Uma Busca Local Iterada para o Problema da Árvore Geradora Mínima sob Restrições de Conflitos

Marco Aurélio Lopes Barbosa
Departamento de Informática - Universidade Estadual de Maringá
Av. Colombo, 5790 - UEM - Bloco C56, Maringá - PR, CEP 87020-900
malbarbosa@uem.br; malbarbo@gmail.com

Alexandre Cláudio Botazzo Delbem
Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo
Avenida Trabalhador São-carlense, 400 - Centro, São Carlos - SP, CEP 13566-590
acbd@icmc.usp.br

RESUMO

O problema da árvore geradora mínima e suas variantes são bastante estudados na literatura e têm muitas aplicações práticas, principalmente em projetos de redes. O foco deste trabalho é a variante que considera restrições de conflito nas arestas. Nesta variante a árvore não pode conter determinados pares de arestas, o que torna o problema NP-Difícil. Existem na literatura diversos resultados de complexidade e algoritmos exatos para o problema, mas apenas um estudo preliminar de algoritmos heurísticos. Neste trabalho apresentamos um algoritmo de busca local iterada e uma nova estrutura de vizinhança para o problema. Em um conjunto de instâncias padrão, a busca local iterada usando a nova vizinhança encontrou todas as soluções ótimas conhecidas e melhorou o resultado para três instâncias, superando os outros algoritmos heurísticos existentes.

PALAVRAS CHAVE. Árvore geradora mínima sob restrições de conflito. Vizinhança. Busca local iterada.

Otimização Combinatória. Metaheurísticas.

ABSTRACT

The minimum spanning tree problem and its variants are well studied in the literature and have many practical applications, especially in the design of networks. The focus of this paper is the variant that considers edge conflict constraints. In this variant the tree can not contains certain pairs of edges, which makes the problem NP-Hard. There are many results of complexity and exact algorithms for the problem in the literature, but only a preliminary study of heuristic algorithms. In this paper we present an iterated local search algorithm and a new neighborhood structure for the problem. In a set of standard instances, the iterated local search using the new neighborhood found all the known optimal solutions and improved the result for three instances, outperforming the other existing heuristic algorithms.

KEYWORDS. Minimum spanning tree with conflict constraints. Neighbourhood. Iterated local search.

Combinatorial Optimization. Metaheuristics.

1. Introdução

O problema da árvore geradora mínima (AGM) é bastante conhecido na área de otimização combinatória. Diversos problemas práticos, principalmente relacionados a projeto de redes, podem ser modelados como variantes do AGM. Muitas destas variantes, como o problema da árvore geradora mínima quadrática (Assad e Xu [1992]) e o problema da árvore geradora mínima capacitada (Voß [2009]), são NP-Difíceis.

O foco deste trabalho é uma variante do AGM chamada de problema da árvore geradora mínima sob restrições de conflito (AGMRC). Dados um grafo $G = (V, E)$, uma função de custo $w : E \to \mathbb{R}^+$ e um conjunto de pares de arestas conflitantes $C \subset E \times E$, o AGMRC consiste em encontrar uma árvore geradora T de G de custo $\sum_{e \in E(T)} w(e)$ mínimo e livre de conflitos, isto é, T não contém nenhum par de arestas que pertence a C.

Um conceito que usualmente é utilizado na definição do AGMRC é o de grafo de conflitos. No grafo de conflitos $\hat{G} = (E, C)$ os vértices correspondem as arestas de G e as arestas correspondem aos conflitos em C. Desta forma, as arestas de uma árvore geradora livre de conflitos de G formam um conjunto independente em $\hat{G}$.

A maioria dos problemas de árvore geradora mínima tem uma interpretação natural quando restrições de conflito são consideradas. O AGMRC também é utilizado em algoritmos exatos e heurísticos para o problema da árvore geradora mínima quadrática com gargalo (Zhang et al. [2011]).

1.1. Trabalhos relacionados

O AGMRC foi introduzido em Darmann et al. [2009, 2011]. Os autores apresentam diversos resultados sobre a complexidade do problema. Eles mostram que o problema é NP-Difícil, mesmo quando o grafo de conflitos é formado pela união de caminhos de tamanho dois. No caso do grafo de conflitos ser formado por arestas disjuntas (caminhos de tamanho 1), o problema admite solução em tempo polinomial.

Outros resultados sobre a complexidade do AGMRC são apresentados por Zhang et al. [2011]. Os autores mostram que o problema tem solução em tempo polinomial quando o grafo de conflitos é uma coleção de cliques disjuntos. Também apresentam algoritmos exatos, testes de factibilidade e um estudo preliminar de heurísticas para o problema. Três heurísticas foram propostas: Busca Local de Primeira Melhora (*Local Search*, LS), Busca Tabu (*Tabu Search*, TS) e Tabu *Threshoulding* (TT). Essas heurísticas utilizam a mesma estrutura de vizinhança, que consiste em trocar uma aresta da árvore por uma aresta que não está na árvore.

Samer e Urrutia [2013, 2015] propõem duas novas formulações e a utilização de uma abordagem *branch and cut* para resolver o problema. Uma formulação é baseada em restrições de eliminação de sub-rotas (*Subtour Elimination Constraints*, SECs) e a outra em restrições de cortes direcionados. A formulação baseada em SECs encontrou novos certificados de factibilidade e otimalidade e limites duais mais fortes do que os obtidos por Zhang et al. [2011].

Bittencourt et al. [2016] apresentam uma modelagem que utiliza rotulação de vértices para eliminação de ciclos. Em experimentos iniciais este modelo definiu a infactibilidade de três instâncias que estavam em aberto e apresentou um bom desempenho computacional em comparação com as abordagens de Samer e Urrutia [2013, 2015].

Embora os modelos mais recentes sejam mais robustos do que os anteriores, eles ainda não encontram as soluções ótimas para algumas instâncias factíveis usadas em testes na literatura. Além disso, o *gap* entre os limitantes de algumas instâncias são grandes. Uma alternativa para tentar diminuir estes *gaps* é a utilização de algoritmos heurísticos. No melhor do nosso conhecimento, o único trabalho na literatura que utiliza uma abordagem heurística é o de Zhang et al. [2011]. No entanto, nos resultados preliminares apresentados pelos autores, estes algoritmos heurísticos encontram soluções que estão distantes dos ótimos obtidos por algoritmos exatos e não diminuem o *gap* nas instâncias para as quais a solução ótima não é conhecida.

Neste trabalho propomos um algoritmo de busca local iterada e uma estrutura de vizinhança para o AGMRC. Em experimentos iniciais usando um conjunto de testes da literatura, nossa

abordagem encontrou todos os ótimos conhecidos e diminuiu o *gap* entre os limitantes de 3 instâncias, o que sugere que a nossa proposta é promissora.

1.2. Organização do texto

O restante deste trabalho está organizado da seguinte maneira: na Seção 2 descrevemos a nossa proposta. Na Seção 3 discutimos os resultados dos experimentos computacionais. Concluímos o artigo na Seção 4 onde também apresentamos as direções futuras do nosso trabalho.

2. Busca local iterada

Nesta seção nós propomos uma busca local iterada (*Iterated Local Search*, ILS) para o AGMRC, descrevemos uma estrutura de vizinhança da literatura e propomos uma nova estrutura de vizinhança. Também descrevemos como implementar a busca local nessas duas vizinhanças.

A ideia da ILS (Lourenço et al. [2010]) é executar repetidamente uma busca local e aproveitar características das soluções obtidas nas buscas anteriores. O algoritmo mantém uma solução incumbente T e a melhor solução encontrada até o momento T_b. A cada iteração uma busca local é realizada a partir de T e, se o resultado for melhor do que T_b, então T_b é atualizada. Antes da próxima iteração, T sofre uma "perturbação" para tentar fazer a próxima busca sair do mínimo local. A perturbação deve ser balanceada, não pode ser muito forte, caso contrário a nova vizinhança de busca perde a relação com a vizinhança de T, descartando assim as características obtidas nas buscas anteriores. Por outro lado, a perturbação não pode ser muito fraca, caso contrário o mínimo local se manterá o mesmo.

2.1. Estruturas de vizinhança

Uma vizinhança comumente utilizada em problemas de árvore geradora mínima é a vizinhança baseada em troca de arestas (Zhang et al. [2011]). Podemos entender esta vizinhança de duas formas. Na primeira, uma aresta e é removida da solução (ou seja, da árvore geradora) criando duas componentes, em seguida uma aresta diferente de e que conecta as duas componentes é adicionada a solução. Na segunda forma, uma aresta e é inserida na solução criando um ciclo, em seguida uma aresta do ciclo diferente de e é removida. Observe que este esquema preserva a propriedade de árvore da solução (subgrafo conexo e sem ciclos). A seguir, damos uma definição formal dessa vizinhança.

Seja $T \subset E$ o conjunto de arestas de uma árvore geradora de um grafo $G = (V, E)$ e $\overline{T} = E \setminus T$ o conjunto de arestas de E que não estão em T. Chamamos de N_1 a vizinhança baseada na troca de uma aresta e a definimos como $N_1(T) = \{(T \setminus \{e\}) \cup \{f\} : e \in E, f \in \overline{T}$ e $(T \setminus \{e\}) \cup \{f\}$ não contém ciclos$\}$. N_1 é chamada de *2-exchange* em Zhang et al. [2011].

Nós estendemos a ideia de trocar uma aresta da árvore e propomos uma vizinhança definida pela troca de duas arestas, isto é, $N_2(T) = \{(T \setminus \{e_1, e_2\}) \cup \{f_1, f_2\} : \{e_1, e_2\} \in \binom{T}{2}, \{f_1, f_2\} \in \binom{\overline{T}}{2}$ e $(T \setminus \{e_1, e_2\}) \cup \{f_1, f_2\}$ não contém ciclos$\}$. No melhor do nosso conhecimento, essa vizinhança não foi utilizada em nenhum problema de árvore geradora mínima. Talvez um aspecto que tenha inibido o seu uso tenha sido o seu tamanho. Enquanto o tamanho de N_1 é $O(nm)$, o tamanho de N_2 é $O(n^2m^2)$, onde $n = |V|$ e $m = |E|$. Embora a vizinhança N_2 seja grande, nós implementamos uma busca local em N_2 que é eficiente na prática.

Para definir uma busca nas vizinhanças N_1 e N_2 precisamos lidar com soluções infactíveis. Para isso, usamos a estratégia proposta em Zhang et al. [2011] que utiliza uma função objetivo que penaliza os conflitos na solução. A função objetivo é definida como: $obj(T) = w(T) + \alpha c(T)$, onde $w(T)$ é a soma dos custos das arestas em T, α é uma constante de valor alto e $c(T) = |\{(e, f) : \{e, f\} \in C$ e $\{e, f\} \in T\}|$ é a quantidade de restrições violadas (do conjunto de restrições C).

No Algoritmo 1 listamos os procedimentos UMA-TROCA e DUAS-TROCAS, que descrevem em pseudocódigo, respectivamente, a busca pelas vizinhanças N_1 e N_2. Descrevemos na próxima seção como estes procedimentos podem ser implementados de forma eficiente.

Algoritmo 1 Busca nas vizinhanças N_1 e N_2

UMA-TROCA($T, \overline{T}$)

1 **para cada** aresta $e \in T$ **faça**
2 $\quad T' = T \setminus \{e\}$
3 $\quad$ **se** existe uma aresta $f \in \overline{T}$ tal que:
$\qquad\qquad T' \cup \{f\}$ forme uma árvore **e**
$\qquad\qquad obj(T' \cup \{f\}) < obj(T)$
$\quad$ **então**
4 $\qquad T = (T \setminus \{e\}) \cup \{f\}$
5 $\qquad \overline{T} = (\overline{T} \setminus \{f\}) \cup \{e\}$
6 $\qquad$ **devolve** MELHOROU
7 **devolve** NÃO-MELHOROU

DUAS-TROCAS($T, \overline{T}$)

1 **para cada** par de arestas $(e_1, e_2) \in \binom{T}{2}$ **faça**
2 $\quad T' = T \setminus \{e_1, e_2\}$
3 $\quad$ **se** existe um par de arestas $(f_1, f_2) \in \binom{\overline{T}}{2}$ tal que:
$\qquad\qquad T' \cup \{f_1, f_2\}$ forme uma árvore **e**
$\qquad\qquad obj(T' \cup \{f_1, f_2\}) < obj(T)$
$\quad$ **então**
4 $\qquad T = (T \setminus \{e_1, e_2\}) \cup \{f_1, f_2\}$
5 $\qquad \overline{T} = (\overline{T} \setminus \{f_1, f_2\}) \cup \{e_1, e_2\}$
6 $\qquad$ **devolve** MELHOROU
7 **devolve** NÃO-MELHOROU

2.2. Implementação da busca nas vizinhanças

Para cada aresta e definimos um atributo $e.custo$ que representa o custo de adicionar a aresta e na solução (a notação $a.b$ denota o atributo b do objeto a). Os conjuntos T e $\overline{T}$ são representados por arranjos. Os índices dos arranjos são usados para acessar as arestas permitindo que as trocas (linhas 4 e 5 dos procedimentos UMA-TROCA e DUAS-TROCAS) sejam realizadas em $O(1)$. A atribuição na linha 2 é conceitual e não precisa ser realizada. O existencial na linha 3 do procedimento UMA-TROCA (DUAS-TROCAS) é implementado com um laço que testa cada aresta (par de arestas).

A avaliação da função objetivo obj pode ser feita utilizando a estratégia incremental proposta em Zhang et al. [2011]. A ideia é computar apenas o custo das alterações na solução incumbente T, permitindo que as soluções vizinhas sejam avaliadas rapidamente. Para isto definimos um atributo $e.conf$ para cada aresta e que armazena a quantidade de arestas conflitantes com e que estão na solução incumbente T, isto é, $e.conf = |\{f : f \in T \text{ e } \{e, f\} \in C\}|$. Dado o valor da função objetivo $obj(T)$ e as arestas $e \in T$ e $f \in \overline{T}$, o valor da função objetivo para $T' = (T \setminus \{e\}) \cup \{f\}$ pode ser computado em $O(1)$ por $obj(T') = w(T) + f.custo - e.custo + \alpha C(T')$, onde

$$C(T') = \begin{cases} c(T) + f.conf - e.conf - 1 & \text{se } \{e, f\} \in C \\ c(T) + f.conf - e.conf & \text{caso contrário.} \end{cases}$$

Neste caso temos que $T' \in N_1(T)$. Para os elementos de $N_2(T)$ a avaliação incremental é feita de maneira similar. Quando uma aresta f é removida da árvore (resp., adicionada a árvore)

os valores $e.conf$ são decrementados (resp., incrementados) para toda aresta e tal que $\{f, e\} \in C$. Considerando que a quantidade de conflitos de uma aresta é $O(m)$, o tempo de execução desta operação é de $O(m)$.

Para podermos verificar em $O(1)$ se a troca da(s) aresta(s) gera uma árvore válida (primeira condição da linha 3 dos procedimentos UMA-TROCA e DUAS-TROCAS) executamos uma busca em profundidade (*Depth First Search*, DFS) em T antes da linha 1. Para cada aresta $e \in T$ definimos dois atributos, o carimbo de tempo de descoberta $e.desc$ e o carimbo de tempo de término $e.term$ (conforme descrito em Cormen et al. [2009]). Dizemos que um vértice v é descendente de um vértice u na árvore da busca em profundidade se $u.desc \leq v.desc$ e $v.term \leq u.term$. O tempo de execução da DFS para a árvore T é $O(n)$.

Vamos considerar a troca de uma aresta (a troca de duas arestas é semelhante). Seja $\{u, v\}$ os extremos de uma aresta e tal que u tenha sido descoberto antes que v na busca em profundidade, isto é, $u.desc < v.desc$. No subgrafo $T' = T \setminus \{e\}$ os vértices descendentes de v na árvore da busca em profundidade formam uma componente e os demais vértices formam outra componente. Dados os extremos $\{x, y\}$ de uma aresta f, sendo que $x.desc < y.desc$, podemos testar em $O(1)$ se f conecta as duas componentes de T': se y é descendente de v na árvore da busca em profundidade (y está em uma componente) e x não é descendente de v na árvore da busca em profundidade (x está em outra componente), então, a aresta f conecta as duas componentes de T'.

Considerando que os testes da linha 3 podem ser executados em $O(1)$, que o pré-processamento (DFS) tem tempo de execução $O(n)$ e que a atualização do atributo $conf$ tem tempo de execução $O(m)$, o tempo de execução do procedimento UMA-TROCA é de $n + m + nm = O(nm)$ e o tempo de execução do procedimento DUAS-TROCAS é de $n + m + n^2m^2 = O(n^2m^2)$.

Para o procedimento DUAS-TROCAS o laço que implementa o existencial da linha 3 é o mais custoso (tempo $O(m^2)$). Para melhorar a eficiência computacional do procedimento nós usamos uma estratégia para diminuir o número de pares de arestas que são verificados. Podemos observar que o subgrafo $T' = T \setminus \{e_1, e_2\}$ (linha 2) tem três componentes. Vamos denominar estas componentes de A, B, C. Antes da linha 3 separamos as arestas de $\overline{T}$ em três grupos dependendo dos seus extremos: G_{AB}, G_{AC} e G_{BC}. Por exemplo, se os extremos de uma aresta e estão nas componentes A e B, então a aresta e é colocada no grupo G_{AB}. Arestas com os extremos na mesma componente não são colocadas em nenhum grupo. Desta forma não precisamos testar todos os pares em $\binom{\overline{T}}{2}$, basta testar os pares que conectam as três componentes: $G_{AB} \times G_{AC} \cup G_{AB} \times G_{BC} \cup G_{AC} \times G_{BC}$. Embora esta estratégia não diminua a complexidade assintótica do procedimento DUAS-TROCAS, ela se mostrou bastante eficiente na prática.

Uma descrição em pseudocódigo da ILS que utiliza o procedimento UMA-TROCA ou DUAS-TROCAS para resolver o AGMRC é apresentado no Algoritmo 2. Na próxima seção apresentamos os resultados computacionais da nossa proposta.

3. Resultados computacionais

Apresentamos nesta seção os resultados de uma avaliação computacional da busca local iterada (Algoritmo 2) utilizando as vizinhanças N_1 e N_2. Utilizamos o conjunto de instâncias proposto por Zhang et al. [2011] para fazer a avaliação. As instâncias são divididas em tipo 1 e tipo 2. As instâncias do tipo 2 foram geradas de maneira a garantir a factibilidade e por isso são consideradas fáceis. Todas as soluções ótimas para as instâncias do tipo 2 são conhecidas. As instâncias do tipo 1 incluem casos mais difíceis. São 85 instâncias do tipo 1, das quais 35 foram identificadas como infactíveis, 11 identificadas como factíveis e, para o restante, a factibilidade não foi determinada. Das 11 instâncias factíveis do tipo 1, há 9 que tem soluções ótimas conhecidas. Cada instância é definida por um grafo $G = (V, E)$ e por um conjunto de conflitos C, e é identificada pelo rótulo $|V|$-$|E|$-$|C|$.

Implementamos os algoritmos na linguagem de programação Rust [1] versão 1.16 e utiliza-

[1] https://www.rust-lang.org/

Algoritmo 2 Busca local iterada

ILS(E, T, P, $Iters$)

```
1  T_b = T
2  T̄ = E \ T
3  repita Iters vezes
       // Busca local
4      execute TROCA(T, T̄) enquanto o valor devolvido for MELHOROU
5      se obj(T) < obj(T_b) então
6          T_b = T
       // Perturbação
7      selecione aleatoriamente um subconjunto A ⊂ T de tamanho P
       e um subconjunto B ⊂ T̄ tal que (T \ A) ∪ B seja uma árvore
8      T = (T \ A) ∪ B
9      T̄ = (T̄ \ B) ∪ A
10 devolve T_b
```

mos o comando `cargo build --release` para fazer a compilação. Os experimentos foram executados em um notebook com processador Intel Core i5-3320M 2.6Mhz, 8GB de memória RAM e sistema operacional Debian GNU/Linux 8.7 [2]. Cada algoritmo foi executado 30 vezes. Nas tabelas com os resultados apresentamos a mediana do tempo de execução em segundos e a mediana do valor da função objetivo. Também apresentamos diagramas de caixa que permitem visualizar a distribuição do valor da função objetivo das soluções encontradas.

Cada execução inicia com uma solução gerada pela execução do algoritmo de Kruskal (Cormen et al. [2009]), para o qual utilizamos pesos aleatórios nas arestas para que cada solução inicial seja diferente. Utilizamos o número de arestas da instância para definir o número de iterações da busca local iterada, isto é $Iters = |E|$. A perturbação foi realizada substituindo 3 arestas da solução incumbente.

Denotamos a busca local iterada utilizando as vizinhanças N_1 e N_2 por B_1 e B_2, respectivamente.

3.1. Soluções factíveis

Considerando que encontrar soluções factíveis (árvores geradoras sem arestas conflitantes) para o AGMRC é NP-Difícil, é interessante avaliar a capacidade dos algoritmos em encontrar soluções factíveis rapidamente.

Nos experimentos desta seção os algoritmos foram interrompidos assim que uma solução factível foi encontrada. Quando uma solução factível não era encontrada o algoritmo era executado até atingir o número máximo de iterações.

Os resultados para as instâncias factíveis do tipo 1 são apresentados na Tabela 1 e na Figura 1. Na Tabela 1 podemos notar que tanto B_1 quanto B_2 encontram soluções factíveis rapidamente. No geral, B_1 é mais rápido e mais eficiente do que B_2. Apenas para o caso 200-600-1797 o algoritmo B_1 não encontrou soluções factíveis em todas as execuções (falhou em 7 execuções). Isto fez com que o tempo de execução de B_1 fosse maior do que o de B_2, pois ele executou até o limite das iterações nas 7 execuções que falharam. Apenas para o caso 50-200-199 a solução mediana encontrada por B_1 foi pior do que a encontrada por B_2. Nós estamos investigando porquê B_1 consegue encontrar melhores soluções e em menor tempo do que B_2.

[2] https://www.debian.org/

Tabela 1: Desempenho para encontrar soluções factíveis (tipo 1)

			B_1		B_2	
$\|V\|$	$\|E\|$	$\|C\|$	Obj	T (s)	Obj	T (s)
50	200	199	1200	0,000	**1192**	0,000
50	200	398	**1215**	0,000	1230	0,000
50	200	597	**1379**	0,000	1409	0,000
50	200	995	**1676**	0,000	1805	0,010
100	300	448	**4842**	0,010	4934	0,030
100	300	897	**6095**	0,050	6362	0,060
100	500	1247	**5848**	0,020	6159	0,045
100	500	2495	**8083**	0,020	8758	0,040
100	500	3741	**9221**	0,040	10354	0,060
200	600	1797	**14674**	2,255	15449	1,310
200	800	3196	**27478**	0,150	28800	0,245

Figura 1: Diagrama de caixa dos *gaps* das soluções factíveis (tipo 1)

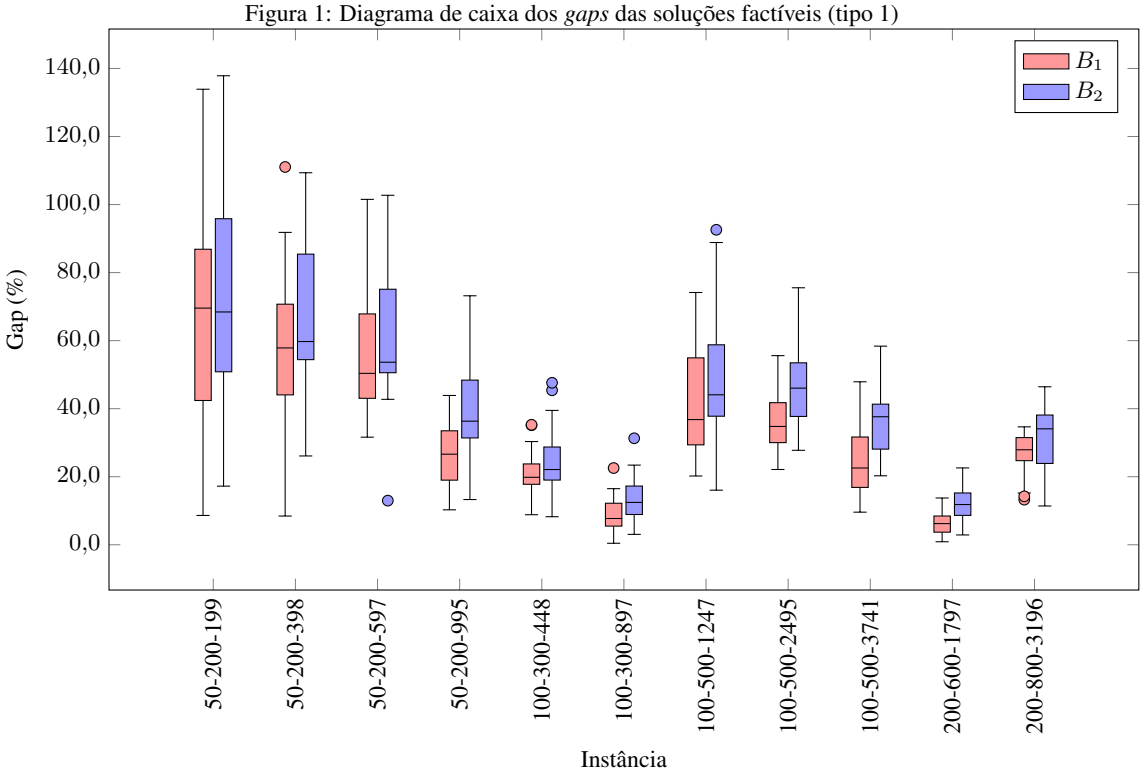

Tabela 2: Desempenho para encontrar soluções factíveis (tipo 2)

			B_1		B_2	
$\|V\|$	$\|E\|$	$\|C\|$	Obj	T (s)	Obj	T (s)
50	200	3903	**1722**	0,010	1844	0,020
50	200	4877	**2150**	0,010	2205	0,020
50	200	5864	**2378**	0,010	2434	0,030
100	300	8609	**7507**	0,030	7534	0,090
100	300	10686	**8030**	0,060	8040	0,160
100	300	12761	**8169**	0,040	8182	0,170
100	500	24740	**12731**	0,100	12890	0,340
100	500	30886	**11392**	0,115	11542	0,470
100	500	36827	**11564**	0,090	11686	0,565
200	400	13660	**17758**	0,100	17897	0,340
200	400	17088	**18713**	0,140	18784	0,375
200	400	20469	**19189**	0,130	19274	0,530
200	600	34504	**20827**	0,315	20895	1,535
200	600	42930	**18128**	0,340	18162	1,685
200	600	50984	**20864**	0,405	20954	2,615
200	800	62625	**39911**	0,520	40049	3,340
200	800	78387	**38170**	0,595	38352	4,505
200	800	93978	**38819**	0,620	39100	4,485
300	600	31001	**43721**	0,380	**43721**	9,990
300	600	38216	**44280**	0,455	44394	1,325
300	600	45310	**43206**	0,465	43347	2,320
300	800	59600	**43125**	0,785	43247	4,420
300	800	74500	**42377**	0,930	42436	5,230
300	800	89300	**44164**	1,085	44270	5,225
300	1000	96590	**71675**	1,210	71734	7,995
300	1000	120500	**76345**	1,415	**76345**	14,125
300	1000	144090	**78880**	1,610	**78880**	20,535

A Figura 1 apresenta um diagrama de caixa que mostra a distribuição do valor da função objetivo das soluções obtidas nas 30 execuções dos algoritmos B_1 e B_2. No eixo y temos o *gap* em relação a solução ótima para as instâncias de 50 e 100 vértices e o *gap* em relação a melhor solução conhecida (veja a Tabela 3) para as demais instâncias. O *gap* de uma solução s em relação a uma solução b é dado por $100 \cdot (obj(b) - obj(s))/obj(b)$. Podemos observar que o *gap* das soluções para grafos com o mesmo número de vértices e arestas diminui conforme o número de restrições aumenta. Isto porque o aumento no número de restrições diminui o número de soluções factíveis, diminuindo também o valor médio da função objetivo (supomos uma diminuição uniforme na distribuição dos valores da função objetivo).

Os resultados para as instâncias do tipo 2 são apresentados na Tabela 2 (os resultados para as instâncias do tipo 2 são similares aos resultados para as instâncias do tipo 1, por este motivo não apresentamos o diagrama de caixa). Note que o tempo de execução de B_2 aumenta mais do que o de B_1 conforme o tamanho da instância aumenta. Isto é esperado porque a vizinhança N_2 é maior do N_1 (conforme discutido na Seção 2). Nós estamos investigando uma implementação mais eficiente para N_2.

3.2. Qualidade das soluções

Nesta seção avaliamos a capacidade dos algoritmos B_1 e B_2 de encontrarem boas soluções. Os resultados são apresentados na Tabela 3 e na Figura 2. Na Tabela 3 além da mediana do valor da função objetivo, também apresentamos o melhor valor encontrado nas 30 execuções para B_1 e B_2 na coluna "Melhor". A coluna "Modelo / Obj" apresenta o valor da solução ótima obtida usando um método exato. Para as três últimas instâncias a solução ótima não é conhecida, para esses casos, mostramos os melhores limitantes conhecidos.

Tabela 3: Qualidade das soluções (instâncias tipo 1)

			Modelo	B_1			B_2		
$\|V\|$	$\|E\|$	$\|C\|$	Obj	Obj	Melhor	T (s)	Obj	Melhor	T (s)
50	200	199	**708**	**708**	**708**	0,090	**708**	**708**	0,470
50	200	398	**770**	**770**	**770**	0,100	**770**	**770**	0,460
50	200	597	**917**	946	**917**	0,110	918	**917**	0,450
50	200	995	**1324**	1364	**1324**	0,120	**1324**	**1324**	0,460
100	300	448	**4041**	4118	4058	0,380	**4041**	**4041**	3,230
100	300	897	**5658**	5704	5662	0,410	5663	**5658**	2,425
100	500	1247	**4275**	4364	4284	1,205	**4275**	**4275**	8,590
100	500	2495	**5997**	6347	6125	1,410	6024	**5997**	8,200
100	500	3741	[6538 - 9207] [1]	7675	7582	1,510	7538	**7523**	7,225
200	600	1797	[13264 - 14161] [1]	14418	13959	3,315	13915	**13815**	29,465
300	800	3196	[20744 - 21852] [2]	22453	22193	6,950	21536	**21480**	62,480

[1] Bittencourt et al. [2016] - Programa fornecido por Samer e Urrutia [2] Samer e Urrutia [2015]

Figura 2: Diagrama de caixa dos *gaps* (tipo 1)

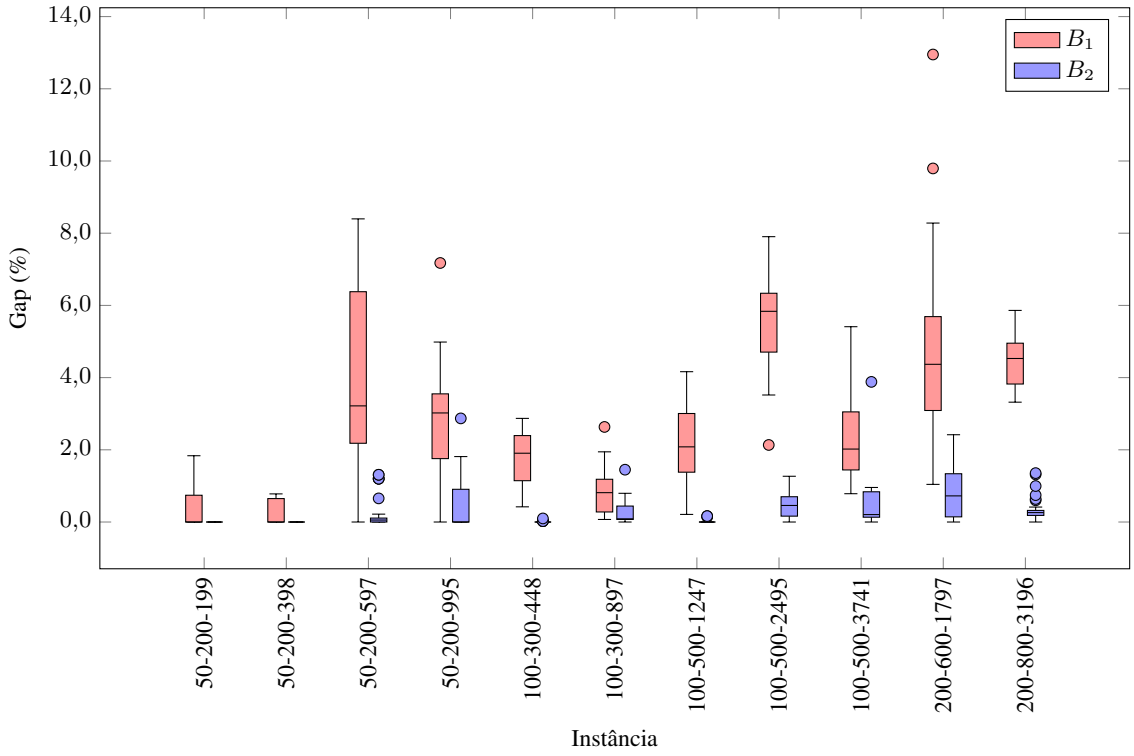

Tabela 4: Comparação das heurísticas (instâncias tipo 1)

			LS		TT		TS		B_1		B_2	
$\|V\|$	$\|E\|$	$\|C\|$	Obj	T (s)	Obj	T (s)	Obj	T (s)	Obj	T (s)	Obj	T (s)
50	200	199	**708**	0,031	735	0,434	711	0,513	**708**	0,090	**708**	0,470
50	200	398	797	0,053	789	0,519	785	0,497	**770**	0,100	**770**	0,460
50	200	597	-	0,088	1044	0,522	1086	0,428	946	0,110	**917**	0,450
50	200	995	1424	0,125	1721	0,397	1629	0,506	1364	0,120	**1324**	0,460
100	300	448	4102	0,392	4316	2,831	4207	2,772	4118	0,380	**4041**	3,230
100	300	897	-	0,769	-	2,381	-	2,625	5704	0,410	**5663**	2,425
100	500	1247	4293	0,941	4913	5,797	4539	7,756	4364	1,205	**4275**	8,590
100	500	2495	6603	1,275	7959	6,206	6812	7,484	6347	1,410	**6024**	8,200
100	500	3741	-	1,728	10066	5,681	8787	6,544	7675	1,510	**7538**	7,225
200	600	1797	-	6,431	-	37,282	-	31,744	14418	3,315	**13915**	29,465
300	800	3196	-	10,463	-	49,369	-	46,082	22453	6,950	**21536**	62,480

Assim como nos resultados da seção anterior, o tempo de execução de B_2 é maior do que o de B_1. Mas para este experimento, B_2 superou B_1 na qualidade das soluções. Considerando a mediana das 30 execuções, o algoritmo B_1 encontrou soluções ótimas para duas instâncias e B_2 encontrou soluções ótimas para 5 instâncias. Considerando o melhor valor entre as 30 execuções, o algoritmo B_1 encontrou soluções ótimas para 4 instâncias e B_2 encontrou soluções ótimas para todas as instâncias para as quais os ótimos são conhecidos. Para as três instâncias em que os ótimos não são conhecidos, o algoritmo B_2 conseguiu superar as melhores soluções conhecidas tanto considerando a mediana quanto o melhor valor, diminuindo desta forma o *gap* para o limitante inferior. Para a instância 100-500-3741 o *gap* diminui de 28,99% para 13,09%, para a instância 200-600-1797 o *gap* diminui de 6,33% para 3,99% e para a instância 200-800-3196 o *gap* diminui de 5,07% para 3,43%.

Na Figura 2 podemos observar que B_2 é consistente em encontrar boas soluções. Para as instâncias 50-200-199 e 50-200-398 B_2 encontrou a solução ótima em todas as execuções. Exceto para as instâncias 50-200-995, 100-500-3741 e 200-800-3196, todas as soluções encontradas tem um *gap* menor que 2%.

Os resultados para as instâncias do tipo 2 são discutidos na próxima seção, onde comparamos a nossa abordagem com heurísticas existentes na literatura.

3.3. Comparação com heurísticas existentes

Nesta seção comparamos B_1 e B_2 com os algoritmos heurísticos LS, TT e TS, propostos por Zhang et al. [2011] (descritos na Seção 1). É importante destacar que os autores descrevem os resultados das heurísticas como preliminares. Os experimentos de Zhang et al. [2011] foram executados em um computador Dell com processador Intel Pentium 4 3.4 Ghz e uma estação de trabalho Dell com processador Intel Xeon 2.0 Ghz.

Os resultados para as instâncias do tipo 1 são mostrados na Tabela 4. Em Zhang et al. [2011] não é informado quantas vezes cada algoritmo foi executado e se os tempos de execução e valores da função objetivo são os melhores encontrados ou a mediana (ou média) de várias execuções. Para B_1 e B_2 os valores apresentados são a mediana do tempo de execução e do valor da função objetivo. Note que os valores da coluna "Obj" e "T (s)" para B_1 e B_2 da Tabela 4 são os mesmos da Tabela 3, apenas repetidos a fim de facilitar a comparação entre as heurísticas. O valor "-" na coluna "Obj" de LS, TT e TS significa que o algoritmo não encontrou solução factível.

Podemos observar que as abordagens LS, TT e TS não encontraram solução factível para todas as instâncias. Para os casos 100-300-448 e 100-500-1247 o algoritmo LS encontrou melhores soluções do que B_1, para o caso 50-200-199, os algoritmos LS e B_1 encontraram a mesma solução, nos demais casos B_1 supera a heurística LS. O algoritmo B_1 supera TT e TS em todos os casos e

Tabela 5: Comparação das heurísticas (instâncias tipo 2)

				Tempo (s)				
$\|V\|$	$\|E\|$	$\|C\|$	Ótimo	LS	TT	TS	B_1	B_2
50	200	3903	1636	0,266	0,444	0,578	0,200	0,510
50	200	4877	2043	0,331	0,491	0,847	0,230	0,510
50	200	5864	2338	0,500	0,419	0,684	0,200	0,490
100	300	8609	7434	0,953	3,125	3,516	0,660	2,390
100	300	10686	7968	1,066	2,378	3,272	0,620	2,400
100	300	12761	8166	1,081	3,328	3,453	0,770	2,510
100	500	24740	12652	6,009	6,647	8,450	2,550	7,960
100	500	30886	11232	7,491	6,513	7,344	2,965	9,310
100	500	36827	11481	9,938	4,960	6,641	2,570	8,155
200	400	13660	17728	0,863	12,441	13,260	2,155	10,840
200	400	17088	18617	0,966	11,013	16,922	2,310	10,935
200	400	20469	19140	2,169	9,884	11,669	1,970	11,550
200	600	34504	20716	13,063	25,632	36,132	7,040	33,050
200	600	42930	18025	12,163	22,900	42,117	7,210	31,290
200	600	50984	20864	14,606	26,541	53,729	6,795	33,095
200	800	62625	39895	55,135	43,810	51,391	12,970	59,930
200	800	78387	37671	59,366	47,335	46,657	18,045	67,170
200	800	93978	38798	56,188	39,410	46,176	11,495	65,745
300	600	31001	43721	8,002	40,106	49,069	5,225	49,945
300	600	38216	44267	4,253	49,394	67,401	9,125	50,960
300	600	45310	43071	2,553	31,135	43,360	8,110	54,320
300	800	59600	43125	33,144	88,516	93,951	13,620	105,210
300	800	74500	42292	28,803	59,757	83,932	17,360	110,600
300	800	89300	44114	9,200	81,076	107,364	20,595	113,275
300	1000	96590	71562	140,117	106,301	131,958	28,630	177,240
300	1000	120500	76345	145,935	114,238	125,920	21,105	191,390
300	1000	144090	78880	125,441	123,320	142,421	26,210	205,030

B_2 supera LS, TT e TS em todos os casos.

Para fazermos uma comparação dos tempos de execução, precisamos ajustar os tempos, pois os experimentos foram executados em computadores diferentes. Utilizando as páginas CpuBenchmark [3] e UserBenchmark [4] estabelecemos um limite de 5 vezes para o desempenho (superior) da máquina que executamos nossos testes em relação as máquinas usadas por Zhang et al. [2011]. Os tempos de execução ajustados (tempo de execução reportado em Zhang et al. [2011] dividido por 5) para LS, TT e TS estão nas respectivas colunas "T (s)". Considerando o tempo ajustado, B_1 e LS apresentam os melhores tempos e são competitivos entre si. B_2 apresenta os tempos mais elevados mas é competitivo com TT e TS.

Os resultados para as instâncias dos tipo 2 estão na Tabela 5. Na coluna "Ótimo" estão os valores das soluções ótimas para cada instância. Todas as heurísticas encontraram soluções ótimas para todas as instâncias (B_1 e B_2 encontraram soluções ótimas em todas as execuções). Os resultados são semelhantes aos obtidos para as instâncias do tipo 1, com destaque para B_1 com tempo de execução total 68% melhor do que LS (a segunda heurística mais rápida).

4. Conclusões e trabalho atual

Neste trabalho propusemos um algoritmo de busca local iterada e uma vizinhança para o problema da árvore geradora mínima sob restrições de conflito. Também apresentamos a vizinhança N_1 descrita em Zhang et al. [2011]. A vizinhança N_1 é baseada na troca de uma aresta. A

[3] `https://www.cpubenchmark.net/compare.php?cmp[]=1315&cmp[]=1077&cmp[]=817` (comparação dos valores *Single Thread Rating*)

[4] `http://cpu.userbenchmark.com/Compare/Intel-Pentium-D-340GHz-vs-Intel-Core-i5-3320M/m5820vsm402` (comparação dos valores *Average User Bench*)

vizinhança N_2 proposta neste trabalho, se difere de N_1 por ser baseada na troca de duas arestas. Descrevemos implementações eficientes (na prática) para busca local nessas duas vizinhanças.

Apresentamos resultados de experimentos computacionais para um conjunto de instâncias da literatura. O algoritmo de busca local iterada é capaz de identificar soluções factíveis rapidamente tanto utilizando a vizinhança N_1 quanto a N_2. O algoritmo utilizando a vizinhança N_2 encontrou soluções ótimas para todas as instâncias para as quais o ótimo é conhecido e melhorou o resultado em três instâncias para as quais os ótimos não são conhecidos. Estes resultados superam os resultados obtidos por outras heurísticas da literatura.

Estes são resultados preliminares do nosso trabalho. Atualmente estamos explorando implementações mais eficientes para a busca na vizinhança N_2, o que vai permitir executar o algoritmo para instâncias maiores de problemas relacionados, como por exemplo, o problema da árvore geradora mínima quadrática (Assad e Xu [1992]).

5. Agradecimentos

O primeiro autor gostaria de agradecer à Fundação Araucária, à Secretaria de Estado da Ciência, Tecnologia e Ensino Superior (SETI-PR), ao Governo do Estado do Paraná e a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro para o desenvolvimento deste trabalho.

Referências

Assad, A. e Xu, W. (1992). The quadratic minimum spanning tree problem. *Naval Research Logistics (NRL)*, 39(3):399–417.

Bittencourt, Y. B., Campêlo, M., e Dias, F. C. S. (2016). Formulação MTZ para o problema da Árvore geradora mínima sob restrições de conflito. In *Anais do XLVIII SBPO*, p. 2441–2448.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., e Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition. ISBN 978-0-262-03384-8.

Darmann, A., Pferschy, U., e Schauer, J. (2009). Determining a minimum spanning tree with disjunctive constraints. In *Algorithmic Decision Theory*, p. 414–423. Springer, Berlin, Heidelberg. URL https://link.springer.com/chapter/10.1007/978-3-642-04428-1_36.

Darmann, A., Pferschy, U., Schauer, J., e Woeginger, G. J. (2011). Paths, trees and matchings under disjunctive constraints. *Discrete Applied Mathematics*, 159(16):1726–1735. ISSN 0166-218X. URL http://www.sciencedirect.com/science/article/pii/S0166218X1000435X.

Lourenço, H. R., Martin, O. C., e Stützle, T. (2010). Iterated local search: Framework and applications. In *Handbook of metaheuristics*, p. 363–397. Springer.

Samer, P. e Urrutia, S. (2013). Um algoritmo de branch and cut para árvores geradoras mínimas sob restrições de conflito. In *Anais do XLV SBPO*, p. 2521–2532.

Samer, P. e Urrutia, S. (2015). A branch and cut algorithm for minimum spanning trees under conflict constraints. *Optimization Letters*, 9(1):41–55. ISSN 1862-4472, 1862-4480. URL https://link.springer.com/article/10.1007/s11590-014-0750-x.

Voß, S. (2009). Capacitated minimum spanning trees. In Floudas, C. A. e Pardalos, P. M., editors, *Encyclopedia of Optimization*, p. 347–357. Springer US, Boston, MA. ISBN 978-0-387-74759-0. URL http://dx.doi.org/10.1007/978-0-387-74759-0_62.

Zhang, R., Kabadi, S. N., e Punnen, A. P. (2011). The minimum spanning tree problem with conflict constraints and its variations. *Discrete Optimization*, 8(2):191–205. ISSN 15725286. URL http://linkinghub.elsevier.com/retrieve/pii/S1572528610000551.