57º SBPO
SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL
Gramado • RS
DE 5 A 9 DE OUTUBRO DE 2025

Heurística GRASP para o Problema da Floresta de Steiner

Murilo Brugger Stockinger[1], **Isabel Rosseti**[1], **Simone L. Martins**[1], **Luidi Simonetti**[2], **Alexandre Plastino**[1]
[1]Universidade Federal Fluminense, Niterói, RJ, Brasil
[2]Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, RJ, Brasil
murilobs@gmail.com, rosseti@ic.uff.br, simone@ic.uff.br, luidi@cos.ufrj.br, plastino@ic.uff.br

RESUMO

Neste trabalho, foi proposta uma heurística construtiva para resolver o Problema da Floresta de Steiner (SFP). Experimentos computacionais em instâncias conhecidas da literatura indicaram que os resultados produzidos pela nossa heurística construtiva superaram àqueles obtidos pelo algoritmo primal-dual estado-da-arte proposto por Agrawal et al. [1995] e implementado por Ghalami e Grosu [2022]. Adicionalmente, propusemos uma nova heurística chamada GRASP-SFP, que aplica a metaheurística GRASP para resolver o SFP. Experimentos computacionais adicionais mostraram que o GRASP-SFP, mais especificamente a sua fase de busca local, melhorou significativamente os resultados obtidos pelo nosso algoritmo construtivo proposto.

PALAVRAS CHAVE. Problema da Floresta de Steiner. GRASP. Metaheurísticas

ABSTRACT

In this work, we propose a constructive heuristic to solve the Steiner Forest Problem (SFP). Computational experiments on benchmark instances from the literature showed that the results produced by our constructive heuristic outperformed those obtained by the state-of-the-art primal-dual algorithm proposed by Agrawal et al. [1995] and implemented by Ghalami e Grosu [2022]. Additionally, we introduce a new heuristic called GRASP-SFP, which applies the GRASP metaheuristic to solve the SFP. Further computational experiments showed that GRASP-SFP, specifically its local search phase, significantly improved the results obtained by our proposed constructive algorithm.

KEYWORDS. Steiner Forest Problem. GRASP. Metaheuristics.

1. Introdução

O Problema da Floresta de Steiner (SFP) é uma generalização do Problema da Árvore de Steiner (STP), definido em Pajor et al. [2018]. Segundo Gassner [2010], seja $G = (V, E)$ um grafo não direcionado e conexo, onde V é o conjunto de vértices, E representa o conjunto de arestas, $|V| = n$ e $|E| = m$. Dada uma função de pesos não negativos $w : E \rightarrow \mathbb{R}^+$ associada às suas arestas, e o conjunto $\tau = \{T_i \subseteq V \mid i = 1, \ldots, p\}$, composto pelos conjuntos terminais T_i, o SFP consiste em encontrar um subconjunto $F \subseteq E$ de arestas de custo mínimo tal que todos os vértices de cada T_i (para $i = 1, \ldots, p$) pertençam à mesma componente conexa no grafo induzido por F. Como o SFP é uma generalização do STP, ele é um problema NP-difícil. Moldenhauer [2013] afirmou que alguns problemas do mundo real podem ser modelados como um SFP, abrangendo desde

DOI: 10.59254/sbpo-2025-212506
http://live-proceedings.pantheonsite.io/p/212506

a manutenção de redes elétricas (Guha et al. [1999]) até sustentabilidade computacional (Dilkina e Gomes [2010]).

Na literatura, identificamos estratégias para resolver o SFP, utilizando as seguintes abordagens: algoritmos de aproximação, modelos exatos e heurísticas. A seguir, resumimos investigações a respeito de soluções para o SFP, categorizando-as de acordo com a abordagem empregada.

- **Algoritmos de aproximação para o SFP:** Agrawal et al. [1995] apresentaram o primeiro algoritmo de aproximação com tempos de execução viáveis para o SFP. Posteriormente, Goemans e Williamson [1995] aprimoraram esse algoritmo de aproximação para o SFP, com tempo de execução em $O(m \log m)$ e, no pior caso, alcançando um *gap* de integralidade de 2. As investigações sobre o SFP se expandiram para múltiplas variações do SFP. Essas variações consistem em subproblemas distintos, para os quais configurações ainda mais restritivas foram propostas (Angelopoulos [2006]; Li et al. [2009]; Moss e Rabani [2001]; Remy e Steger [2005]). Uma descoberta importante atribuída a Feige [1998] provou que, utilizando uma redução de preservação de aproximação simples do problema de cobertura de conjuntos (do inglês, *Set Cover Problem*), as aproximações do SFP alcançam um limite inferior de $(1 - o(1)) \ln(k)$, onde k é o número de conjuntos terminais.

  Uma análise extensa realizada por Ghalami e Grosu [2022] considerou vários algoritmos de aproximação. Essa investigação também disponibilizou as implementações de todas as estratégias eficazes criadas para resolver o SFP. Ademais, os autores mostraram que o algoritmo primal-dual proposto por [Agrawal et al., 1995] é o mais rápido que os demais na maioria dos casos, obtendo custos de solução próximos das soluções ótimas fornecidas pela formulação de programação inteira.

- **Modelos exatos para o SFP:** Schmidt et al. [2021] responderam a um antigo problema em aberto de projeto de redes com requisitos de conectividade, também abordado por Magnanti e Raghavan [2005]. Em seu trabalho, Schmidt *et al.* propuseram uma formulação de programação linear inteira (ILP), baseada em cortes para o SFP, mais forte do que os modelos clássicos direcionados e não direcionados. Eles também destacaram a falta de estudos sobre formulações ILP para o SFP, uma vez que a maioria das investigações conhecidas da literatura se concentra em estratégias de aproximação.

  Mahjoub et al. [2019] propuseram uma nova forma genérica de construir formulações estendidas para uma ampla classe de problemas de projeto de redes com requisitos de conectividade. A abordagem baseia-se no mapeamento de grafos em grafos em camadas, de acordo com uma função de distância, sendo aplicável a diversos problemas de projeto de redes. Experimentos em instâncias do SFP mostraram uma redução significativa no *gap* em comparação com resultados de formulações de última geração.

- **Heurísticas para o SFP:** Gupta e Kumar [2015] formularam uma heurística construtiva gulosa que gera soluções viáveis para o SFP, introduzindo um interessante esquema de construção baseado em compartilhamento de custos. Durante a fase de construção, cada caminho que conecta um par de terminais tem o custo de suas arestas ajustado para zero. Essa estratégia permite que o processamento dos pares subsequentes aproveite o custo já computado para arestas previamente utilizadas. Posteriormente, Groß et al. [2018] introduziram um procedimento de busca local acoplado à heurística construtiva previamente mencionada. Essa busca local concentra seus movimentos nas arestas primordiais, removendo custos desnecessários das soluções exploradas.

DOI: 10.59254/sbpo-2025-212506
http://live-proceedings.pantheonsite.io/p/212506

Na literatura sobre o SFP, poucos estudos oferecem estratégias heurísticas eficazes para resolver instâncias desse problema. A heurística proposta por Groß et al. [2018] incorpora estruturas de busca local e métodos construtivos. Entretanto, ela permanece uma abordagem puramente teórica, sem experimentos computacionais ou implementações. Essa falta de estratégias motiva o desenvolvimento de propostas capazes de superar as abordagens atuais, como a de Ghalami e Grosu [2022]. Neste trabalho, propomos uma heurística construtiva capaz de ser comparada a estratégia primal-dual de Ghalami e Grosu [2022]. Esta nova abordagem construtiva proposta também serve como componente de nossa proposta subsequente: uma heurística baseada na metaheurística GRASP (do inglês, *Greedy Randomized Adaptive Search Procedure*) de Feo e Resende [1995], para resolver instâncias maiores de forma eficiente, com tempos de computação reduzidos e com valores de custos de soluções razoáveis. Nossa heurística proposta, GRASP-SFP, introduz uma abordagem eficaz para o SFP, representando uma contribuição importante para a literatura. Em nossos experimentos computacionais, inicialmente comparamos a heurística construtiva proposta com a estratégia primal-dual introduzida em Agrawal et al. [1995] e posteriormente implementada em Ghalami e Grosu [2022]. Esta comparação inicial foi necessária para estabelecer uma base justa, visto que a heurística GRASP-SFP executa múltiplas iterações, gerando soluções iniciais distintas e refinando-as por meio de busca local, diferentemente do algoritmo primal-dual, que produz uma única solução aproximada por instância. Ademais, experimentos computacionais fornecendo o mesmo tempo de execução médio para nossas duas propostas mostraram que a fase de busca local melhorou significativamente os resultados obtidos pelo nosso algoritmo construtivo proposto.

O restante do trabalho está organizado como segue. A Seção 2 descreve o algoritmo construtivo proposto e sua aplicação na heurística GRASP-SFP. Os experimentos computacionais são apresentados na Seção 3. Finalmente, as considerações finais são apresentadas na Seção 4.

2. Abordagem GRASP Proposta

Métodos exatos e aproximativos estão entre as principais alternativas para resolver o SFP. No entanto, eles se tornam impraticáveis em instâncias maiores. Uma boa alternativa nesse cenário é a utilização de heurísticas. Conforme afirmado por Gendreau e Potvin [2010], heurísticas baseadas em metaheurísticas são métodos de solução que combinam procedimentos de busca local com estratégias mais amplas para escapar de ótimos locais, realizando uma busca robusta no espaço de soluções. Nas últimas décadas, essas técnicas têm proporcionado soluções quase ótimas em tempos computacionais razoáveis para muitos problemas complexos de otimização combinatória.

Neste trabalho, propomos uma heurística GRASP (do inglês, *Greedy Randomized Adaptive Search Procedure*) para resolver o SFP. A principal motivação é que o uso de múltiplas soluções iniciais, melhoradas por um procedimento de busca local, possibilita encontrar boas soluções em um tempo de execução aceitável.

GRASP, proposto por Feo e Resende [1995], é uma metaheurística simples e iterativa que tem sido aplicada com sucesso a diversas categorias de problemas de otimização. Cada iteração do GRASP é dividida em duas fases: inicialmente, uma solução viável é gerada na fase de construção. Posteriormente, na segunda fase, a vizinhança da solução é explorada por meio de uma busca local, com o objetivo de melhorar seu valor de custo. A melhor solução obtida durante todas as iterações é retornada como resultado final.

2.1. Heurística Construtiva para o SFP

A heurística construtiva proposta para o SFP assume que não há interseções entre cada par de conjuntos de terminais, isto é, dado um conjunto $\tau = \{T_i \subseteq V | i = 1, \ldots, p\}$ de redes de terminais, $T_i \cap T_j = \emptyset$ para $i \neq j$, $\forall i, j$. Para garantir essa afirmação, aplica-se, como uma fase de pré-processamento, um algoritmo de tempo linear para aglutinar todas as redes de terminais que compartilham ao menos um vértice em comum.

DOI: 10.59254/sbpo-2025-212506
http://live-proceedings.pantheonsite.io/p/212506

O Algoritmo 1 apresenta a heurística construtiva proposta. Primeiramente, um conjunto inicial de pares de vértices terminais é construído utilizando o Algoritmo 2, que forma a lista de candidatos (CL) (linha 2). A CL é ordenada em ordem crescente de custo (linha 4). A heurística construtiva cria a lista restrita de candidatos (RCL) com a porcentagem α dos melhores candidatos da CL (linha 5). Um par é então selecionado aleatoriamente da RCL (linha 6) e removido da CL (linha 7). Na linha 8, o par é conectado utilizando o algoritmo de caminho mínimo de Dijkstra (Cormen et al. [2001]) e o custo de cada aresta utilizada no caminho é zerado (linha 9). A prática de zerar o peso das arestas utilizadas assegura que os próximos pares aproveitem as arestas já usadas, cujo custo já foi contabilizado. Na linha 10, o caminho e o par são adicionados à solução, armazenando-os nas tuplas S_P (caminhos da solução) e S_V (pares da solução), respectivamente.

O Algoritmo 2 gera um conjunto completo de pares. Para cada rede de terminais (linhas 2 a 9), escolhe-se aleatoriamente um nó pivô pertencente à rede (linha 4) e remove-se esse nó do conjunto (linha 5). Posteriormente, seleciona-se aleatoriamente um nó de destino do mesmo conjunto (linha 6), e o par (pivô, destino) é adicionado à lista de pares usada para construir a solução (linha 7). Esse processo continua até que reste apenas um nó no conjunto, o que indica que todos os nós da rede de terminais foram emparelhados.

Algoritmo 1 Heurística_Construtiva(τ, G)

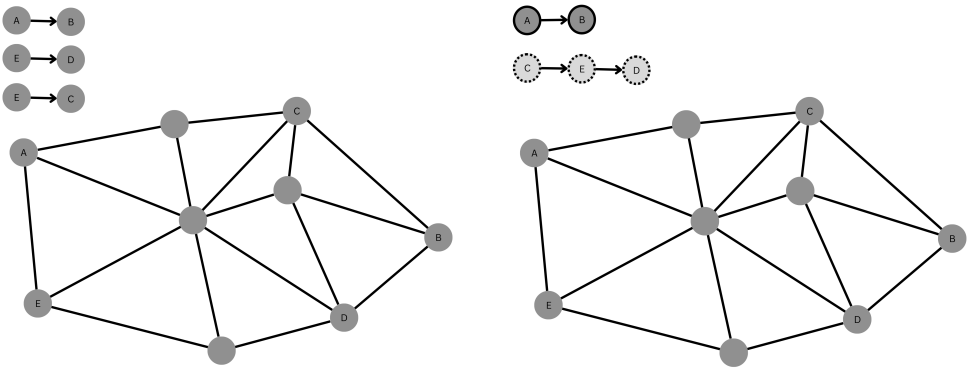
1: $(S_P, S_V) \leftarrow \emptyset$
2: $CL \leftarrow$ Gera_Pares(τ)
3: **Enquanto** $|CL| > 1$ **faça**
4: $CL \leftarrow$ Ordene(CL)
5: $RCL \leftarrow CL \times \alpha$
6: Selecionar aleatoriamente um par $P = (pivot, destination) \in RCL$
7: $CL \leftarrow CL - \{P\}$
8: $path \leftarrow$ Conecte_Usando_Dijkstra(($pivot$, $destination$), G)
9: Zerar o custo das arestas $edge \in$ E, $\forall\, edge \in path$
10: $(S_P, S_V) \leftarrow (S_P \cup \{edge \mid \forall edge \in path\}, S_V \cup \{(pivot, destination)\})$
11: **Fim Enquanto**
12: **Retorne** (S_P, S_V)

Algoritmo 2 Gera_Pares (τ)

1: $pairs \leftarrow \emptyset$
2: **Para** $terminalGroup \in \tau'$ **faça**
3: **Enquanto** $|terminalGroup| > 1$ **faça**
4: Selecionar aleatoriamente $pivot \in terminalGroup$
5: $terminalGroup \leftarrow terminalGroup - \{pivot\}$
6: Selecionar aleatoriamente $destination \in terminalGroup$
7: $pairs \leftarrow pairs \cup \{(pivot, destination)\}$
8: **Fim Enquanto**
9: **Fim Para**
10: **Retorne** $pairs$

As Figuras 1, 2 e 3 ilustram todo o processo de resolução de uma instância do SFP. A Figura 1(a) mostra o grafo da instância juntamente com o conjunto de nós terminais: A, B, C, D e E. Para essa instância, o conjunto de redes de terminais é $\tau = \{T_1, T_2, T_3\}$, onde $T_1 = \{A, B\}$, $T_2 = \{D, E\}$ e $T_3 = \{C, E\}$, ou seja, deve haver um caminho de A para B e vice-versa, de D para E e vice-versa, e de C para E e vice-versa. O processo de aglutinação é iniciado na Figura 1(a). Cada par de terminais com um vértice em comum é agrupado como parte de uma mesma rede de terminais. O resultado é mostrado na Figura 1(b), com os terminais A e B conectados como um

DOI: 10.59254/sbpo-2025-212506
http://live-proceedings.pantheonsite.io/p/212506



(a) Início do processo de aglutinação. (b) Término do processo de aglutinação.

Figura 1: Processo de aglutinação.

conjunto de terminais e C, E e D em um outro conjunto de terminais.

Após o processo de aglutinação, o Algoritmo 2 é executado, conforme ilustrado na Figura 2. O conjunto de terminais com borda em negrito possui um processo de geração de pares simples, pois contém apenas dois terminais. A iteração prossegue para o conjunto de terminais com borda pontilhada, onde se selecionam um pivô e um destino do mesmo conjunto (na figura, os nós C e E foram escolhidos). O pivô é removido do conjunto (nó C, na figura), e repete-se a seleção até restar apenas um nó. A lista final de pares de terminais é mostrada na parte inferior da Figura 2.

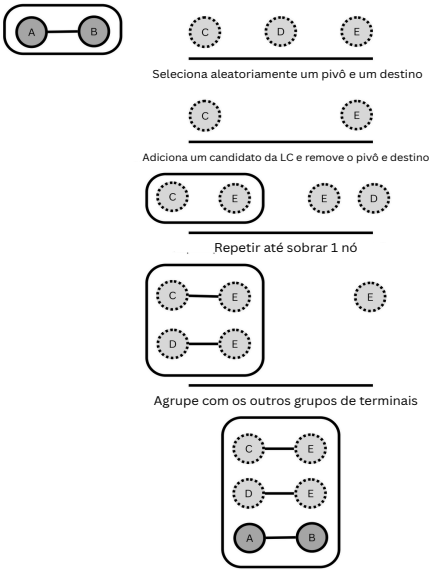
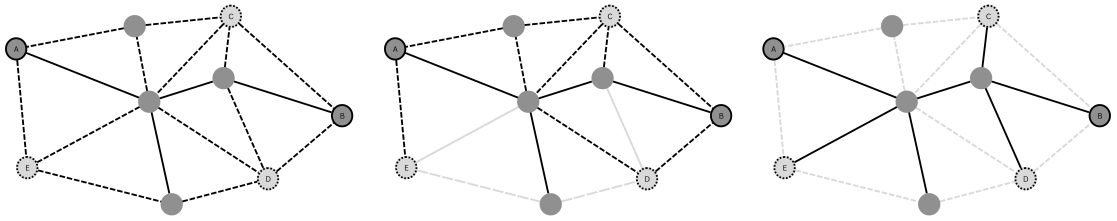


Figura 2: Criação da lista de pares de terminais.

Com a lista de pares criada, inicia-se a execução da fase construtiva (Algoritmo 1). Na primeira iteração, seleciona-se aleatoriamente o primeiro par, por exemplo, A e B, conectando-os com o algoritmo de Dijkstra, conforme mostrado na Figura 3(a). As linhas tracejadas representam arestas não utilizadas e as linhas em negrito representam as arestas do caminho construído.

Na segunda iteração, ilustrada na Figura 3(b), seleciona-se o segundo par, por exemplo, D

DOI: 10.59254/sbpo-2025-212506
http://live-proceedings.pantheonsite.io/p/212506



(a) Inserção do primeiro par. (b) Inserção do segundo par. (c) Última inserção.

Figura 3: Exemplo da fase construtiva.

e E. As linhas cinzas tracejadas representam o caminho original mais curto entre D e E, ignorando arestas já utilizadas. Considerando as arestas ativadas anteriormente, o novo caminho pode reutilizar algumas delas, reduzindo o custo da solução. As arestas adicionadas são destacadas em cinza.

Por fim, na Figura 3(c), o último par de terminais (C e E) é conectado. Nesse caso, a reutilização de arestas é priorizada, necessitando apenas de uma nova aresta para completar o caminho. As arestas não utilizadas estão pintadas em cinza para melhor visualização.

2.2. Busca Local para o SFP

A busca local tem como objetivo trocar arestas utilizadas nos caminhos que conectam os pares. O procedimento, ilustrado no Algoritmo 3, itera sobre todas as arestas utilizadas (linhas 4 a 18). Cada aresta do caminho recebe, temporariamente, uma penalidade correspondente ao maior valor de custo possível (linha 5). Cada par de vértices conectados por essa aresta é desconectado (linha 6), e novos caminhos são construídos (linhas 7 a 11). A função $f()$ calcula o valor de custo da solução somando os pesos de todas as arestas utilizadas. Se a nova solução apresentar um valor de custo melhor, ela é atualizada (linha 13); caso contrário, é mantida a solução original (linhas 15 e 16).

Algoritmo 3 Busca_Local($(S_P, S_V), G$)

1: Copiar (S_P, S_V) para (S'_P, S'_V)
2: Copiar G para G'
3: $cost \leftarrow f(S_P, S_V)$
4: **Para todo** $e \in S'_P$ **faça**
5: Definir o custo da aresta e como infinito em G'
6: Desconectar todos os pares $(pivot, destination)$ afetados pela remoção da aresta e
7: **Para** cada $(pivot, destination)$ afetado **faça**
8: Reconectar $(pivot, destination)$ usando Dijkstra em G'
9: Atualizar S'_P e S'_V com o novo caminho
10: **Fim Para**
11: $newCost \leftarrow f(S'_P, S'_V)$
12: **Se** $newCost < cost$ **então**
13: $(S_P, S_V) \leftarrow (S'_P, S'_V)$
14: $cost \leftarrow newCost$
15: **senão**
16: Reverter G' e (S'_P, S'_V) para os seus valores originais
17: **Fim Se**
18: **Fim Para**
19: **Retorne** (S_P, S_V)

2.3. GRASP-SFP

A heurística GRASP-SFP proposta para o SFP é descrita no Algoritmo 4. Enquanto o número máximo de iterações não for atingido (linhas 4 a 21), uma nova solução é construída (linha

DOI: 10.59254/sbpo-2025-212506
http://live-proceedings.pantheonsite.io/p/212506

5), e um ciclo de busca local é executado enquanto forem obtidas melhorias na solução (linhas 8 a 15). Caso a solução obtida após a fase de busca local apresente um custo melhor (linha 16), a tupla da solução é armazenada em (S_P^*, S_V^*), conforme as linhas 17 e 18. A melhor solução (S_P^*, S_V^*) é, finalmente, retornada após a conclusão de todas as iterações (linha 22).

Algoritmo 4 GRASP-SFP (G, T, max_iter)

1: $(S_P^*, S_V^*) \leftarrow \emptyset$
2: $f^* \leftarrow \infty$
3: $iter \leftarrow 0$
4: **Enquanto** $iter < max_iter$ **faça**
5: $(S_P, S_V) \leftarrow$ Heurística_Construtiva(T, G)
6: $currentCost \leftarrow f((S_P, S_V))$
7: $melhoria \leftarrow$ Verdadeiro
8: **Enquanto** $melhoria$ **faça**
9: $melhoria \leftarrow$ Falso
10: $(S_P, S_V) \leftarrow$ Busca_Local((S_P, S_V))
11: **Se** $f((S_P, S_V)) < currentCost$ **então**
12: $melhoria \leftarrow$ Verdadeiro
13: $currentCost \leftarrow f((S_P, S_V))$
14: **Fim Se**
15: **Fim Enquanto**
16: **Se** $currentCost < f^*$ **então**
17: $(S_P^*, S_V^*) \leftarrow (S_P, S_V)$
18: $f^* \leftarrow currentCost$
19: **Fim Se**
20: $iter \leftarrow iter + 1$
21: **Fim Enquanto**
22: **Retorne** (S_P^*, S_V^*)

3. Resultados Computacionais

Comparamos os resultados obtidos pela heurística construtiva com àqueles alcançados pelo algoritmo primal-dual apresentado em Ghalami e Grosu [2022] a fim de avaliar a robustez da estratégia proposta. As novas heurísticas propostas foram implementadas em C++, utilizando o compilador GNU GCC (versão C/C++ 5.4.0). Todos os experimentos foram realizados em um computador pessoal equipado com um processador Intel® Core$^{\text{TM}}$ i7-8700K CPU @ 3,70 GHz e 16GB de memória RAM, executando o sistema operacional Linux Ubuntu versão 18.04 LTS, com as funcionalidades de processamento paralelo desabilitadas.

O algoritmo primal-dual, conforme apresentado em Ghalami e Grosu [2022] e disponibilizado no GitHub[1], também foi recompilado e executado na mesma arquitetura para realizarmos essa comparação.

3.1. Instâncias

Avaliamos todas as estratégias em um subconjunto de 191 instâncias, disponibilizadas por Ghalami e Grosu [2022], em uma biblioteca pública de instâncias do problema da floresta de Steiner, baseada na biblioteca SteinLib (Koch et al. [2001]). A Tabela 1 apresenta os subconjuntos de instâncias do SFP utilizado em nossos experimentos, juntamente com suas respectivas características. A primeira coluna especifica o tipo de grafo e os pesos das arestas: *esparso* denota uma densidade máxima de 20%; *aleatório* indica que os pesos de arestas gerados aleatoriamente; *incidência* descreve pesos projetados para dificultar o pré-processamento das instâncias de árvores de Steiner; e *roteamento de fios* representa instâncias reais de roteamento de fios VLSI de chips da

[1] https://github.com/lalehghalami/A-Library-of-Approximation-Algorithms-for-Steiner-Forest

DOI: 10.59254/sbpo-2025-212506
http://live-proceedings.pantheonsite.io/p/212506

Tabela 1: Subconjunto de instâncias de SFP proposto por Ghalami e Grosu [2022] e utilizado para avaliar nossas propostas.

Tipo de grafo	Conjunto	#Instâncias	#Vértices	#Arestas	Densidade(%)
esparso (aleatório)	b	18	50 – 100	63 – 200	3,0 – 8,0
esparso (incidência)	i080	20	80	120 – 3160	3,8 – 20,0
esparso (incidência)	i160	34	160	240 – 12720	1,9 – 20,0
esparso (incidência)	i320	52	320	480 – 51040	0,9 – 20,0
esparso (incidência)	i640	65	640	960 – 204480	0,5 – 20,0
roteamento de fios (real)	wrp3	2	84 – 3168	149 – 6220	0,1 – 4,3

IBM ([Ghalami e Grosu, 2022]). As colunas subsequentes fornecem o nome do conjunto, o número de instâncias, o número de vértices, o número de arestas e a densidade do grafo.

Selecionamos as 191 instâncias, exceto as 17 usadas no ajuste do parâmetro α, para comparar os resultados obtidos pelo algoritmo primal-dual [Ghalami e Grosu, 2022] e pelo algoritmo construtivo proposto. Observamos que somente as instâncias do grupo "b" possuem os valores das soluções ótimas conhecidas ([Ghalami e Grosu, 2022]).

As 174 instâncias que não possuem os ótimos conhecidos (isto é, todas as instâncias que não pertencem ao conjunto "b"), foram selecionadas com base na classificação de dificuldade das variantes do problema da árvore de Steiner fornecida em Koch et al. [2001]. As instâncias são classificadas em dois conjuntos principais: *P* (solucionáveis em tempo polinomial) e *NP* (para as quais não se conhece algoritmo de tempo polinomial). O conjunto *NP* é subdividido em categorias: *s* (resolvidas em segundos), *m* (resolvidas em minutos), *h* (resolvidas em horas) e *d* (resolvidas em dias). Para aumentar a dificuldade do *benchmark* de teste, selecionamos apenas instâncias das categorias *NP* (*s*, *m*, *h* e *d*).

3.2. Ajuste de Parâmetros

A heurística GRASP-SFP foi parametrizada utilizando a ferramenta IRACE [Alfaro-Fernández et al., 2017; López-Ibáñez et al., 2016; Pérez Cáceres et al., 2014]. Investigamos o parâmetro α (percentual da CL utilizado para construir o conjunto RCL). O experimento de ajuste do α com o IRACE foi executado por 1000 iterações [Bouamama e Blum, 2017; Maschler et al., 2017] sobre um subconjunto de 17 instâncias (i080-044, i080-244, i080-345, i160-043, i160-213, i160-313, i160-343, i320-014, i320-212, i320-305, i320-342, i640-024, i640-123, i640-215, i640-322, wrp3-52 e wrp3-88) que não foram utilizadas nas comparações subsequentes. Após a execução do IRACE, o melhor valor alcançado para α foi igual a 1,0. O parâmetro α, seus valores possíveis, o incremento fornecido para esse parâmetro e o melhor valor encontrado estão apresentados na Tabela 2.

Tabela 2: Intervalo de valores testados e valor selecionado para o parâmetro α.

Parâmetro	Intervalo de Teste	Melhor Valor	Passo
α	[0,0, 1,0]	1,0	0,1

3.3. Experimentos

Os experimentos seguintes foram realizados para validar a eficiência das estratégias propostas. No primeiro experimento, comparamos os resultados obtidos pelo algoritmo construtivo proposto com àqueles alcançados pelo algoritmo primal-dual apresentado por Agrawal et al. [1995] e implementado por Ghalami e Grosu [2022]. A comparação foi inicialmente realizada com o algoritmo construtivo porque entendemos que é injusto comparar a heurística GRASP-SFP, que executa múltiplas iterações gerando soluções iniciais e as aprimorando por meio de busca local, com o algoritmo primal-dual, que fornece uma única solução aproximada para cada instância.

DOI: 10.59254/sbpo-2025-212506
http://live-proceedings.pantheonsite.io/p/212506

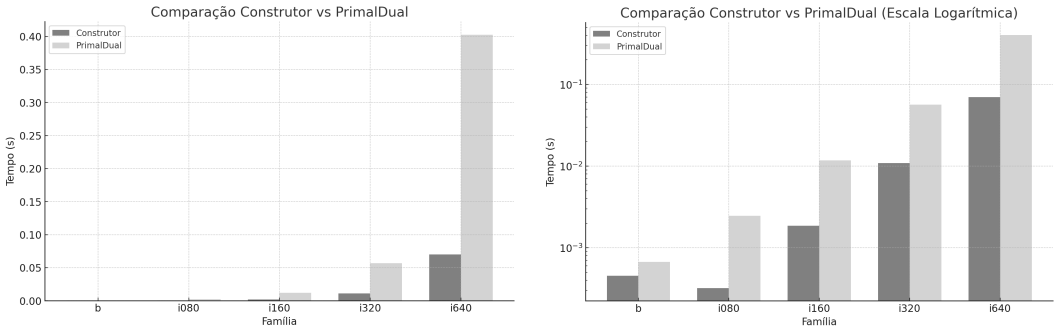
A Tabela 3 resume o experimento supracitado. As comparações são agrupadas por famílias de instâncias, categorizadas nas colunas “b”, “i080”, “i160”, “i320”e “i640”, respectivamente. Os números entre parênteses ao lado de cada nome de família indicam o número total de instâncias consideradas no experimento. Três métricas são utilizadas para comparar as heurísticas: cada linha representa o número de vitórias em melhor solução, o número de vitórias em tempo de execução e o número de empates em melhor solução, respectivamente.

Analisando-se a Tabela 3, os resultados obtidos indicam uma clara superioridade do algoritmo construtivo, perdendo na melhor solução encontrada e tempo de execução em apenas dois casos na família de instâncias “b” e 18 vezes no tempo de execução na família de instâncias “i640”.

Tabela 3: Desempenho geral por família de instâncias na comparação dos resultados obtidos pelo algoritmo Primal Dual e pela heurística construtiva proposta.

	b (18)		i080 (17)		i160 (30)		i320 (48)		i640 (61)	
	Constr.	PrimalDual	Constr.	PrimalDual	Constr.	PrimalDual	Constr.	PrimalDual	Constr.	PrimalDual
Vit. em Melhor	14	2	17	0	34	0	48	0	61	0
Vit. em Tempo	16	2	17	0	34	0	42	6	43	18
Emp. em Melhor	2		0		0		0		0	

A Figura 4 apresenta duas comparações de tempo de execução para cada família de instância, de acordo com as informações das Tabelas 1 e 3, sendo a segunda em escala logarítmica. É possível observar um maior crescimento em tempo gasto para o algoritmo Primal Dual quando comparado ao nosso método construtivo, o que demonstra maior robustez do método construtivo proposto com relação ao aumento do tamanho do grafo e de sua densidade.



(a) Tempo em segundos.

(b) Tempo em segundos, escala logarítmica

Figura 4: Comparação de tempo médio de execução entre cada estratégia para cada família de instância.

Para comparar o desempenho da heurística GRASP-SFP com a heurística construtiva, realizamos um experimento com tempos de execução fixados. Primeiramente, executamos a heurística GRASP-SFP com 10 sementes iniciais distintas para todas as instâncias não utilizadas na fase de ajuste do valor de α. Para cada execução do GRASP-SFP, utilizou-se 200 iterações como critério de parada. Posteriormente, executamos a heurística construtiva proposta até que o tempo médio, calculado sobre os 10 tempos de execução alcançados pela heurística GRASP-SFP, fosse atingido.

A Tabela 4 resume o experimento supracitado, com uma estrutura similar a apresentada na Tabela 3. Desta vez, quatro métricas são utilizadas para comparar as heurísticas: cada linha representa o número de vitórias em qualidade média de solução, o número de vitórias em melhor solução, o número de empates em qualidade média e o número de empates em melhor solução, respectivamente. No caso das comparações de qualidade média de solução, o número entre parênteses indica o número de instâncias em que as vitórias possuem significância estatística.

DOI: 10.59254/sbpo-2025-212506
http://live-proceedings.pantheonsite.io/p/212506

Tabela 4: Desempenho geral por família de instâncias na comparação dos resultados obtidos pelo GRASP-SFP e pela heurística construtiva propostos.

	i080 (17)		i160 (30)		i320 (48)		i640 (61)	
	GRASP-SFP	Constr.	GRASP-SFP	Constr.	GRASP-SFP	Constr.	GRASP-SFP	Constr.
Vit. em Média	5 (4)	0	8 (5)	0	16 (15)	0	11 (9)	0
Vit. em Melhor	3	0	4	0	15	0	9	0
Emp. em Média	12		22		32		50	
Emp. em Melhor	14		26		33		52	

Para indicar significância estatística, empregamos o teste não-paramétrico pareado de Wilcoxon. Este teste estatístico é comumente utilizado para verificar se observações pareadas diferem apenas devido ao acaso (Corder e Foreman [2014]). Neste caso, verifica-se se a diferença entre os dez valores de melhor custo, obtidos usando dez sementes distintas para cada heurística em cada instância, não é aleatória, considerando um *p-valor* inferior a 0,05.

Analisando a Tabela 4, o processo de busca local da heurística GRASP-SFP foi capaz de encontrar melhores soluções médias em 5 casos, sendo 4 deles estatisticamente significativos, para a família "i080", empatando em 12 instâncias. Além disso, o GRASP-SFP alcançou melhores soluções em 3 casos, empatando em 14. Na família "i160", a heurística GRASP-SFP encontrou melhores soluções médias em 8 instâncias, sendo 5 estatisticamente significativas, empatando em 22; também obteve melhores soluções em 4 instâncias, empatando em 26. Para a família de instâncias "i320", a heurística GRASP-SFP encontrou melhores soluções médias em 16 instâncias, sendo 15 delas estatisticamente significativas, empatando em 32; também atingiu melhores soluções em 15 instâncias, empatando em 33. Por fim, na família "i640", a heurística GRASP-SFP encontrou melhores soluções médias em 11 instâncias, sendo 9 delas estatisticamente significativas, empatando em 50; e alcançou melhores soluções em 9 instâncias, empatando em 52. Observa-se um elevado número de empates na maioria das instâncias. Acreditamos que este fato acontece devido ao nível baixo de dificuldade das instâncias, convergindo somente com a utilização do método construtivo presente em ambos os métodos.

4. Conclusões

Nesta investigação, apresentamos uma nova heurística construtiva para o problema da floresta de Steiner em grafos (SFP), que foi, em seguida, empregada como a fase construtiva da heurística GRASP-SFP proposta. Experimentos computacionais em 174 instâncias da literatura mostraram que o método de construção proposto superou as soluções obtidas pela aproximação primal-dual de Agrawal et al. [1995], implementada por Ghalami e Grosu [2022].

Para uma comparação justa entre o algoritmo construtivo e o GRASP-SFP propostos, ambas as heurísticas foram executadas, para 156 instâncias, com tempos médios de execução fixados. Resultados obtidos indicam que, embora a heurística construtiva seja eficaz no encontro de boas soluções iniciais, o GRASP-SFP a superou em diversas famílias de instâncias, alcançando melhores soluções médias e melhores valores de custo de soluções em um número significativo de casos. Ademais, observou-se que as diferenças entre as soluções obtidas apresentaram significância estatística em diversas instâncias.

Embora a heurística GRASP-SFP supere a construtiva, o elevado número de empates ressalta a necessidade de experimentos realizados sobre um conjunto de instâncias mais desafiador, visando avaliar mais a utilização, eficiência e capacidade de exploração do espaço de soluções pela busca local. Porém, os resultados deste estudo contribuem para a literatura sobre o SFP ao apresentar uma heurística construtiva robusta e uma heurística GRASP-SFP promissora.

DOI: 10.59254/sbpo-2025-212506
http://live-proceedings.pantheonsite.io/p/212506

Agradecimentos

O desenvolvimento deste trabalho contou com o apoio da FAPERJ (E-26/201.139/2022) e do CNPq (315750/2021-9, 309433/2021-5, 316625/2021-3 e 304680/2023-0).

Referências

Agrawal, A., Klein, P., e Ravi, R. (1995). When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM Journal on Computing*, 24:440–456.

Alfaro-Fernández, P., Ruiz, R., Pagnozzi, F., e Stützle, T. (2017). Exploring automatic algorithm design for the hybrid flowshop problem. In *Proceedings of the 12th Metaheuristics International Conference*, p. 201–203.

Angelopoulos, S. (2006). The node-weighted steiner problem in graphs of restricted node weights. In Arge, L. e Freivalds, R., editors, *Algorithm Theory – SWAT 2006*, p. 208–219, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bouamama, S. e Blum, C. (2017). A population-based iterated greedy algorithm for the knapsack problem with setup. In *Proceedings of the 12th Metaheuristics International Conference*, p. 558–565.

Corder, G. W. e Foreman, D. I. (2014). *Nonparametric Statistics: A Step-by-Step Approach*. John Wiley & Sons, Inc., New Jersey, NY, USA.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., e Stein, C. (2001). *Introduction to Algorithms*. The MIT Press, 2nd edition.

Dilkina, B. e Gomes, C. P. (2010). Solving connected subgraph problems in wildlife conservation. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, p. 102–116. Springer.

Feige, U. (1998). A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45:634–652.

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

Gassner, E. (2010). The steiner forest problem revisited. *Journal of Discrete Algorithms*, 8:154–163.

Gendreau, M. e Potvin, J.-Y. (2010). *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*. Springer, 2nd edition.

Ghalami, L. e Grosu, D. (2022). Approximation algorithms for Steiner forest: An experimental study. *Networks*, 79:164–188.

Goemans, M. X. e Williamson, D. P. (1995). A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24:296–317.

Groß, M., Gupta, A., Kumar, A., Matuschke, J., Schmidt, D. R., Schmidt, M., e Verschae, J. (2018). A Local-Search Algorithm for Steiner Forest. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, p. 31:1–31:17.

DOI: 10.59254/sbpo-2025-212506
http://live-proceedings.pantheonsite.io/p/212506

Guha, S., Moss, A., Naor, J. S., e Schieber, B. (1999). Efficient recovery from power outage (extended summary). In *Proceedings of the thirty-first annual ACM Symposium on Theory of Computing, STOC*, p. 574–582.

Gupta, A. e Kumar, A. (2015). Greedy algorithms for steiner forest. In *Proceedings of the forty-seventh annual ACM Symposium on Theory of Computing, STOC*, p. 871–878.

Koch, T., Martin, A., e Voß, S. (2001). Steinlib: An updated library on steiner tree problems in graphs. In Cheng, X. Z. e Du, D.-Z., editors, *Steiner Trees in Industry*, p. 285–325. Springer US, Boston, MA.

Li, X., Xu, X.-H., Zou, F., Du, H., Wan, P., Wang, Y., e Wu, W. (2009). A ptas for node-weighted steiner tree in unit disk graphs. In Du, D.-Z., Hu, X., e Pardalos, P. M., editors, *Combinatorial Optimization and Applications*, p. 36–48, Berlin, Heidelberg. Springer Berlin Heidelberg.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., e Stützle, T. (2016). The IRACE package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Magnanti, T. L. e Raghavan, S. (2005). Strong formulations for network design problems with connectivity requirements. *Networks*, 45:61–79.

Mahjoub, A. R., Poss, M., Simonetti, L., e Uchoa, E. (2019). Distance transformation for network design problems. *SIAM Journal on Optimization*, 29:1687–1713.

Maschler, J., Hackl, T., Riedler, M., e Raidl, G. R. (2017). An enhanced iterated greedy metaheuristic for the particle therapy patient scheduling problem. In *Proceedings of the 12th Metaheuristics International Conference*, p. 465–474.

Moldenhauer, C. (2013). Primal-dual approximation algorithms for node-weighted steiner forest on planar graphs. *Information and Computation*, 222:293–306. ISSN 0890-5401. 38th International Colloquium on Automata, Languages and Programming (ICALP 2011).

Moss, A. e Rabani, Y. (2001). Approximation algorithms for constrained for constrained node weighted steiner tree problems. In *Proceedings of the thirty-first annual ACM Symposium on Theory of Computing, STOC*, p. 373–382.

Pajor, T., Uchoa, E., e Werneck, R. F. (2018). A robust and scalable algorithm for the steiner problem in graphs. *Mathematical Programming Computation*, 10:69–118.

Pérez Cáceres, L., López-Ibáñez, M., e Stützle, T. (2014). An analysis of parameters of irace. In Blum, C. e Ochoa, G., editors, *Evolutionary Computation in Combinatorial Optimisation*, p. 37–48, Berlin, Heidelberg. Springer Berlin Heidelberg.

Remy, J. e Steger, A. (2005). Approximation schemes for node-weighted geometric steiner tree problems. In Chekuri, C., Jansen, K., Rolim, J. D. P., e Trevisan, L., editors, *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, p. 221–232, Berlin, Heidelberg. Springer Berlin Heidelberg.

Schmidt, D., Zey, B., e Margot, F. (2021). Stronger mip formulations for the steiner forest problem. *Math. Program.*, 186:373–407.

DOI: 10.59254/sbpo-2025-212506
http://live-proceedings.pantheonsite.io/p/212506