.37%
pr124	124	**38375.70**	0.07%	0.08%
gr137	137	**389.96**	0.37%	0.29%
ch150	150	0.09%	**3913.05**	0.29%
u159	159	**25334.30**	0.47%	0.63%
d198	198	0.18%	**7821.75**	0.05%
gr229	229	0.89%	0.20%	**849.10**

4.5.2 Comparação dos Resultados da Busca Tabu com a Resolução do Modelo Exato

Nesta seção, são comparados os resultados obtidos pela busca Tabu com o algoritmo de busca local *Melhor Aprimorante* nas configurações escolhidas através da primeira e segunda etapas de testes com aqueles obtidos na seção 3.3 para as mesmas instâncias e probabilidades.

O tempo de execução necessário para a busca Tabu não foi apresentado na tabela 4.6, pois, mesmo somado ao tempo para se construir a solução inicial, não chegou a atingir 50ms para atingir o valor apresentado – independente da instância e da probabilidade. O algoritmo ainda foi executado através de um critério de parada de

Tabela 4.5. Refinamento da segunda etapa de testes, que compara as melhores configurações de cada algoritmo de busca local em função do tempo necessário, em segundos, para encontrar cada solução correspondente da tabela 4.4. Para cada instância, o melhor tempo é disposto em segundos. Para cada uma das outras configurações, apresenta-se a diferença percentual do tempo encontrado na configuração para o melhor tempo.

		Melhor Aprimorante	Melhor Aprimorante por Aresta	Primeiro Aprimorante
		$T_{rem} = n\ T_{ins} = \frac{n}{3}$	$T_{rem} = \frac{n}{2}\ T_{ins} = \frac{n}{6}$	$T_{rem} = n\ T_{ins} = \frac{n}{2}$
instância	n	$t(s)$	$t(s)$	$t(s)$
uly22	22	223450.00%	**0.02**	**0.02**
eil51	51	76.64%	41903.74%	**1.07**
st70	70	805.41%	**0.74**	51560.81%
rat99	99	393.74%	10038.31%	**5.43**
pr124	124	25.98%	**54.31**	15.17%
gr137	137	250.20%	**49.56**	15.86%
ch150	150	572.65%	**21.24**	229.47%
u159	159	134.72%	**81.25**	67.91%
d198	198	201.97%	**196.16**	22.44%
gr229	229	254.78%	**166.00**	49.23%

4000 iterações ou 1h, mas nenhuma solução melhor foi encontrada. Percebe-se que valores pequenos de probabilidade não influenciam o tempo de execução das heurísticas, ao contrário da resolução do modelo exato, que é bastante sensível a essa variável. Esse é um ponto positivo para a busca Tabu que superou o CPLEX em todos os tempos de execução. Um aspecto interessante é que o algoritmo utilizado ainda foi capaz de encontrar as mesmas soluções que o resolvedor encontrou, algumas delas ótimas e uma delas até melhor, tendo em vista que a execução do resolvedor foi limitada a 2 horas de execução.

4.5.3 Análise dos Resultados para Instâncias Grandes

A busca Tabu, com o algoritmo de busca local *Melhor Aprimorante*, foi executada para diversas instâncias de TSPLIB [2008] e os custos ativos esperados das soluções foram comparados ao limite inferior L_{AGM}. Para esses experimentos, limitou-se a execução para cada instância em 1 hora ou 4000 iterações da busca local, sem que houvesse melhoria da solução corrente.

Observa-se na tabela 4.7 que o intervalo entre o limite inferior L_{AGM} e o custo ativo esperado $E[L_T]$ encontrado pela metaheurística é menor para probabilidades maiores. Isto acontece pois o limite é baseado em uma árvore geradora mínima e o PMST se aproxima desse problema à medida que a probabilidade dos nós aumenta.

Tabela 4.6. Resultados da execução da busca Tabu, com algoritmo de busca local Melhor Aprimorante, comparados às soluções encontradas pelo resolvedor da modelagem exata.

Instâncias			Resultado Resolvedor		Limite Inferior	Busca Tabu Melhor Aprimorante		
p	instância	n	$E[L_T]$	gap(%)	L_{AGM}	$E[L_T]$	diferença L_{AGM}(%)	diferença resolv.(%)
0.30	NL4	4	218.30	0.00	213.26	218.30	2.36	0.00
	NL6	6	574.30	0.00	449.24	574.30	27.84	0.00
	NL8	8	797.96	0.00	592.43	797.96	34.69	0.00
	NL10	10	1020.08	18.85	719.16	1020.08	41.84	0.00
	NL12	12	1591.76	26.70	1167.74	1591.76	36.31	0.00
	NL14	14	2468.06	42.83	1563.01	2468.06	57.90	0.00
	NL16	16	2903.96	50.56	1586.93	2903.96	82.99	0.00
0.50	NL4	4	483.38	0.00	473.38	483.38	2.11	0.00
	NL6	6	1075.63	0.00	871.88	1075.63	23.37	0.00
	NL8	8	1351.52	0.00	1067.59	1351.52	26.60	0.00
	NL10	10	1632.21	8.10	1246.56	1632.21	30.94	0.00
	NL12	12	2490.31	23.61	1984.53	2490.31	25.49	0.00
	NL14	14	3671.36	35.59	2630.18	3671.36	39.59	0.00
	NL16	16	4071.57	40.65	2657.42	3968.91	49.35	-2.52
0.80	NL4	4	868.92	0.00	858.68	868.92	1.19	0.00
	NL6	6	1582.30	0.00	1439.54	1582.30	9.92	0.00
	NL8	8	1892.61	0.00	1721.58	1892.61	9.93	0.00
	NL10	10	2217.68	0.00	1998.40	2217.68	10.97	0.00
	NL12	12	3448.18	18.76	3176.80	3448.18	8.54	0.00
	NL14	14	4767.82	27.62	4208.80	4767.82	13.28	0.00
	NL16	16	4875.67	40.63	4252.00	4875.67	14.67	0.00

Tabela 4.7. Resultados computacionais para a execução da busca Tabu em instâncias de 14 a 200 nós, apresentando o valor do limite inferior L_{AGM} e o custo ativo esperado ($E[L_T]$) para cada solução encontrada pela busca Tabu.

		$p = 0.30$			$p = 0.50$			$p = 0.80$		
inst.	n	L_{AGM}	$E[LT]$	dif. L_{AGM} (%)	L_{AGM}	$E[LT]$	dif. L_{AGM} (%)	L_{AGM}	$E[LT]$	dif. L_{AGM} (%)
bur14	14	6.46654	10.35	37.51	10.88	14.70	25.95	17.41	19.25	9.52
uly16	16	14.3224	21.79	34.27	23.98	32.17	25.45	38.38	43.01	10.77
ulys22	22	14.8018	25.01	40.82	24.68	34.45	28.35	39.49	44.36	10.96
att48	48	8293.1	16597.60	50.03	13821.80	20985.70	34.14	22114.90	25482.20	13.21
eil51	51	112.947	215.65	47.62	188.25	280.97	33.00	301.19	347.44	13.31
ber52	52	1824.49	3439.89	46.96	3040.82	4489.46	32.27	4865.30	5571.93	12.68
st70	70	169.862	361.69	53.04	283.10	448.21	36.84	452.97	529.68	14.48
eil76	76	141.699	272.26	47.95	236.17	353.17	33.13	377.87	435.64	13.26
pr76	76	26165.3	57498.00	54.49	43608.90	71069.00	38.64	69774.20	82626.40	15.55
gr96	96	130.868	280.59	53.36	218.11	346.53	37.06	348.98	407.98	14.46
rat99	99	334.419	695.98	51.95	557.37	874.71	36.28	891.78	1042.69	14.47
rd100	100	2088.98	4516.65	53.75	3481.64	5572.23	37.52	5570.62	6535.07	14.76
kroB100	100	5777.09	12946.10	55.38	9628.48	15811.00	39.10	15405.60	18310.20	15.86
kroC100	100	5520.7	12733.80	56.65	9201.16	15370.40	40.14	14721.90	17538.90	16.06
kroD100	100	5579.01	12500.40	55.37	9298.36	15240.90	38.99	14877.40	17640.60	15.66
kroE100	100	5767.32	12874.00	55.20	9612.20	15586.30	38.33	15379.50	18151.40	15.27
kroA100	100	5631.65	12932.40	56.45	9386.09	15356.20	38.88	15017.70	17766.00	15.47
eil101	101	168.677	321.26	47.50	281.13	412.97	31.92	449.81	512.63	12.26
lin105	105	3918.13	8769.48	55.32	6530.22	10611.80	38.46	10448.40	12349.30	15.39
pr107	107	10427.2	26354.80	60.44	17378.70	29753.50	41.59	27806.00	33162.00	16.15
pr124	124	15160.6	38375.70	60.49	25267.70	43903.50	42.45	40428.40	48691.50	16.97
bier127	127	28415.3	55963.20	49.23	47358.80	71340.30	33.62	75774.10	87137.80	13.04
ch130	130	1549.22	3485.26	55.55	2582.03	4207.12	38.63	4131.24	4906.94	15.81
pr136	136	26689.9	54256.30	50.81	44483.10	67910.60	34.50	71172.90	82263.00	13.48
gr137	137	175.742	389.96	54.93	292.90	472.55	38.02	468.65	551.70	15.05
pr144	144	14839.3	42755.70	65.29	24732.20	46601.70	46.93	39571.60	48909.40	19.09
ch150	150	1764.29	3916.44	54.95	2940.48	4785.09	38.55	4704.76	5588.42	15.81
kroA150	150	7067.22	16106.10	56.12	11778.70	19147.50	38.48	18845.90	22170.40	15.00
kroB150	150	6841.27	15540.00	55.98	11402.10	18641.10	38.83	18243.40	21508.20	15.18
pr152	152	17750.5	48096.40	63.09	29584.20	53098.20	44.28	47334.80	57453.60	17.61
u159	159	11146.4	25334.30	56.00	18577.30	30238.70	38.56	29723.60	35009.70	15.10
rat195	195	649.438	1395.38	53.46	1082.40	1731.50	37.49	1731.83	2044.96	15.31
d198	198	3530.13	7816.01	54.83	5883.56	9378.42	37.26	9413.69	10970.30	14.19
kroA200	200	7779.78	17805.70	56.31	12966.30	21355.30	39.28	20746.10	24675.10	15.92
kroB200	200	7861.1	17785.40	55.80	13101.80	21422.50	38.84	20962.90	24932.90	15.92

Capítulo 5

Conclusão e Trabalhos Futuros

O presente trabalho apresenta diversas constribuições para o Problema da Árvore Geradora Mínima Probabilística, um problema pouco estudado e de difícil resolução. O problema é NP-Difícil e sua aplicação a situações reais resulta em instâncias grandes e complexas. Para solucioná-lo em tempo hábil, torna-se necessária a utilização de heurísticas, cuja utilidade é ainda mais evidente à medida que se utiliza instâncias maiores e tem-se probabilidades menores de os nós estarem ativos. Percebeu-se, através dos diversos experimentos realizados e resultados apresentados, que a probabilidade dos nós estarem ativos não interfere no tempo de execução dos algoritmos de busca local ou Busca Tabu.

Foi apresentado um modelo de Programação Linear Inteira, que poderia ser melhorado de duas formas. A primeira delas seria tentar aperfeiçoar as duas últimas restrições, que tratam dos cortes na árvore geradora e prevenção de ciclos. A segunda seria tentar utilizar a formulação do problema de modelagem de redes para tentar reduzir o tempo de execução do modelo exato no ILOG CPLEX [2010] para instâncias de probabilidades muito pequenas. De qualquer forma, o papel do modelo é prover uma base comparativa para a resolução do problema através de heurísticas. Embora o problema já tenha sido investigado através de algoritmos genéticos, não foi possível comparar os resultados obtidos devido à impossibilidade de conseguirmos as mesmas instâncias.

A contribuição mais importante deste trabalho está relacionada à estratégia de avaliação do custo ativo esperado de uma árvore no momento em que se realiza busca local. A atualização eficiente dos coeficientes k utilizados para o cálculo do custo ativo esperado das árvores é de fundamental importância para o emprego de busca local, que foi adaptado a uma heurística de busca Tabu. Este trabalho também é inédito no que se diz respeito à utilização de busca local para o problema da árvore geradora

mínima probabilística. Além de terem sido apresentados três diferentes algoritmos de busca local, embora baseados no mesmo conceito, incorporam-se estes algoritmos em uma heurística de Busca Tabu.

Como trabalho futuro sugere-se melhorar ainda mais a avaliação do custo das árvores durante a busca local, através da utilização de fórmulas não só para a menor das sub-árvores envolvidas na remoção de uma aresta, mas também para a sub-árvore maior – atualmente avaliada através da lista de adjacência dos nós. A possibilidade de se definir uma formulação para a influência da troca de uma aresta por outra nas sub-árvores reconectadas está sendo atualmente investigada. É possível que se obtenha uma melhora no tempo de execução das heurísticas com a utilização uma variável s_{ij}, com a mesma finalidade na modelagem exata, para identificar a qual das duas sub-árvores se refere o valor de k de uma determinada aresta. Esse procedimento pode facilitar na avaliação dos coeficientes k de uma árvore e a posterior computação de seu custo ativo esperado.

Seria interessante, também, utilizar heurísticas não determinísticas, como por exemplo uma heurística baseada em GRASP [Feo & Resende, 1995]. Assim seria possível comparar os resultados obtidos com aqueles já encontrados através da utilização de busca Tabu.

Referências Bibliográficas

Abuali, F. N.; Schoenefeld, D. A. & Wainwright, R. L. (1994). Designing telecommunications networks using genetic algorithms and probabilistic minimum spanning trees. In *Proceedings of the 1994 ACM Symposium on Applied Computing*, pp. 242–246, Phoenix.

Abuali, F. N.; Wainwright, R. L. & Schoenefeld, D. A. (1995). Determinant factorization: A new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem. In *Proceedings of The 6th Intl. Conf. on Genetic Algorithms (ICGA-95)*, pp. 470–477, Pittsburgh.

Balaprakash, P.; Birattari, M. & Stützle, T. (2007). Engineering stochastic local search algorithms: A case study in estimation-based local search for the probabilistic traveling salesman problem. Technical Report Series TR/IRIDIA/2007-022, IRIDIA.

Bertsimas, D. J. (1988a). *Probabilistic Combinatorial Optimization Problems*. Technical report no. 194, Massachusetts Institute of Technology, Operations Research Center. Ph.D. Thesis.

Bertsimas, D. J. (1988b). The probabilistic minimum spanning tree, part i: Complexity and combinatorial properties. Working Paper Rm E53-359, Massachusetts Institute of Technology, Sloan School of Management.

Bertsimas, D. J. (1988c). The probabilistic minimum spanning tree, part ii: Probabilistic analysis and asymptotic results. Working Paper OR184-88, Massachusetts Institute of Technology, Sloan School of Management.

Bertsimas, D. J. (1990). The probabilistic minimum spanning tree problem. *Networks*, 20:245–275.

Bertsimas, D. J.; Jaillet, P. & Odoni, A. R. (1989). A priori optimization. Sloan W.P. 3059-89-MS, Massachusetts Institute of Technology, Sloan School of Management.

Bondy, J. A. & Murty, U. R. (1976). *Graph Theory with Applications*. MacMillan.

Borüvka, O. (1926). O jistem problemu minimalim. in prace mor. *Prirodoved. Spol. v Brne*, [s.n.]:37–58.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. & Stein, C. (2001). *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2a. edição.

CTT Instances (2010). Challenge traveling tournament instances: a library of sample instances for the ttp. `http://mat.gsia.cmu.edu/TOURN/` (acessado em 31 de abril de 2010).

Feo, T. A. & Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133.

Garey, M. R. & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549.

Glover, F. (1989). Tabu search - part i. *ORSA Journal on Computing*, 1:190–206.

Glover, F. (1990). Tabu search - part ii. *ORSA Journal on Computing*, 2:4–32.

Hwang, F.; Richards, D. & Winter, P. (1992). The steiner tree problem. In *Annals of Discrete Mathematics*, volume 53, North-Holland. Elsevier Science Publishers B.V.

ILOG CPLEX (2010). Ilog inc. `http://www.ilog.com/products/cplex/` (acessado em 31 de abril de 2010).

Ishii, H.; Shiode, H. & Nishida, T. (1981). Stochastic spanning tree problem. *Discrete Applied Mathematics*, 3:263–273.

Jaillet, P. (1985). Probabilistic traveling salesman problems. Technical Report 185, Operations Research Center, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Jaillet, P. & Odoni, A. R. (1988). The probabilistic vehicle routing problem. *Vehicle Routing: Methods and Studies*, pp. 293–318.

Kruskal, J. J. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. ACM*, 7(1):48–50.

Narula, S. C. & Ho, C. A. (1980). Degree-constrained minimum spanning tree. *Computer & Operations Research*, 7:239–249.

Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401.

Souza, R. F. B. & Urrutia, S. (2008). Heurísticas para o problema da Árvore geradora mínima probabilística. In *Anais do XL Simpósio Brasileiro de Pesquisa Operacional (XL SBPO)*, pp. 1344–1353, João Pessoa - PB, Brasil.

Souza, R. F. B. & Urrutia, S. (2010). O problema da Árvore geradora mínima probabilística homogêneo: Modelagem exata e heurísticas. In *Anais do XLII Simpósio Brasileiro de Pesquisa Operacional (XLII SBPO)*, p. [n.d.], Bento Gonçalves - RS, Brasil. Nota: Aceito para publicação nos Anais do XLI SBPO (2009) - Salvador - BA, mas pulicado como Errata nos Anais do XLII SBPO (2010).

TSPLIB (2008). Tsplib, a library of sample instances for the tsp (and related problems) from various sources and of various types. `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/` (acessado em 31 de abril de 2010).

Xu, W. & Gen, M. (1995). On the quadratic minimum spanning tree problem. In *Proceedings of 1995 Japan-China International Workshops on Information Systems*, pp. 141–148, Ashikaga, Japan.

Ziviani, N. (2004). *Projeto de Algoritmos com Implementações em Pascal e C*. Cengage Learning, 2a. edição.