Victor Ferreira Ferrari - 187890
Vinícius Couto Espindola - 188115
TP4 - MC658 - Grupo 2

1. Problema e Métodos Utilizados

O problema de otimização a ser resolvido consiste em encontrar uma árvore geradora mínima com restrição de grau em cada vértice. Este problema, conhecido como "*Degree Constrained Minimum Spanning Tree Problem*" (DCMSTP), pertencente à classe NP-Completo.

O problema foi abordado por dois métodos: o cálculo de limitantes duais e primais através de relaxação e heurística lagrangiana, e uso de heurísticas/metaheurísticas para busca de soluções viáveis. As heurísticas foram implementadas levando em conta que as instâncias de teste consistem de grafos completos e pesos maiores ou iguais a zero.

O método heurístico selecionado foi o de busca local, realizado através da remoção e inserção de arestas (sem perda de viabilidade) em uma solução conhecida. A metaheurística implementada para controlar a busca local foi a Busca Tabu, que atua no bloqueio de arestas recém removidas exigindo um número mínimo de iterações para que sejam re-inseridas.

Definições:

Vagas - soma dos graus livres dos vértices pertencentes a uma componente.

Saturação - dizemos que uma componente está saturada se não houver mais vértices com graus livres contidos nesta componente.

2. Relaxação Lagrangiana

a. Teoria

A relaxação lagrangiana do problema foi feita ao se dualizar as restrições de grau por vértice, gerando um Problema Primal Lagrangiano (PPL) equivalente ao problema de encontrar uma árvore geradora mínima no grafo lagrangiano (grafo com custos lagrangianos):

$$z(\lambda) = min\{ \sum_{e=(i,j) \in E} (c_{ij} + \lambda_i + \lambda_j)x_e \; - \; \sum_{i \in V} \lambda_i d_i \; : \; x \; \in R_{MSTP} \}$$

A solução do PPL é utilizado no método de resolução do Problema Dual Lagrangiano (PDL), que tenta encontrar os melhores multiplicadores de Lagrange λ. Para resolver o PDL, foi utilizado o método do subgradiente.

A implementação do método do subgradiente foi feita de acordo com a descrição e parâmetros de Beasley [1]:

- Defina λ_0 e $0 < \pi \leq 2$.
- Solucione o PPL para λ_i, obtendo uma solução X_j de valor Z_{LB}.
- Defina *subgradientes* $G_i = \sum_{i \in V} X_j - d_i$

- Defina um passo $T = \frac{\pi(Z_{UB} - Z_{LB})}{\sum_{i \in V} G_i^2}$, com Z_{UB} um limitante primal calculado em algum momento.
- Atualize λ_i: $\lambda_i = max(0,\ \lambda_i + TG_i)$

Como definido, o processo continuaria para sempre. Portanto, é necessário verificar condições de parada em algum momento. Diversas condições podem ser utilizadas em momentos diferentes do fluxo de execução, então foi dedicada uma seção para detalhes de implementação.

Há um valor máximo que pode ser alcançado (que pode ser próximo da solução ótima) pelo limitante dual, a partir do qual ele não atualiza. Porém, nem toda iteração do método fornece um limitante dual melhor que os anteriores, com o passo na direção oposta do subgradiente tendendo a atingir um dual ótimo, mas instantaneamente podendo variar.

Para não ficar com o passo muito pequeno, com pequenos incrementos no limitante dual, uma estratégia comum é multiplicar o limitante primal no cálculo do passo por (1+B), para um B pequeno (<0.1).

b. Implementação

Os multiplicadores de Lagrange foram todos iniciados com valor 1. Esse valor foi determinado empiricamente, analisando a qualidade dos limitantes duais e primais fornecidos para cada valor.

O primeiro valor de π é 2, como especificado em [1]. Esse valor, porém, não é estático. A partir de um certo número de iterações em que um dual melhor não é encontrado (30, no caso) o valor é cortado pela metade.

Foi utilizada a estratégia de multiplicação do primal, com B = 0.0005. Para valores maiores, a qualidade dos limitantes encontrados se tornava cada vez menor.

A métrica principal de parada é o tempo atual. É definido um tempo máximo, que se atingido o processo termina, sem outras condições. Além dessa, se π alcança um valor muito pequeno, 0.005, o processo também termina, pois os incrementos no limitante dual ficam muito pequenos em muitas iterações, o que não melhora muito a qualidade do limitante.

Se o limitante dual é igual ao primal, a solução é ótima não há motivos para continuar o processo. Como o valor exato pode não ser atingido, foi utilizada uma distância de 1 para a condição de otimalidade.

A última condição de parada é a verificação de que o limitante dual encontrado é ótimo. Isso pode ser visto no cálculo dos subgradientes. Se, em uma iteração, todos os subgradientes têm valor 0, os multiplicadores não são atualizados e assim não há dual melhor a ser obtido. Se a solução dual é viável, é a solução ótima [1]. Um subgradiente negativo pode ser transformado em zero se o multiplicador relacionado for 0.

Para solucionar o PPL, foi utilizada uma versão do Algoritmo de Prim para matrizes de adjacência (representação escolhida para o grafo), com complexidade $O(n^2)$.

A AGM foi representada como um vetor de tamanho $n = |V|$. Em cada posição i, estava o índice do "pai" de i. A raiz (normalmente o vértice 1) possui valor -1 para indicar raiz. Para verificar viabilidade para o DCMSTP, a árvore é percorrida contando a quantidade de arestas presentes para cada vértice (pai, se existir, e filhos, se existirem). Se os graus de todos os vértices respeitarem as restrições, a árvore é viável. Isso é feito em $O(n)$. Os subgradientes são calculados de maneira similar, porém com um vértice de cada vez.

No geral, a implementação do método do subgradiente (sem levar em conta a heurística lagrangiana) tem complexidade $O(n^2)$ por iteração. Se para terminar, T iterações são feitas, a complexidade seria $O(Tn^2)$. Como é necessário um limitante primal para cálculo do passo, usamos a heurística *First Primal*, descrita na seção 3a. Assim, a complexidade é $O(Tn^2 + n^3)$.

c. Heurística Lagrangiana

Além do método do subgradiente, foi feita uma heurística lagrangiana para transformar a solução do PPL em uma solução do problema original, gerando um limitante primal para o problema que pode ou não ser melhor que o encontrado anteriormente. Essa heurística é, a princípio, aplicada em toda iteração.

Se a solução do PPL já é viável e fornecer um limitante primal melhor que o atual, o atualiza. Senão, é feito um processo de **viabilização** da árvore.

Nele, inicialmente se calcula as violações e "folga" das restrições de grau para cada vértice. No pior caso, temos $n - 1$ vértices saturados. Feito isso, para cada vértice, se estiver saturado, são trocadas arestas de modo a diminuir o grau do vértice e não saturar outro, até que o vértice saia da saturação.

Para troca de arestas em um vértice v, são trocados apenas as arestas que ligam v a seus **filhos**. A seleção dos filhos é feito de maneira *first fit*, então a aresta para o primeiro filho que encontrar em ordem numérica. Então, de forma **gulosa no peso das arestas**, tenta trocar o pai do filho selecionado **se o possível pai não estiver saturado**. Após a tentativa de troca, verifica se o grafo resultante é uma árvore. Se for, a troca foi bem-sucedida.

A verificação de ciclo cria uma matriz de adjacências a partir do vetor da árvore, e verifica se algum vértice tem dois pais. Depois, verifica se é conexo por um DFS.

Essa heurística não cobre todos os casos possíveis, então é possível que, após todos esses processos, o resultado não seja viável (restrições de vértices ou árvore). Assim, essas verificações são feitas mais uma vez, e se o resultado for positivo a árvore viabilizada é retornada. Senão, o processo falhou e retorna resultado nulo.

Então:

- Verificação de ciclo + conectividade: $O(n^2)$
- Trocar pai de um filho, se possível: $O(n^3)$
- Trocar uma aresta: $O(n^4)$
- Tirar a saturação de um vértice: $O(n^5)$
- Viabilizar a árvore: $O(n^6)$

Assim, fica claro que a heurística lagrangiana domina a complexidade por iteração. A complexidade total então fica $O(Tn^6)$, sendo T o número de iterações do método do subgradiente.

Existem várias maneiras de melhorar o desempenho na prática ou teórico da heurística, e maneiras melhores de viabilizar a árvore (que não falham), mas não puderam ser feitas até a data de entrega.

d. Otimizações

Como a complexidade da heurística lagrangiana é muito alta, o tempo por iteração em instâncias maiores aumentou muito, afetando a qualidade do dual encontrado no tempo alocado. Para melhorar essa situação, a partir de um certo limite (no caso, 900 vértices) a heurística lagrangiana é executada apenas quando um limitante dual melhor for encontrado. Em instâncias menores, é executada em todas as iterações do método do subgradiente.

3. Heurísticas e Metaheurística

a. Heurísticas

Nota: As ordenações das arestas são utilizadas em todas as heurísticas citadas, mas esta é realizada apenas duas vezes durante toda a execução do código (no *First Primal* e na metaheurística). O *quicksort* possui pior caso $O(x^2)$ dependendo da escolha do pivot, mas como a probabilidade deste caso é muito baixa, principalmente com valores bem distribuídos (como é o caso dos pesos das arestas nas instâncias), consideramos a complexidade média do *quicksort* de $O(x \cdot \lg x)$.

First Primal: heurística que tem como finalidade encontrar um solução viável inicial que fora utilizada tanto para a execução da busca local quanto para a definição do passo na heurística lagrangiana. Utilizou-se uma heurística construtiva gulosa no peso das arestas. Para grafos completos, a *First Primal* garante uma solução viável para o *DCMST*.

Implementação: $O(n^3 + n^2 \cdot \lg n^2 + n)$

- $O(n)$ - Inicializa os n vertices do grafo como componentes desconexas.
- $O(n)$ - Como cada vértice comeca como uma componente, Inicializa as vagas das componentes como o numero maximo de conexoes em cada vertice.
- $O(n^2 \cdot \lg n^2)$ - Ordedena-se as n^2 arestas da instância em ordem crescente de peso com o *quicksort*.

- $O(n^3)$ - Para as n^2 arestas, em ordem crescente de peso, verifica se a aresta: **1 -** Liga componentes distintas (evita ciclos). **2 -** Não viola a restrição de grau dos vértices. **3 -** Não satura ambas componentes a qual se conecta (evita grafos disjuntos) **ou** e a última aresta a ser inserida.
  Caso a aresta cumpra tais condições, insere-a no grafo juntando as componentes de seus vértices e atualizando os graus e vagas.
  Nota: a junção de componentes é feita com o auxílio de uma *DFS* que percorre os vértices da nova componente e os atribui o *ID* de menor valor entre as componentes. Como este é feito com matriz de adjacência, sua execução e da ordem de $O(n^2)$ e ocorre $n-1$ vezes, pois sao inseridas apenas $n-1$ arestas ($O(n^3)$).

Random Primal: Tem o mesmo objetivo e passos do *First Primal* diferindo apenas na forma em que a aresta é selecionada. Ao invés de uma seleção gulosa das arestas a serem inseridas, é escolhido um valor inteiro aleatório no intervalo $[0,\ n^2-1]$. Caso a aresta não seja viável, itera-se pela lista até encontrar uma que seja. Como a lista utilizada é ordenada, esta heurística tem complexidade $O(n^4)$ no pior caso.

Busca local: A busca local visa modificar a soluçao gerada pela First Primal de forma que, a partir da remoção e re-inserção gulosas de arestas, seja realizada uma busca por soluções vizinhas que possam ser melhor do que a encontrada inicialmente. Supõe-se a solução viável G, removendo as três arestas mais pesadas de G temos quatro novas componentes conexas, dado que G e uma árvore. Após a remoção, tenta-se inserir as três arestas mais leves (diferentes das removidas) que juntem essas componentes.

Implementação: $O([k+1]\cdot n^2\ +\ n^2\cdot \lg n^2) \mid K \in [3, \frac{n}{3}] \Rightarrow O(n^3)$ *pior caso*

- $O(n)$ - Inicializa os n vertices do grafo como uma unica componente, ja que G e uma árvore.
- $O(n)$ - Inicializa as vagas da componente sendo essa a soma dos graus livres de todos os vértices da árvore G.
- $O(n^2\cdot \lg n^2)$ - ordedena-se as n^2 arestas da instância em ordem decrescente de peso.
- $O(K\cdot n^2)$ - Para as n^2 arestas, em ordem decrescente de peso e ate que $K \in [3, \frac{n}{3}]$ arestas sejam removidas, verifica se a aresta Faz parte de G. Caso sim, remove a aresta, separa as componentes, atualiza os graus e as vagas e continua. Caso não, apenas continua.
  Nota: a disjunção de componentes é feita com o auxílio de uma *DFS* que percorre os vértices da nova componente e os atribui uma nova *ID* de componente. Como a busca é feita com matriz de adjacência, sua execução e da ordem de $O(n^2)$ e ocorre K vezes, pois são removidas apenas K arestas ($O(K\cdot n^2)$).

- $O(K \cdot n^2)$ - Para as n^2 arestas, em ordem crescente de peso e até que $K \in [3, \frac{n}{3}]$ arestas sejam inseridas. A aresta é inserida se as seguintes condições são satisfeitas: **1 -** Liga componentes distintas (evita ciclos). **2 -** Não viola a restrição de grau dos vértices. **3 -** Não satura ambas componentes a qual se conecta (evita grafos disjuntos) **ou** e a última aresta a ser inserida. **4 -** Não está na lista de tabus. Caso contrário, continua percorrendo as arestas.
- $O(K \cdot n^2)$ - Critério de aspiração de viabilidade: Caso o grafo seja disjunto, o que pode ocorre quando uma das arestas removidas é a única viável para a permutação selecionada, percorre as n^2 arestas, em ordem crescente de peso, e insere-as seguindo as condições anteriores **excetuando-se a condição (4)**, ou seja, ignorando se é um tabu.

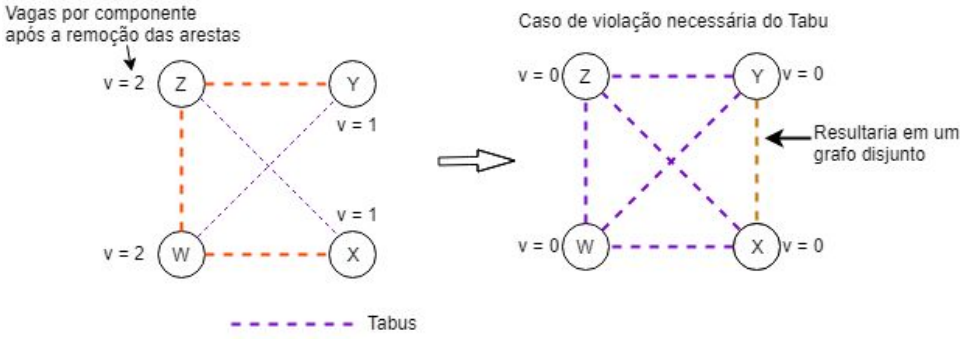

Figura 1 - Exemplo de caso de quebra de tabu necessária.

b. Metaheurística

Busca Tabu ponderada e expansiva: A fim de garantir que a heurística de busca local não ficasse presa em ótimos locais, utilizou-se busca tabu para proibir a re-inserção de arestas dentro de um intervalo de tempo. O tempo que uma aresta deve se manter fora da solução após ser removida foi definido por $M \% (c + 1)$, onde "M" e o maior peso dentre todas as arestas e "c" e o custo da aresta que foi removida, penalizado, assim, arestas mais pesadas. Além disso, definiu-se um limite de iterações em que, se a heurística não gera resultados melhores com K permutacoes dentro deste limite, aumenta-se a quantidade de arestas permutas para $K + 1$, o que acarreta em maior diversidade de resultados em troca de menor qualidade. A quantia de permutas expande até alcançar o valor $K = piso(n/3)$.

Implementação: $O(n^3)$

- $O(n^3)$ - Executa o First Primal para obter uma solução viável inicial.
- $O(n^2)$ - Cria matriz de arestas tabus e seus respectivos tempos.
- $O(K \cdot n^2)$ - Execute a heuristica de busca local.

- $O(n^2)$ - Atualizar primal e converter matriz de adjacência para formato de vetor de pais (uso de *DFS*).
- $O(n^2)$ - Atualizar lista de tabus e os respectivos tempos.

c. Otimizações

Inicialmente utilizou-se uma busca tabu com tempos constantes e $K = 3$ fixo. Tal implementação, não gerou melhorias para nenhuma instância e apresentou baixíssima diversidade de soluções. Atribuiu-se isso a três fatores. Primeiro, devido ao fato que a remoção e inserção de arestas era feita de forma gulosa tornando-se muito propícia remover/inserir sempre as mesmas arestas. Segundo, dado que o valor $K = 3$ não permitia muitas permutações distintas da original (antes da remoção), principalmente dado que os graus dos vértices são menores ou iguais a dois na maioria das instâncias de entrada. Terceiro, por fim, os tempos constantes fazem com que as mesmas arestas removidas em uma iteração fiquem disponíveis no mesmo instante de tempo, aumentando as chances de que as mesmas arestas sejam inseridas juntas.

Para melhorar a baixa variabilidade e tentar manter a qualidade das soluções, adicionou-se o tempo de remoção baseado no peso das arestas, favorecendo arestas de menor peso. Com esta mudança, a busca iterou por mais soluções distintas e foram encontradas soluções melhores para três instâncias. Todavia, a busca ainda iterava por muitas soluções repetidas.

Visando aumentar a variabilidade, variou-se K de forma que, se o primal não fosse encontrado um novo ótimo em 100 iterações, acresia K em uma unidade até que alcançasse $n/3$. Com esta mudança, sete instâncias encontraram soluções melhores e a diversidade de resultados aumentou significativamente.

Apesar da boa diversidade de soluções alcançadas, a qualidade raramente foi melhoradas. A partir da suspeita que a implementação atual não conseguia sair dos mínimos locais encontrados pelas heurísticas gulosas, tentou-se utilizar soluções iniciais construídas de forma aleatória utilizando-se a heurística *Random Primal* que fora aplicada quando o melhor primal não é melhorado durante um limite de iterações. Esta mudança não apresentou melhorias, mesmo testando com aplicações diferentes dos tempos de tabu e de valores de K.

As otimizações foram analisadas a partir de diversos valores: média dos primais, maior primal, primeiro primal, melhor primal, número de iterações, número de soluções distintas e quantidade de atualizações. Foi implementada uma lista de matrizes de adjacência para registrar as *DCMSTs* obtidas e verificar repetições. Também foi feito o uso de um vetor de *hash* para contagem de primais. Sendo estes aparatos de teste que consomem demasiada memória e processamento, não são executados no código final.

4. Resultados

Os testes foram realizados com tempo máximo igual a 300s (5 minutos), em um computador pessoal cujas especificações estão disponíveis na Tabela 1.

Tabela 1: Especificações de Hardware

Modelo da CPU	Intel(R) Core 17-3630QM (4C/8T)
Frequência do *Clock* da CPU	2.40 GHz
RAM Disponível	12 GB/1600 MHz

A saída do programa depende do método utilizado: para a relaxação lagrangiana, a saída inclui o melhor limitante dual encontrado e o melhor limitante primal encontrado pela viabilização da solução dual, e para a metaheurística, a saída inclui apenas o melhor limitante primal encontrado.

Os resultados para a metaheurística e relaxação lagrangiana podem ser encontrados nas Tabelas 2 e 3, respectivamente. O código foi compilado usando o compilador *GCC*, utilizando a *flag* de otimização “O3”, para otimização máxima por parte do compilador.

5. Comparação

Considera-se que o ótimo foi encontrado quando a diferença entre os limitantes dual e primal é menor que 1, o que é verdade visto que o limitante primal é um número inteiro. Tendo essa definição, percebe-se que a heurística lagrangiana encontrou a solução ótima em 11/40 instâncias, o que é um bom número se considerando o aumento de tamanho das instâncias.

Comparando com os resultados da referência [2], observa-se uma melhora no resultado em diversas instâncias. Isso não ocorre, porém, com as instâncias iniciadas em “tb8”. Essas instâncias, mesmo menores, são complicadas, então vê-se que os resultados da literatura são em alguns casos muito melhores que os observados pela heurística implementada.

Agora, comparando ambas estratégias, percebe-se que a metaheurística produziu resultados superiores em apenas uma das instâncias, a “*tb8ct500_1*”, não atingindo a solução ótima em instância alguma.

Os resultados obtidos para a metaheurística, em grande parte, não foram melhorados a partir da inicial, com exceção de sete instâncias que melhoraram o primal pelo menos uma vez durante a execução de teste. No entanto, para quase todas as instâncias o primal obtido pela heurística *First Primal* foram percentualmente boas em relação ao dual fornecido pela relaxação lagrangeana. Percebe-se o *tradeoff* fornecido pela heurística gulosa do *First Primal*: devido a boa qualidade da solução, implementar uma busca local que seja eficiente para escapar de ótimos locais e conseguir encontrar soluções ótimas. A escolhas dos parâmetros K, número de iterações para aumentar K e os tempos para a lista de tabus foram regulados visando explorar o mínimo local fornecido pela solução inicial e gradualmente ampliar a região de busca possibilitando soluções piores mas mais variadas.

Tabela 2: Resultados para Metaheurística

Instância	Melhor Primal	Tempo(s)
tb1ct100_1	3815	300
tb1ct100_2	3858	300
tb1ct100_3	3983	300
tb1ct200_1	5373	300
tb1ct200_2	5766	300
tb1ct200_3	5754	300
tb1ct300_1	6498	300
tb1ct300_2	6894	300
tb1ct300_3	6461	300
tb1ct400_1	7453	300
tb1ct400_2	7889	300
tb1ct400_3	7698	300
tb2ct500_1	8285	300
tb2ct500_2	8429	300
tb2ct500_3	8652	300
tb2ct600_1	9057	300
tb2ct700_1	9817	300
tb2ct800_1	10365	300
tb3ct900_1	10947	300
tb3ct1000_1	11439	300
tb3ct2000_1	15716	300
tb3ct2000_2	16460	300
tb3ct2000_3	16907	300
tb3ct2000_4	16596	300
tb3ct2000_5	16772	300
tb8ch100_0	4781	300
tb8ch100_1	4345	300
tb8ch100_2	5067	300
tb8ch200_0	6563	300
tb8ch200_1	6307	300
tb8ch200_2	6988	300
tb8ch300_0	7871	300
tb8ch300_1	8067	300
tb8ch300_2	7943	300
tb8ch400_0	9316	300
tb8ch400_1	9849	300
tb8ch400_2	8854	300
tb8ch500_0	9810	300
tb8ch500_1	10181	300
tb8ch500_2	9779	300

Tabela 3: Resultados para Relaxação Lagrangiana

Instância	Melhor Primal	Melhor Dual	Tempo(s)
tb1ct100_1	3790	3790.0000	0.0169
tb1ct100_2	3829	3829.0000	0.0252
tb1ct100_3	3916	3915.7415	0.0252
tb1ct200_1	5316	5316.2372	0.1975
tb1ct200_2	5648	5647.9304	0.6842
tb1ct200_3	5698	5698.0000	0.2890
tb1ct300_1	6480	6474.9721	0.4127
tb1ct300_2	6817	6803.9368	1.4440
tb1ct300_3	6430	6429.2417	0.3883
tb1ct400_1	7414	7413.6767	0.3837
tb1ct400_2	7794	7777.4952	2.6357
tb1ct400_3	7617	7601.4769	5.0774
tb2ct500_1	8272	8271.8402	1.0600
tb2ct500_2	8395	8394.1857	2.8093
tb2ct500_3	8507	8502.9768	5.5521
tb2ct600_1	9037	9036.5529	0.8245
tb2ct700_1	9796	9789.0039	3.8619
tb2ct800_1	10336	10334.9922	6.6541
tb3ct900_1	10923	10919.9207	3.0730
tb3ct1000_1	11426	11408.0701	7.4295
tb3ct2000_1	15704	15673.6304	20.6801
tb3ct2000_2	16324	16247.4677	96.0225
tb3ct2000_3	16755	16684.0762	83.8004
tb3ct2000_4	16485	16381.1777	81.4307
tb3ct2000_5	16627	16533.2889	66.6601
tb8ch100_0	4553	4274.7920	0.5464
tb8ch100_1	4112	3873.7411	0.3077
tb8ch100_2	4821	4418.1251	0.5423
tb8ch200_0	5991	5600.0162	2.6815
tb8ch200_1	6059	5700.9582	2.6049
tb8ch200_2	6274	5971.6013	3.7524
tb8ch300_0	7770	6861.0655	8.9270
tb8ch300_1	7870	6845.6991	10.7176
tb8ch300_2	7943	7088.2110	9.9621
tb8ch400_0	9163	7925.9252	24.2518
tb8ch400_1	9681	8112.5840	24.6475
tb8ch400_2	8854	7732.3270	18.6650
tb8ch500_0	9810	8776.4130	49.3352
tb8ch500_1	10256	8811.5751	38.4382
tb8ch500_2	9779	8735.6292	44.9351

Em algumas instâncias menores, na relaxação lagrangiana pode-se notar limitantes duais estranhos se comparados com o limitante primal ou à literatura. Mais especificamente, o limitante dual encontrado é maior que o primal na instância "*tb1ct200_1*", e é maior que a solução ótima vista em [2] na instância "*tb8ct500_1*". Diversos testes foram feitos para descobrir a origem desse erro (imprecisão numérica, erro de implementação ou erro de lógica) mas não foi possível descobri-la. Nas duas instâncias em que esse erro foi visto, simplesmente tirar o piso do valor é o suficiente

para eliminar os erros, mas sem saber exatamente a causa não pode-se concluir uma solução para o problema.

Como na relaxação lagrangiana todas as instâncias terminaram antes do tempo máximo ser atingido, já que há mais critérios de parada que são atingidos antes, não há relação entre tempo de computação e qualidade da solução nesse caso. A solução encontrada pode ser ótima, ou o melhor dual possível pode ser encontrado sem necessidade de muito tempo. Porém, o oposto pode ser dito para a metaheurística, cujo único critério de parada é o tempo máximo de execução. Nesse caso, quanto maior o tempo de execução, mais vizinhos são explorados, e pela metaheurística diferentes áreas do espaço de soluções podem ser explorados.

Esse *tradeoff* diminui a medida em que respostas melhores vão sendo obtidas, e inexiste após a solução ótima ser encontrada, pois a metaheurística continua realizando operações sem chegar em solução melhor, e o tempo excedente é inútil. Porém, como não há limitantes duais pela metaheurística, não é possível descobrir quando essa solução ótima é encontrada, então o tempo máximo de computação ideal pode não existir, e o utilizado deve ser definido empírica ou estatisticamente.

6. Conclusão

A relaxação lagrangiana implementada produziu resultados satisfatórios em pouco tempo, inclusive solucionando de modo ótimo diversas instâncias. Porém, não se pode concluir com confiança muito sobre os limitantes duais.

Supondo, a partir das instâncias em que isso é visível, que o erro é baixo e pode ser facilmente desconsiderado, tem-se que o *gap* de otimalidade é baixo para a maioria das instâncias. Melhoras pontuais ou significativas podem ser feitas na heurística para diminuir sua complexidade, o que é muito útil para instâncias maiores, ou a qualidade das soluções (eliminando a possibilidade de falhas e melhorando a substituição de arestas e a decisão de qual filho trocar).

Em relação a metaheurística, reitera-se a importância de fatores aleatórios para possibilitar a diversificação da busca de soluções assim como a qualidade de soluções obtidas por heurísticas gulosas. Em relação a implementação adotada, seria adequado uma boa aplicação da estratégia *multistart local search* e um parâmetro para a coordenação da busca que relacionasse os passos tomados com as qualidades de soluções obtidas, visando a convergência da busca em novos mínimos locais que sejam melhores que o inicial ou mesmos ótimos globais. Uma metaheurística mais simples seria mais fácil de analisar e parametrizar, sendo, possivelmente, uma boa alternativa a métodos complexos com muitos parâmetros como o que foi utilizado.

Referências:

[1] J. E. Beasley. Lagrangian relaxation. In *Modern heuristic techniques for combinatorial problems*, pages 243–303. John Wiley & Sons, Inc., 1993

[2] R. Andrade, A. Lucena, and N. Maculan. Using lagrangian dual information to generate degree constrained spanning trees. *Discrete Applied Mathematics*, 154(5):703–717, 2006.