UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
UNIDADE ACADÊMICA DE GARANHUNS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FELIPE TENÓRIO DE HOLANDA ROCHA LIBÓRIO

Algoritmos Exatos e Heurísticos para os Problemas de Steiner e de Conexão de Terminais com Número Restrito de Roteadores e Elos

GARANHUNS

2019

Felipe Tenório de Holanda Rocha Libório

Algoritmos Exatos e Heurísticos para os Problemas de Steiner e de Conexão de Terminais com Número Restrito de Roteadores e Elos

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Unidade Acadêmica de Garanhuns da Universidade Federal Rural de Pernambuco.

Orientador:

Prof. Dr. Rian Gabriel Santos Pinheiro

Garanhuns, 18 de julho de 2019

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema Integrado de Bibliotecas da UFRPE
Biblioteca Ariano Suassuna, Garanhuns-PE, Brasil

L696a Libório, Felipe Tenório de Holanda Rocha
Algoritmos Exatos e Heurísticos para os Problemas de Steiner e de Conexão de Terminais com Número Restrito de Roteadores e Elos / Felipe Tenório de Holanda Rocha Libório. - 2019.
60 f. ; il.
Orientador: Rian Gabriel Santos Pinheiro.
Trabalho de Conclusão de Curso (Graduação em Ciência da Computação)-Universidade Federal Rural de Pernambuco, Departamento de Ciência da Computação, Garanhuns, BR-PE, 2019.
Inclui referências.
1. Computação 2. Algoritmos 3. Meta-heurísticas
I. Pinheiro, Rian Gabriel Santos, orient. II. Título

CDD 004

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Unidade Acadêmica de Garanhuns da Universidade Federal Rural de Pernambuco, aprovada pela comissão examinadora que abaixo assina.

Prof. Dr. Rian Gabriel Santos Pinheiro - Orientador
Instituto de Computação
UFAL

Prof. Dr. Jean Carlos Teixeira de Araujo - Examinador
UAG
UFRPE

Prof. Dr. Tiago Buarque Assunção de Carvalho - Examinador
UAG
UFRPE

Garanhuns, 18 de julho de 2019

À Kyoto Animation.

Agradecimentos

Agradeço ao meu pai, por não ter cobrado nada de mim durante este cruciante e infindo período universitário.

A meu orientador, Rian Pinheiro, por ter seguido como tal mesmo após ter deixado a UAG para ir para outra universidade. Ele definitivamente fez um esforço além do esperado e me ajudou bastante na construção deste trabalho.

A Jean de Araujo e Tiago de Carvalho, membros da banca, por aceitarem este papel tão em cima da hora e pelas correções sugeridas ao texto.

Aos amigos que fiz durante a graduação que tornaram esta fase menos insuportável.

Aos demais professores do curso que, de uma forma ou de outra, contribuíram para minha formação.

Mas, acima de tudo, agradeço a Akira Toriyama e sua obra por me despertarem o gosto por animação japonesa. A Wataru Watari por Oregairu; Isin Nisio e Akiyuki Shinbō por Monogatari Series; Yoshihiru Togashi, por HunterxHunter; Hideaki Anno, por *Evangelion*; e Hiromu Arakawa por *Fullmetal Alchemist*. Agradeço à *Shaft* e à *Kyoto Animation* por sua imensa contribuição a esta indústria. A *From Software* e Hidetaka Miyazaki por *Dark Souls III*, mesmo que seus outros jogos tenham de desapontado. A *Irrational Games* e Ken Levine por *Bioshock*. À *Rockstar* por *Grand Theft Auto*. A Fireaxis e 2K Games por *Civilization VI* e a Sid Meier por ter criado esta franquia. E até mesmo à Bethesda, por Skyrim, mesmo que ela tenha, majoritariamente, trazido decepções desde então. Pra quem vê de fora, pode parecer que consumir estas obras atrapalhou meu desempenho mas, na realidade, foram elas as principais responsáveis, além de eu mesmo, por me levar ao fim do curso.

「偽物のほうが圧倒的に価値があるてな。
そこに本物にあるという意思があるだけ偽物のほうが本物よりが本物だ」

– 貝木 泥舟

“O falso é esmagadoramente mais valioso.
Em sua vontade deliberada de ser real, ele é mais real que o real.”
– Deishū Kaiki

Resumo

Neste trabalho foram apresentadas soluções para o Problema de Conexão de Terminais com Número Restrito de Roteadores e Elos (TCP, do inglês *Terminal Connection Problem*). O TCP consiste em encontrar uma árvore que conecte um subconjunto de vértices de um dado grafo. Ele se diferencia do problema de Steiner pela introdução de uma restrição adicional ao número de vértices de Steiner que podem fazer parte de uma solução. O TCP pode ser aplicado nos mesmos tipos de problema em que se pode fazer uso do problema de Steiner, o que inclui: projetos de circuitos VLSI; roteamento *multicast*; modelar e resolver problemas de planejamento em telecomunicações; e distribuição de eletricidade. Como o TCP é uma generalização do problema de Steiner em grafos, também foram feitos testes em instâncias deste problema. Além disso, foram geradas diversas instâncias de diferentes níveis de dificuldade para o TCP, que trabalhos posteriores poderão utilizar para a comparação da performance de futuras soluções. Os resultados obtidos nas instâncias do Problema de Steiner foram comparados ao de outra solução da literatura e a meta-heurística utilizada na solução, a *Large Neighborhood Search* (LNS), se mostrou viável como uma forma simples e de baixo custo computacional, tanto em tempo de execução quanto em requisitos de memória, de se obter resultados satisfatórios para este problema. Conseguindo uma taxa de erro média abaixo de 2% nas instâncias avaliadas, a depender do tempo de execução dado ao algoritmo. Além disso, este é o primeiro trabalho a apresentar soluções para o TCP de que se tem conhecimento. Foi implementada, além da solução meta-heurística, uma solução exata com o uso do *solver* IBM Ilog CPLEX. Para as instâncias avaliadas cujos valores ótimos foram encontrados pela solução exata apresentada, a implementação da LNS obteve taxa de erro média de 0,66% ao mesmo tempo em que apresentou um tempo de execução 22 vezes menor que a solução exata e conseguiu obter o valor ótimo em 14 das 18 instâncias testadas. Os resultados obtidos na resolução das instâncias do TCP, juntamente às próprias instâncias geradas, formaram uma base para comparações de soluções futuras que possam vir a ser propostas para o problema.

Palavras-chave: Problema de Steiner em Grafos. Meta-heurísticas. Problema de Conexão de Terminais.

Abstract

This work presents solutions for the Terminal Connection Problem with Bounded Number of Routers and Links (TCP). The TCP consists in finding a spanning tree to a subset of vertices of a given graph. It differentiates itself from the Steiner's problem by having additional restrictions to the number of Steiner nodes allowed in a solution. The TCP can be applied in the same types of problems on which you can make use of the Steiner's problem, which includes: VLSI circuit project; multicast routing; to model and solve telecomunications planning problems; and electricity distribution. As the TCP is a generalization on the Steiner's problem in graphs, tests in instances of this problem were also made. Moreover, a multitude of instances of different difficulty levels was generated for the TCP, instances which posterior works on the problem will be ablem to use for comparing the performance of future solutions. The results obtained for the Steiner Problem instances were compared to those of another solution found in the literature, and the metaheuristic utilised to solve them, the Large Neighborhood Search (LNS), has shown to be viable as a simple and low cost, both in time and in memory requirements, way of reaching satisfactory results for this problem. Achieving a mean error below 2% for the evaluated instances, depending on the time given for the aolgorithm. Furthermore, this is the first known work to present a solution to the TCP. Besides the LNS metaheurístic, an exact solution was implemented by the means of the IMB ILOG CPLEX solver. For the tested instances whose optimal values were found by the presented exact solution, the LNS implementation managed to get an mean error rate of 0.66% meanwhile having a run time 22 times smaller than that of the exact solution and found the optimal value in 14 out of the 18 tried instances. The results obtained on the solving of the TCP instances, alongside the generated instances themselves, have formed a base for the comparison of future solutions that may come to be proposed for this problem.

Keywords: Steiner Problem in Graphs. Metaheuristics. Terminal Connection Problem.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviaturas e Siglas

CMSA	*Construct Merge Solve & Adapt*
GRASP	*Greedy Randomized Adaptive Search Procedure*
LNS	*Large Neighborhood Search*
PH-GRASP	*Parallel Hybrid GRASP*
TCP	*Terminal Connection Problem*
VLNS	*Very Large Neighborhood Search*

Sumário

Capítulo 1

Introdução

Otimização combinatória é o processo de buscar pelo máximo (ou mínimo) de uma função objetivo cujo domínio é um espaço de configuração [30], ou seja, possui um conjunto finito de possíveis soluções [1].

O espaço de soluções viáveis normalmente é muito grande para se realizar uma busca por força bruta. Alguns problemas, de tempo polinomial determinístico, podem ser solucionados em tempo viável com o uso de algoritmos exatos. Em outros casos, problemas de tempo polinomial não-determinístico, não há solução ótima de tempo polinomial conhecida. Assim, para instâncias grandes destes problemas, busca-se usar técnicas que gerem soluções aproximadas de boa qualidade e que possam ser executadas em tempo hábil.

O Problema de Conexão de Terminais com Número Restrito de Roteadores e Elos (TCP, do inglês *Terminal Connection Problem*) é um destes problemas. Ele generaliza um problema clássico de otimização combinatória, o Problema de Steiner em Grafos.

O Problema da Árvore Steiner em Grafos consiste em encontrar uma árvore de peso mínimo que conecte um subconjunto de vértices (terminais) de um grafo. Além dos terminais, uma árvore Steiner admite vértices opcionais, chamados vértices de Steiner.

Seja $G = (V, E)$ um grafo não direcionado com arestas e_i de peso não negativo c_i, $i \in \{1, \ldots, |E|\}$, e seja $S \subseteq V$ um subconjunto de terminais não propriamente contido em V. Uma árvore Steiner é uma árvore em G que se projeta sobre S. A versão de decisão do problema (dizer se há ou não uma árvore Steiner para uma dada instância do problema) é um dos 21 problemas NP-Completos de Karp [18], consequentemente, a versão de otimização (encontrar a árvore) é um problema NP-Difícil.

O Problema de Steiner em Grafos é um generalização de dois problemas clássicos, Caminho Mínimo e Árvore Geradora Mínima. Note que se $S = \{s, t\}$, então o problema consiste em encontrar o caminho mínimo de s a t. Além disso, se $S = V$, então o problema consistem em encontrar o Árvore Geradora Mínima de G.

A partir do problema de Steiner em grafos, Dourado et al. [12] propuseram o TCP. Ele difere em relação ao problema de Steiner ao incluir restrições ao número máximo de

vértices de Steiner que uma solução pode conter. Adicionando limites ao número de *elos* (l) e de *roteadores* (r) aceitos em uma solução. O problema é definido a seguir.

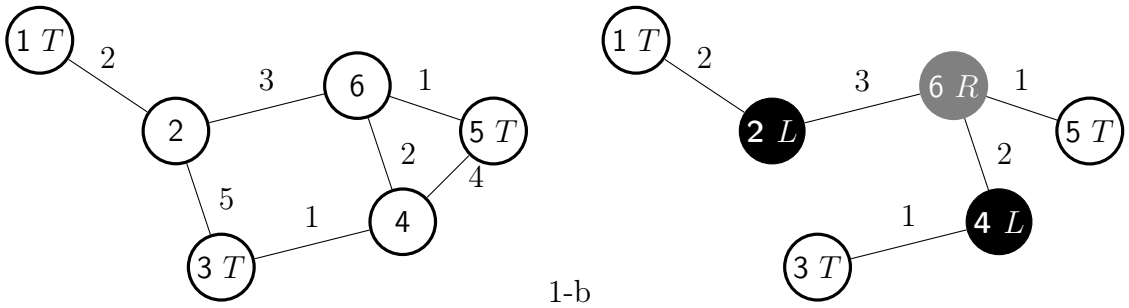
Dado um grafo conexo e ponderado G, uma *árvore de conexão* para um subconjunto $W \subseteq V(G)$ é um subgrafo acíclico e conexo $T \subseteq G$, onde $W \subseteq V(T)$ e todas as folhas de T estão em W. Uma árvore de conexão pode, então, ser vista como uma árvore Steiner sem a restrição de peso mínimo. Os vértices de uma árvore de conexão podem ser classificados como: *terminais*, os vértices de W, ou seja, aqueles que se quer conectar; *elos*, os vértices que estão em $V(T) \setminus W$ e tem grau dois, ou seja, os vértices opcionais de grau dois; e *roteadores*, os vértices em $V(T) \setminus W$ com grau três ou mais.

A Figura 1.1 ilustra uma árvore de conexão, na Figura 1.1-b, para um subconjunto de três vértices terminais do grafo apresentado na Figura 1.1-a. Esta também pode ser considerada uma árvore Steiner, pois o peso resultante do grafo gerado para a conexão dos terminais é mínimo. O grafo resultante possui dois elos e um roteador.

Dourado et al. [12] propuseram limitar o número de elos e roteadores em uma árvore de conexão definindo assim o *Problema de Conexão de Terminais com Número Restrito de Roteadores e Elos* como:

> Problema de Conexão de Terminais - TCP
> **Instância:** Um grafo conexo G, um subconjunto $W \subseteq V(G)$, e dois inteiros não negativos l e r.
> **Questão:** G contém uma árvore de conexão T com até l elos e r roteadores?

Dourado et al. [12] provaram que o problema de decisão do TCP é NP-Completo mesmo quando um dos parâmetros l ou r é fixo. Porém pode ser resolvido em tempo polinomial quando ambos l e r são fixos.



1-a 1-b

Figura 1.1: Grafo cujos nós 1, 3 e 5 são terminais. Na Figura 1-b é apresentada uma árvore de conexão para a Figura 1-a, onde os nós 2 e 4 são elos e o nó 6 é um roteador.

As aplicações do TCP são as mesmas do Problema de Steiner. Problemas que possam ser modelados como a minimização do peso de uma árvore de conexão, como aqueles relacionados a circuitos digitais ou redes, podem ser resolvidos com o TCP. O que inclui: projeto de circuitos VLSI [11], roteamento *multicast* [19] ou para modelar e resolver problemas de planejamento em telecomunicações e distribuição de eletricidade [10].

1.1 Motivação e Objetivos

A motivação deste trabalho é suprir a ausência de soluções e instâncias para o TCP, pois não foi possível encontrar na literatura trabalhos que preencham esta lacuna. Os trabalhos que tratam do TCP disponíveis na literatura são de teor teórico e, ao conhecimento dos autores, este é o primeiro trabalho que apresenta algoritmos para o problema.

Sendo assim, este trabalho tem como objetivo apresentar soluções para o TCP, além de gerar instâncias de testes que formarão uma base para trabalhos futuros. Seus objetivos específicos são:

- Implementar uma meta-heurística para o TCP e para o Problema de Steiner;
- Implementar uma solução exata para o TCP e para o Problema de Steiner;
- Avaliar a qualidade das soluções obtidas para o Problema de Steiner com a solução meta-heurística implementada;
- Gerar instâncias de diferentes níveis de complexidade para o TCP;
- Gerar soluções para as instâncias criadas, de modo a constituir uma base de dados com a qual trabalhos futuros poderão ser comparados.

1.2 Trabalhos Relacionados

Não há muitos trabalhos sobre o TCP na literatura, porém, entre os que se pode encontrar, é válido referenciar os seguintes trabalhos: *Design of connection networks with bounded number of non-terminal vertices* de Dourado et al. [13], que primeiro definiu o problema; Conexão de Terminais com Número Restrito de Roteadores e Elos de Dourado et al. [12], que apresentou uma versão estrita do problema; e Conexão de Terminais com Limitação de Roteadores: Complexidade e Relação com Fluxos e Caminhos Disjuntos de Melo [25], que analisou a complexidade da versão estrita do problema e apresentou uma solução de tempo polinomial para esta versão do problema quando o número de roteadores é 0 ou 1.

1.3 Estrutura do Trabalho

A estrutura para o restante do trabalho está organizada como segue: o Capítulo 2 apresenta a fundamentação teórica, contendo os conceitos necessários para o desenvolvimento do trabalho; o Capítulo 3: descreve os algoritmos propostos para o problema; o Capítulo 4: exibe e discute os resultados obtidos; e o Capítulo 5: apresenta conclusões e trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo serão apresentados os conceitos necessários ao entendimento deste trabalho: complexidade computacional, otimização combinatória e meta-heurísticas.

2.1 Complexidade Computacional

A teoria da complexidade computacional foca na classificação de problemas de acordo com a sua dificuldade e em relacionar estas classes umas às outras. Uma das suas funções é determinar os limites práticos do que computadores podem ou não fazer.

2.1.1 Decisão vs Otimização

Um problema de Decisão, na teoria da complexidade computacional, se refere a um problema cuja resposta seja da forma SIM ou NÃO para um conjunto infinito de entradas, por exemplo, o problema de decidir se um número é ou não é primo.

Um problema de otimização, por sua vez, é o tipo de problema cuja resposta consiste em encontrar a melhor solução dentre todas as possíveis soluções. Problemas de otimização podem ser classificados de acordo com o tipo das suas variáveis, contínuas ou discretas. Em um problema de otimização discreta, ou combinatória, busca-se por um objeto como um inteiro, uma permutação ou um subconjunto que satisfaça alguma propriedade. Já os problemas com variáveis contínuas são modelados usando números reais.

2.1.2 P

P é uma classe de complexidade fundamental na teoria da complexidade computacional. Ela contém todos os problemas de decisão que podem ser resolvidos em tempo polinomial por uma máquina de Turing determinística.

Uma linguagem L pertence a P se, e somente se, existe uma máquina de Turing determinística M, tal que:

- M é executada em tempo polinomial para todas as entradas;
- Para todo x em L, M retorna 1;
- Para todo x fora de L, M retorna 0.

Em outras palavras, a classe P é a classe de todos os problemas de decisão que admitem algoritmos de tempo polinomial para sua solução.

2.1.3 NP

NP (tempo polinomial não-determinístico) é a classe de complexidade que contém o conjunto de problemas de decisão para os quais as respostas SIM podem ser verificadas em tempo polinomial [20].

É fácil ver que a classe P está contida na classe NP. Pois, se um problema pode ser solucionado em tempo polinomial, então, uma solução SIM pode ser verificada em tempo polinomial pela simples resolução do problema. Porém NP contém problemas mais difíceis, sendo os mais difíceis entre eles chamados de NP-Completos. Um algoritmo capaz de resolver um destes problemas em tempo polinomial seria capaz de resolver qualquer outro problema em NP em tempo polinomial.

2.1.4 P versus NP

O problema P *versus* NP é um importante problema ainda não solucionado da ciência da computação. Ele busca saber se qualquer problema cuja solução pode ser verificada em tempo polinomial pode também ser resolvido em tempo polinomial. Implicando em P = NP.

O problema foi formalmente apresentado em 1971 por Stephen Cook em *"The complexity of theorem proving procedures"*[9] e é comumente considerado como o mais importante problema aberto na ciência da computação [14]. Ele é um dos sete *Millenium Prize Problems*, selecionados pelo Clay Mathematics Institute, cada um traz um recompensa de um milhão de dólares para quem primeiro o solucionar.

Uma resposta negativa para esse problema, $P \neq NP$, significaria que há problemas que podem ter suas soluções verificadas em tempo polinomial mas não podem ser solucionados em tempo polinomial.

Uma solução para este problema, seja afirmativa ou negativa, traria grandes implicações para vários campos como matemática, criptografia, pesquisa de algoritmos, teoria dos jogos e muitos outros [15].

2.1.5 NP-Completo

Problemas *NP-Completos* são problemas de decisão aos quais se pode reduzir qualquer problema NP em tempo polinomial e cuja solução pode ser verificada em tempo polinomial. Ou seja, qualquer problema em NP pode ser transformado, em tempo polinomial, em qualquer um dos problemas NP-Completos. Pode-se dizer que um problema NP-Completo é qualquer um problema em NP que seja ao menos tão difícil quanto qualquer outro problema em NP. Se qualquer problema NP-Completo estiver em P, então $P = NP$.

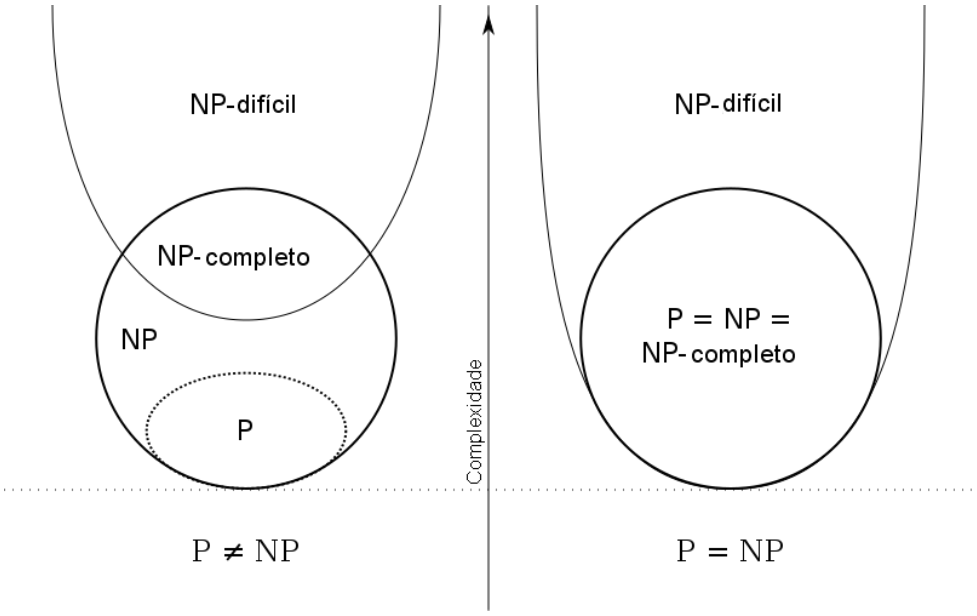
Segue a definição formal da classe NP-Completo [21]. Um problema de decisão π é NP-Completo se:

1. π está em NP;

2. todo problema em NP é redutível a π em tempo polinomial.

Apenas problemas de decisão podem pertencer à classe NP-completo, problemas de otimização podem pertencer à classe *NP-Difícil*. Problemas NP-Difíceis são aqueles que são ao menos tão difíceis quanto problemas NP mas não precisam ter as soluções verificáveis em tempo polinomial. Qualquer problema NP pode ser reduzido em tempo polinomial para um problema NP-Difícil. Ou seja, todo problema NP-Completo é NP-Difícil mas nem todo problema NP-Difícil é NP-Completo. A Figura 2.1 apresenta uma representação visual da relação entre as diferentes classes de problemas.

Figura 2.1: Diagrama de Euler para P, NP, NP-Completo e NP-Difícil[2]



2.2 Otimização Combinatória

Problemas de otimização combinatória consistem em encontrar uma solução que maximize ou minimize uma função objetivo cujo domínio é um conjunto finito de objetos.

Otimização combinatória é um subconjunto da otimização matemática que está relacionado à pesquisa operacional, algoritmos e complexidade computacional. E tem aplicações em diversas áreas como projeto de redes, alocação de recursos ou transportes [4].

Definição

Dada uma instância I; um conjunto S de soluções viáveis para essa instância e uma função objetivo $f(x)$ de x, em que $x \in S$ é uma solução viável. O objetivo do problema é encontrar uma solução viável $x^* \in S$ que minimiza (ou maximiza) a função objetivo. Ou seja,

$f(x^*) \leq f(x) \quad \forall x \in S.$

A solução x^* é denominada solução ótima, e possui o menor custo entre todas as soluções possíveis para a instância I.

Exemplos de problemas de otimização combinatória, além dos abordados neste trabalho, são: o problema do caixeiro viajante, o problema da mochila e o problema da cobertura de conjuntos.

2.3 *Branch-and-Cut*

O algoritmo *Branch-and-Cut* é um algoritmo clássico de otimização combinatória que pertencente ao grupo dos algoritmos exatos. Ele é derivado do algoritmo *Branch-and-Bound* que consiste em uma enumeração das possíveis soluções do problema acoplada com um mecanismo de poda que elimina ramos infrutíferos da árvore de enumeração. O que difere os dois algoritmos é o fato do *Branch-and-Cut* possuir um mecanismo que permite que novas restrições possam ser adicionadas ao modelo de acordo com a necessidade. Dessa forma, no modelo proposto, as restrições de subciclo não são inicializadas no modelo, assim, toda vez que o algoritmo encontra uma solução para o problema, é realizado um teste que verifica se a solução é viável ou se ela contém um ciclo T. Caso o ciclo seja encontrado, então a restrição correspondente é adicionada ao modelo, eliminando assim essa solução.

2.4 Meta-heurísticas

Meta-heurísticas são originalmente definidas como métodos de solução que arranjam uma interação entre procedimentos locais de aprimoramento e estratégias em um nível

mais amplo para criar um processo que consiga fugir de ótimos locais (as melhores soluções dentro de conjuntos de soluções próximas entre si) e realizar uma busca robusta em um espaço de solução [16]. Com o tempo esses métodos passaram a também incluir quaisquer procedimentos que empreguem estratégias para fugir de ótimos locais em espaços de solução complexos. Especialmente aqueles procedimentos que utilizam uma ou mais estruturas de vizinhança como forma de definir movimentos admissíveis ao se transicionar de uma solução à outra, ou para construir e destruir soluções em processos construtivos e destrutivos.

Várias ferramentas e mecanismos que surgiram a partir da criação de métodos meta-heurísticos se provaram altamente efetivos, de modo que meta-heurísticas se tornaram a maneira preferida de resolver vários tipos de problemas complexos, particularmente aqueles de natureza combinatória. Apesar de meta-heurísticas não garantirem uma solução ótima, métodos exatos frequentemente se mostraram incapazes de encontrar uma solução de qualidade em tempo hábil. Especialmente em problemas reais, que geralmente possuem alto nível de complexidade, as meta-heurísticas de ponta se mostram muito mais aptas. Além disso, algumas das mais frutíferas aplicações de métodos de solução exatos surgiram com a incorporação de estratégias heurísticas dentro deles. O que motivou a pesquisa e o uso de métodos aprimorados.

2.4.1 Classificação

Há uma ampla gama de meta-heurísticas e diferentes formas de as classificar. As principais formas são [8]:

Solução única ou baseada em população — Abordagens de solução única focam em modificar e aprimorar uma única solução candidata; exemplos de meta-heurísticas desta classe são *Iterated Local Search*, *Variable Neighborhood Search* e *Guided Local Search*. Abordagens baseadas em população mantém e melhoram múltiplas soluções candidatas, frequentemente guiando as buscas por características populacionais. Exemplos são: *Evolutionary Computation*, Colônia de Formigas e Algoritmos Genéticos [32]. Essas duas formas diferem também pelo tipo de estratégia de busca. Um tipo de estratégia de busca é um aprimoramento de algoritmos de busca local simples, como o *Hill Climbing*, usado para encontrar ótimos locais.

Hibridização e algoritmos meméticos — Uma meta-heurística híbrida combina uma meta-heurística a outras abordagens, como programação linear ou aprendizagem de máquina. Como exemplo existe a CMSA que funciona reduzindo a complexidade da instância com o uso de um método heurístico e então resolvendo esta instância reduzida com o uso de um método exato [7]. Especificamente, a união de uma meta-heurística com um método de programação linear (inteira) é chamada matheurística.

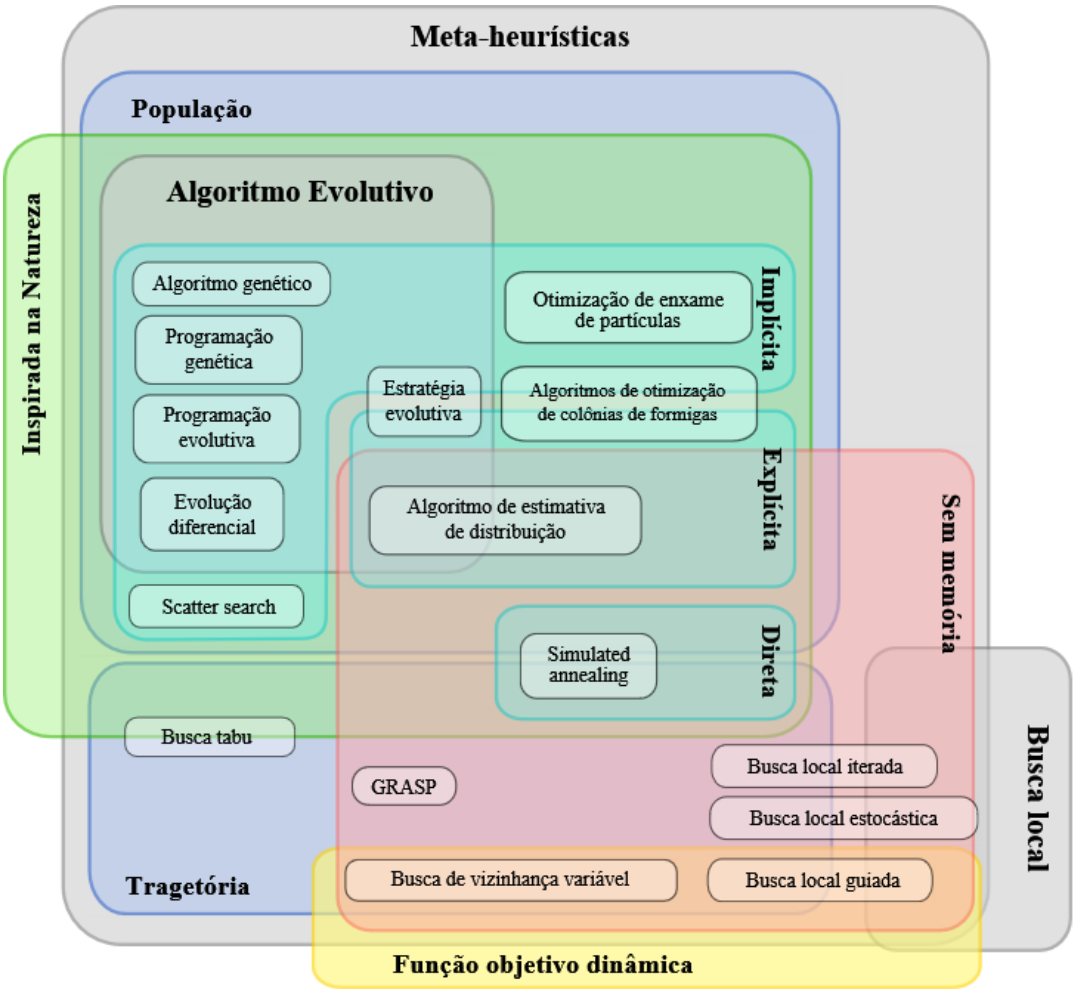
Algoritmos meméticos são meta-heurísticas baseadas em população que incorporam

conceitos e estratégias de busca local de diferentes meta-heurísticas de acordo com o ciclo de gerações, com o objetivo de aproveitar melhor os pontos positivos de cada uma [26, 27]. Basicamente, são algoritmos genéticos com busca local.

Meta-heurísticas inspiradas pela natureza — são meta-heurísticas que fazem analogias a comportamentos observados na natureza. Elas buscam na natureza conceitos, mecanismos e princípios para projetar sistemas que lidem com problemas computacionais complexos [17]. Exemplos destas meta-heurísticas são Otimização por Colônia de Formigas, Otimização por Enxame de Partículas e algoritmos evolutivos.

A Figura 2.2 mostra um diagrama de Euler para as diferentes classificações de meta-heurísticas.

Figura 2.2: Diagrama de Euler das diferentes classificações de meta-heurísticas[3]



2.4.2 *Large Neighborhood Search*

Large Neighborhood Search (LNS) é uma meta-heurística cujos métodos de busca funcionam explorando uma vizinhança complexa com o uso de heurísticas [28].

Em uma LNS, uma solução é gradualmente aprimorada pela destruição e reparação alternadas de uma solução viável. A LNS pertence à classe de heurísticas *Very Large Scale Neighborhood Search*(VLNS). Todos os algoritmos VLNS se baseiam em realizar buscas em uma grande vizinhança visando encontrar um ótimo local de boa qualidade. Para se reduzir o longo tempo necessário às buscas em grandes vizinhanças, técnicas de filtragem são utilizadas e restringem a vizinhança a um subconjunto de soluções que podem ser eficientemente analisadas. Em uma LNS, a vizinhança é implicitamente definida por métodos, geralmente heurísticas, usados para destruir e reparar a solução corrente [28].

Neighborhood Search

Dada uma instância I de um problema de otimização combinatória, onde X é o conjunto das soluções viáveis para I e $c : X \to R$ é uma função que mapeia uma solução ao seu custo. Define-se uma vizinhança(*neighborhood*) de uma solução $x \in X$ como $N(x) \subseteq X$; ou seja, N é uma função que mapeia uma solução a um conjunto de soluções. Assumindo que este seja um problema de minimização, uma solução x é dita localmente ótima em relação a uma vizinhança N se $c(x) \leq c(x') \quad \forall x' \in N(x)$.

Com estas definições podemos definir um algoritmo de *neighborhood search*. O algoritmo toma uma solução inicial x como entrada. Então computa x', a menor solução na vizinhança de x. Se $c(x') < c(x)$, então o algoritmo atualiza o valor de x, $x = x'$. A busca é realizada na vizinhança da nova solução x em busca de uma solução ainda melhor e esse processo se repete até que um ótimo local seja alcançado, quando o algoritmo é encerrado [28].

Very Large-scale Neighborhood Search

Um algoritmo de neighborhood search é considerado como pertencente à classe de algoritmos VLSN por Ahuja et al. [5] se a vizinhança em que ele busca cresce exponencialmente de acordo com o tamanho da instância, ou se a vizinhança é muito grande para ser explicitamente examinada na prática.

Intuitivamente, pode-se imaginar que realizar uma busca em uma vizinhança muito grande deveria levar a soluções de melhor qualidade que buscar em uma vizinhança pequena. Mas, na prática, algoritmos que trabalham com vizinhanças pequenas podem alcançar resultados similares ou superiores se utilizados em uma meta-heurística pois eles geralmente são mais rápidos. Assim, algoritmos VLSN não são universalmente superiores. Porém eles podem prover resultados excelentes nas aplicações adequadas [28].

A *Large Neighborhood Search*

A meta-heurística *large neighborhood search* (LNS) foi proposta por Shaw [31]. Diferentemente da maioria dos algoritmos de busca em vizinhança, que definem a vizinhança explicitamente, na LNS a vizinhança é definida implicitamente por um método de destruição e reparação. O método de destruição destrói parte da solução atual enquanto o método de reparação a reconstrói. O método destrói geralmente contém um elemento de aleatoriedade, de modo que diferentes partes da solução sejam destruídas cada vez que ele é chamado. E a vizinhança de uma solução é definida como o conjunto de soluções que podem ser alcançadas pela aplicação conjunta do método de destruição seguido do de reparação [28].

Para ilustrar estes conceitos, olhemos para o seu funcionamento na resolução do problema de Steiner. Podemos, por exemplo, ter um método de destruição que desfaça parte da solução corrente adicionando à árvore da última solução encontrada arestas pertencentes à instância original do problema. Estas arestas podem ser escolhidas aleatoriamente entre as arestas conectadas aos vértices que fazem parte da solução corrente e o critério de parada para a operação de destruição pode ser a adição de uma certa proporção de arestas em relação ao tamanho da instância. Essa operação transformaria a solução corrente em uma instância reduzida do problema original. Um método de reparo poderia, então, resolver esta instância reduzida, obtendo assim uma solução diferente para a instância original do problema.

O Algoritmo 1 apresenta o pseudo-código da meta-heurística LNS. Ele recebe como entrada uma solução válida x e retorna a melhor solução encontrada durante a sua execução x_{bsf}. Três variáveis são mantidas na memória durante a sua execução: a solução corrente x; a melhor solução encontrada x_{bsf}; e uma solução temporária x^t que pode ser descartada ou promovida a solução corrente. O algoritmo inicia sua execução definindo a solução x recebida como entrada como sendo a melhor solução encontrada. Em seguida é iniciado o laço principal do algoritmo, dentro dele, na linha 4, primeiro se aplica o método de destruição à solução corrente e em seguida é aplicado o método de reparação à saída retornada por `destrói`. O método `repara` retornará, então, uma solução que será temporariamente armazenada em x^t. Na linha 5, a nova solução é avaliada, determinando se ela será aceita como a nova solução corrente ou se deverá ser rejeitada. A função de aceitação pode ser implementada de formas variadas, da maneira que mais se adéque ao problema abordado. Na linha oito é verificado se a nova solução é melhor que a melhor solução armazenada, em caso afirmativo, a melhor solução, x_{bsf}, é atualizada. Por fim, x_{bsf} é retornada.

Como se pode ver, a meta-heurística LNS não busca toda a vizinhança de uma solução, ela simplesmente realiza uma amostragem desta vizinhança. A ideia principal é que uma vizinhança grande permite que a heurística navegue facilmente pelo espaço de soluções,

mesmo se a instância for fortemente restrita. Diferindo de uma *small neighborhood search* que pode tornar a navegação pelo espaço de soluções muito mais difícil [28].

Algoritmo 1 LNS

1: **função** LNS(x)
2: $x_{bsf} := x$
3: **enquanto** critério de parada não for atingido **faça**
4: $x^t :=$ `repara`(`destrói`(x))
5: **se** `aceita` (x^t, x) **então**
6: $x := x^t$
7: **fim se**
8: **se** x^t é melhor que x_{bsf} **então**
9: $x_{bsf} := x^t$
10: **fim se**
11: **fim enquanto**
12: **retorne** x_{bsf}
13: **fim função**

Capítulo 3

Proposta

Este capítulo apresentará as propostas deste trabalho. Inicialmente é apresentada a heurística construtiva adotada na LNS e em seguida é apresentada a forma como a meta-heurística LNS foi implementada.

3.1 Heurística Construtiva - TCPrim

O algoritmo utilizado na fase de reparação da LNS neste trabalho, intitulado `TCPrim`, baseia-se no algoritmo de Prim para árvores geradoras mínimas [29]. Porém, diferentemente da versão original, a versão desenvolvida para o TCP: pode incluir certo grau de aleatoriedade na solução; é capaz de classificar as arestas da instância de acordo com o efeito que elas terão ao ser inseridas na solução em relação ao número de elos, L, e roteadores, R, na árvore gerada; e pode determinar a validade de uma solução quanto ao limite de L e R que ela pode conter. O Algoritmo 2 contém o pseudo-código do `TCPrim` e será detalhado a seguir.

O `TCPrim` recebe como entrada uma instância I; um real α, que determina sua aleatoriedade; um inteiro o, que determina o vértice inicial da solução; um booleano c que determina se as arestas inseridas deverão ou não ser classificadas antes de sua inserção na solução (de acordo com o Algortimo 3 que será detalhado mais adiante); e um booleano d que determina se soluções que estourem o limite de L ou de R devem ser descartadas. A saída é uma árvore que conecta todos os terminais da instância I ou grafo vazio, em caso de solução inválida.

Algoritmo 2 TCPrim

```
1: função TCPrim(I, α, o, c, d)
2:     terminais_restantes := I.terminais
3:     fila := ∅
4:     custos[v] := ∞   ∀v ∈ I
5:     pais[v] := −1   ∀v ∈ I
6:     graus[v] := 0   ∀v ∈ I
7:     naMST[v] := falso   ∀v ∈ I
8:     L := 0
9:     R := 0
10:    fila.inserir({o, 0})
11:    enquanto fila ≠ ∅ faça
12:        se c então
13:            atualizar(graus, L, R)
14:            E := escolher_próximo(fila, α, I.maxL, I.maxR, L, R, pais, graus)
15:        fim se
16:        senão
17:            E := elemento aleatório entre os α∗|fila| primeiros elementos da fila
18:        fim senão
19:        v := fila.extrair(E)
20:        naMST[v] := verdadeiro
21:        se v ∈ I.terminais então
22:            terminais_restantes.remover(v)
23:        fim se
24:        para todo u conectado a v faça
25:            d := custo({u, v})
26:            se naMST[u] = falso  &  custos[u] > d então
27:                fila.inserir(u, d)
28:                custos[u] := d
29:                pais[u] := v
30:            fim se
31:        fim para
32:    fim enquanto
33:    g := construir_grafo(pais, custos)
34:    g.reduzir()
35:    atualizar(graus, L, R)
36:    se [d  &  (L > I.maxL ou R > I.maxR)] ou terminais_restantes ≠ ∅ então
37:        g := ∅
38:    fim se
39:    retorne g
40: fim função
```

Primeiramente, são inicializadas: uma lista contendo os terminais ainda não conectados (terminais_restantes); uma fila de prioridade vazia; uma lista que armazenará os custos de cada conexão; uma lista associando cada vértice ao vértice ao qual ele estará conectado (pais); uma lista que armazenará o grau corrente de cada vértice; e uma lista que armazenará se um vértice já está na árvore de conexão. Além disso a contagem de elos (L) e roteadores (R) é inicializada.

Em seguida o vértice de origem (o) é inserido na fila de prioridade com seu peso definido como 0 e o laço principal do algoritmo é executado até que a fila volte a estar vazia.

Dentro do laço principal, primeiramente, o próximo vértice a ser extraído da fila de prioridade é escolhido. Caso c seja verdadeiro, a lista guardando os graus de cada vértice e a contagem de L e de R são atualizados e E, uma variável armazenando uma posição na fila, é escolhida na linha 14 com o auxílio do Algoritmo 3, que faz uso de α, L máximo, R máximo, L, R, pais e graus e será detalhado adiante. Caso c seja falso, E será uma posição aleatória entre os $\alpha * |\text{fila}|$ primeiros vértices da fila ($|\text{fila}|$ é a cardinalidade da fila).

No passo seguinte, v é definido como o vértice armazenado na posição E dentro da fila e é sinalizado que ele faz parte da árvore de conexão, na linha 20. Caso v seja um terminal, ele é removido da lista de terminais restantes.

Por fim, na linha 24, é iniciado um laço que adiciona os vértices conectados a v na fila de prioridade, caso eles ainda não estejam na árvore de conexão e o custo de cada aresta (u, v), onde u é um vértice conectado a v, seja menor que o custo da conexão entre u e o vértice ao qual ele estava conectado anteriormente. Ou seja, é atualizado o custo de conexão entre a árvore e o vértice u. Caso um vértice u seja adicionado a fila, o custo da aresta conectando a árvore a u é definido como o custo da aresta (u, v), linha 28, e o vértice ao qual u está conectado é definido como v, linha 29.

Ao fim do laço principal, o grafo que deverá ser retornado pelo algoritmo é construído, linha 33, e, em seguida, as folhas não terminais (vértices que não fazem parte da solução do problema de Steiner) são removidas do grafo. O grafo restante é uma árvore que conecta todos os terminais da instância I.

Por fim, g é retornado, porém, antes de a solução ser retornada, é verificada a sua validade. Caso a árvore encontrada não conecte todos os terminais de I (a instância era inválida), ou caso d seja verdadeiro e L ou R extrapolem o limite permitido, g é invalidado. A seguir será detalhado o funcionamento do Algoritmo 3, que escolhe o próximo vértice a ser inserido na árvore de acordo com determinada forma de classificação.

3.1.1 Escolhendo um vértice da fila de prioridade

O Algoritmo 3 recebe como entrada a fila, α, os valores máximos permitidos para L e R na solução, a contagem atual de L e de R, pais e graus. Sua saída é a posição do vetor

escolhido na fila de prioridade.

Inicialmente, αk é definido como α multiplicado pelo tamanho da fila. Em seguida, os αk primeiros elementos da fila, fila$_{\alpha k}$, são classificados em quatro tipos diferentes (t_a, t_b, t_c e t_d) de acordo com o grau do vértice que os conecta à árvore, obtido a partir das listas pais e graus, da seguinte forma: para $v_i \in$ fila$_{\alpha k}$; se o vértice que conecta v_i à árvore de conexão é terminal, então, v_i é classificado como t_a; se o vértice que conecta v_i à árvore de conexão é um roteador (grau maior que dois), então, v_i é classificado como t_b; se o vértice que conecta v_i à árvore de conexão é um elo (grau dois), então, v_i é classificado como t_c; se o vértice que conecta v_i à árvore de conexão é uma folha não terminal, então, v_i é classificado como t_d; e caso nenhuma destas classificações seja adequada, v_i não será classificado. Esta classificação inicializa as listas t_a, t_b, t_c e t_d, que contém as posições de cada vértice em fila$_{\alpha k}$.

Algoritmo 3 Escolher próximo vértice

```
1: função ESCOLHER_PRÓXIMO(fila, α, maxL, maxR, L, R, pais, graus)
2:     αk = α*|fila|
3:     t_a, t_b, t_c, t_d := classificar os αk primeiros elementos da fila
4:     extrair := escolher índice aleatório entre o primeiro e o αk
5:     se |t_a| > 0 então
6:         extrair := elemento aleatório em t_a
7:     fim se
8:     senão,
9:     se |t_b| > 0 então
10:        extrair := elemento aleatório em t_b
11:    fim se
12:    senão,
13:    se |t_c| > 0 & maxR - R < maxL - L então
14:        extrair := elemento aleatório em t_c
15:    fim se
16:    senão,
17:    se |t_d| > 0 & maxL - L < maxR - R então
18:        extrair := elemento aleatório em t_d
19:    fim se
20:    retorne extrair
21: fim função
```

Após a classificação dos vértices, a variável que armazenará a posição do vértice a ser escolhido, extrair, é inicializada como um dos αk primeiros índices da fila, linha quatro. Em seguida, caso necessário, extrair terá seu valor atualizado antes de ser retornada da seguinte maneira: caso hajam elementos do tipo t_a, extrair é atualizada com o valor de

um elemento aleatório pertencente a t_a; caso contrário, caso hajam elementos do tipo t_b, extrair é atualizada com o valor de um elemento aleatório pertencente a t_b; caso contrário, caso hajam elementos do tipo t_c e o limite de roteadores menos a contagem atual de roteadores seja menor que o limite de elos menos a contagem atual de elos, extrair é atualizada com o valor de um elemento aleatório pertencente a t_c; caso contrário, caso hajam elementos do tipo t_d e o limite de elos menos a contagem atual de elos seja menor que o limite de roteadores menos a contagem atual de roteadores, extrair é atualizada com o valor de um elemento aleatório pertencente a t_d. Após isso a variável extrair é retornada.

3.2 *Large Neighborhood Search*

Esta seção apresentará a implementação da meta-heurística utilizada para a solução do Problema de Steiner e do TCP neste trabalho. O Algoritmo 4 contém o pseudo-código da implementação e será detalhado a seguir.

O Algoritmo 4 recebe como entrada a instância, I, do problema e um real positivo y que limita a quantidade de iterações de seu laço principal. Inicialmente, na linha 2, a instância é simplificada pela remoção das folhas não terminais que fazem parte de seu grafo, pois elas não poderiam fazer parte da solução final. Em seguida, as variáveis s, que armazenará a solução final, e n, que armazenará solução corrente, são inicializadas com uma execução do Algoritmo TCPrim, apresentado na Subseção 3.1, os parâmetros passados definem α como zero e não levam em consideração as restrições de elos e de roteadores.

Algoritmo 4 Large Neighborhood Search

```
1: função LNS(I, y)
2:     I.reduzir()
3:     s := TCPrim(I, 0, falso, falso)
4:     n := s
5:     laço_principal(s, n, 5*y, y, falso, I)
6:     se s.L > I.maxL ou s.R > I.maxR então
7:         laço_principal(s, n, 20*y, 5*y, verdadeiro, I)
8:     fim se
9:     se s.L > I.maxL ou s.R > I.maxR então
10:        retorne ∅
11:    fim se
12:    retorne s
13: fim função
```

Na linha 5 é iniciado, pela primeira vez, o laço principal do algoritmo. Ele está definido em uma função separada pois pode vir a ser utilizado na etapa seguinte da LNS. O Algoritmo 5 recebe como parâmetros referências para s (solução final) e para n (solução corrente), além dos reais positivos F1 e F1 que limitarão a quantidade de iterações executadas nas duas diferentes etapas em que o Algoritmo 5 pode se dividir. Os valores destes parâmetros são definidos em relação ao parâmetro y recebido pela LNS. Para a primeira execução do laço principal, F1 vale cinco vezes o valor de y e define a quantidade de iterações sem melhoria que serão executadas antes de o algoritmo passar da sua primeira etapa de execução, mais rápida, para a segunda, que pode obter melhores soluções. O parâmetro F2 vale y na primeira execução do laço principal da LNS e define a quantidade máxima de iterações sem melhoria para a segunda etapa do algoritmo antes de ele ser encerrado. O que diferencia as duas diferentes etapas do algoritmo é o valor do parâmetro c na função TCPrim, que determina se as arestas devem levar em consideração a classificação detalhada na Subseção 3.1.1 ao serem inseridas na solução. O parâmetro LR é um booleano que determinará se a análise da qualidade da solução deverá priorizar o custo da solução ou a quantidade de elos e de roteadores que ela possui. Este é o parâmetro que diferencia a primeira da segunda chamada do Algoritmo 5 na execução da LNS.

Inicialmente, o Algoritmo 5 define algumas variáveis que controlarão sua execução: f determina a fase, ou etapa, em que o laço se encontra; sm armazena o número de execuções seguidas em que não houve melhora no resultado; mc armazena a solução que obteve o menor custo durante a execução do algoritmo e é útil caso o parâmetro LR seja verdadeiro; `e` é definida de acordo com o quão esparso o grafo da instância é e será usada na definição do tamanho da vizinhança a ser destruída; α define o grau de aleatoriedade do `TCPrim`, o seu valor foi determinado de modo a se adaptar as diferentes proporções entre a quantidade de vértices($|I.v|$), arestas($|I.e|$) e terminais($|I.t|$) em uma instância. O valor de α foi definido após uma análise empírica de diferentes valores para diferentes instâncias do problema e busca se aproximar do melhor valor possível para cada instância, de acordo com as suas características específicas. A variável booleana fim serve para marcar o encerramento da execução do laço principal.

Seguindo, entramos no laço em si. Primeiramente, a vizinhança da solução corrente, n, é expandida. Esse passo equivale ao de destruição da LNS. A variável d, definida na linha 10, determina o quão expandida será a vizinhança em relação ao tamanho da instância. O valor de d é definido como um real aleatório entre $e/2$ e $2e$, sendo limitado a 1. A função expandir será detalhada adiante. Na linha 12, a contagem de iterações sem melhoria é aumentada. Em seguida, o fluxo do algoritmo toma dois possíveis caminhos, "laço_parte_A", linha 14, e "laço_parte_B", linha 17, estes trechos de código foram destacados para funções separadas a fim de tornar seu entendimento e sua representação neste trabalho mais simples. Caso LR seja falso, o algoritmo focará no custo das soluções

para determinar qual solução tem melhor qualidade, caso LR seja verdadeiro, o foco será na quantidade de elos e de roteadores da solução. O objetivo desta separação é obter soluções de melhor qualidade (menor custo) caso a instância avaliada tenha os limites de L e R pouco restritos ao mesmo tempo em que torna possível a solução de instâncias mais restritas, mesmo que o resultado final possa acabar apresentando uma qualidade inferior.

Algoritmo 5 Laço principal da LNS

1: **função** LAÇO_PRINCIPAL(s, n, F1, F2, LR, I)
2: $f := 1$
3: $sm := 0$
4: $mc := s$
5: $e := |I.v|$ / $|I.e|$
6:

$$\alpha := \frac{\frac{\sqrt{|I.v|/|I.t|}}{\sqrt{|I.v|}+\sqrt{|I.t|}}}{|I.e|/|I.v|}$$

7: $\alpha := \min(1, \alpha)$
8: fim := falso
9: **enquanto** fim = falso **faça**
10: d = aleatório $\in [e/2, min(1, 2 * e)]$
11: `expandir`(n, I, d)
12: $sm := sm + 1$
13: **se** LR = falso **então**
14: laço_parte_A()
15: **fim se**
16: **senão**
17: laço_parte_B()
18: **fim senão**
19: **fim enquanto**
20: **fim função**

Partindo agora para o Algoritmo 6, temos o procedimento executado caso o laço principal da LNS esteja focando no custo da solução. Primeiramente n, que contém a solução corrente após ser expandida na etapa anterior, é atualizado como o resultado da execução do `TCPrim` sobre o seu estado atual. A depender da etapa em que o algoritmo se encontra o `TCPrim` irá ou não classificar as arestas ao inseri-las na solução. A classificação é feita caso o algoritmo esteja em sua segunda fase. Após isso, é verificado se o custo da solução corrente é menor que o da melhor solução encontrada, s. Caso a solução não tenha melhorado n é definido como s. Caso o custo da solução tenha diminuído ou se

mantido o mesmo, a melhor solução é atualizada com o valor de n, linha 16. Além disso, caso n tenha custo menor que s, o algoritmo volta para a sua primeira fase e a contagem de soluções sem melhoria é zerada. Por último a fase em que o algoritmo se encontra é verificada. Caso o algoritmo esteja há mais de F1 iterações sem melhoria durante a execução da primeira fase, a fase, f, é atualizada para 2 e a contagem de iterações sem melhoria é zerada. Para o caso de o algoritmo estar executando a sua segunda fase, caso ele esteja a F2 iterações sem encontrar uma solução de melhor qualidade, fim é definido como verdadeiro e o laço é encerrado

Com o fim da execução do laço principal, voltamos ao Algoritmo 4, na linha 6 é verificado se a contagem de elos e roteadores viola as restrições da instância, em caso negativo o algoritmo é encerrado, retornando a melhor solução encontrada durante a sua execução, s. Caso contrário, o laço principal voltará a ser executado porém, desta vez, com foco na contagem de elos e roteadores.

Algoritmo 6 Laço principal, A - LNS

```
1: função LAÇO_PARTE_A(s, n, F1, F2, I, sm, f, fim)
2:     se f = 2 então
3:         n := TCPrim(n, α, verdadeiro, verdadeiro)
4:     fim se
5:     senão,
6:         n := TCPrim(n, α, falso, verdadeiro)
7:     fim senão
8:     se n.c > s.c então
9:         n := s
10:    fim se
11:    senão,
12:        se n.c < s.c então
13:            sm := 0
14:            f := 1
15:        fim se
16:        s := n
17:    fim senão
18:    se f = 1 & sm > F1 então
19:        sm := 0
20:        f := 2
21:    fim se
22:    senão
23:    se f = 2 & sm > F2 então
24:        fim := verdadeiro
25:    fim se
26: fim função
```

Agora será explicado como a LNS se comporta caso não tenha conseguido encontrar uma solução durante sua primeira execução do laço principal. Na linha 6 do Algoritmo 4, o Algoritmo 5 é chamado com valores 20*y para o parâmetro F1, 5*y para F2 e verdadeiro para LR. A diferença me relação a sua chamada anterior está a partir da linha 13 do Algoritmo 5, desta vez LR é verdadeiro então o Algoritmo 7 que foca na quantidade de L e R é executado.

O Algoritmo 7 começa atualizando a instância corrente com uma chamada do TCPrim onde c é verdadeiro e d é falso, ou seja, as arestas são classificadas antes da inserção na solução, como descrito na Subseção 3.1.1, e as soluções que infringirem as restrições de L e R não são descartadas. Pois, como o foco do Algoritmo 7 é a contagem de elos e roteadores, ele tenderá encontrar soluções que levem a redução dessa contagem e descartar as soluções inválidas pode dificultar as chances de se chegar a uma solução válida. Em

seguida, na linha 3, é verificado se a solução corrente melhorou ou não em relação a atual. A solução recém encontrada n é considerada de qualidade ao menos equivalente a da melhor solução corrente s se a soma de seus elos e roteadores for menor que a da solução s, ou se o custo da solução for reduzido ao mesmo tempo em que ao menos uma das variáveis L ou R, não teve seu valor incrementado enquanto a outra se manteve abaixo do limite permitido. Caso esta condição se mostre verdadeira, na linha 8, a melhor solução encontrada é atualizada com o valor de n, mas antes disso, na linha 4, é verificado se a solução n tem custo menor que o da solução s, em caso afirmativo a variável mc, que armazena a solução de menor custo encontrada durante a execução do algoritmo, é atualizada com o valor de n e a contagem de iterações sem melhoria é zerada. Isso permite que, ao fim da execução do algoritmo, a solução sm seja retornada pela LNS, caso ela seja válida. Esse passo é importante pois ao fim da execução do laço principal, caso LR seja verdadeiro, s não necessariamente terá o menor custo encontrado durante a execução do algoritmo.

Por fim, é verificado o critério de parada do algoritmo. O Algoritmo 7 não se divide em duas etapas diferentes, mas faz uso dos parâmetros F1 e F2 para limitar seu tempo de execução. Caso ele tenha sido executado por F1 iterações sem melhoria e não tenha encontrado uma solução válida ou tenha sido executado por F2 iterações sem melhoria mas já tenha encontrado uma solução válida, a variável fim é atualizada para verdadeiro, encerrando a execução do laço principal da LNS. Antes disso, porém, é verificado se a solução de menor custo, mc, satisfaz as restrições de L e de R da instância, em caso afirmativo, s é atualizada com o valor de mc.

Após a segunda execução do laço principal da LNS é feita uma nova verificação quanto a validade da solução encontrada. Caso esta solução seja inválida a LNS retornará uma solução vazia, caso contrário, a solução válida de menor custo encontrada execução do algoritmo é retornada.

Algoritmo 7 Laço principal, B - LNS

```
1: função LAÇO_PARTE_B(s, n, F1, F2, I, sm, f, fim)
2:     n = TCPrim(n, α, verdadeiro, falso)
3:     se {[(n.R ≤ I.maxR & n.L ≤ s.L) ou (n.L ≤ I.maxL & n.R ≤ s.R)] & n.c < s.c} ou
   (n.R ≤ s.R & n.L ≤ s.L & n.L + n.R < s.L + s.R) então
4:         se n.c < s.c então
5:             mc = n
6:             sm = 0
7:         fim se
8:         s := n
9:     fim se
10:    senão
11:        n := s
12:    fim senão
13:    se sm > F1 ou (sm > F2 & s.R ≤ I.maxR & s.L ≤ I.maxL) então
14:        se mc.L < I.maxL & mc.R < I.maxR então
15:            s := mc
16:        fim se
17:        fim := verdadeiro
18:    fim se
19: fim função
```

Expandindo a vizinhança

Voltando agora para o Algoritmo 8, responsável pela destruição de parte da solução corrente. Ele atinge o seu objetivo com a adição de arestas à solução corrente, n. O número máximo de arestas a serem adicionadas é definido, na linha 2, pelo produto de d pelo número total de arestas na instância do problema. O grafo contido na solução n é expandido a partir de vértices escolhidos aleatoriamente com a adição a n dos vértices a estes conectados. A cada iteração do laço principal do algoritmo, n é expandido e a lista de vértices vizinhos a ele aumenta. Este processo se repete até que max arestas tenham sido adicionadas a n ou que não hajam mais vértices vizinhos disponíveis para a adição. O resultado é uma versão expandida da solução n, formando uma instância reduzida do problema que poderá resultar em uma solução de melhor qualidade ao ser solucionada novamente pelo `TCPrim`.

Algoritmo 8 Expandir - LNS

```
1: função EXPANDIR(n, I, d)
2:     max := d * |I.e|
3:     inseridas := 0
4:     visto := {falso ∀j ∈ I.v}
5:     visitar := ∅
6:     para v ∈ n.v faça
7:         visto[v] := verdadeiro
8:         visitar.adicionar(v)
9:     fim para
10:    enquanto inseridas < max & |visitar| > 0 faça
11:        u := aleatório ∈ visitar
12:        visitar.remover(u)
13:        inserir := arestas ∈ I & conectadas a u
14:        para e ∈ inserir faça
15:            n.adicionar(e) inseridas := inseridas + 1
16:            para v ∈ e faça
17:                se visto[v] = falso então
18:                    visitar.inserir(v)
19:                    visto[v] = verdadeiro
20:                fim se
21:            fim para
22:        fim para
23:    fim enquanto
24: fim função
```

3.3 Método Exato

Nesta seção é apresentado o modelo matemático desenvolvido e o algoritmo *Branch-and-Cut*.

A formulação usada é baseada em uma das formulações de Lucena & Beasley [23]. O modelo pode ser descrito como:

1. acrescente um vértice artificial v_0 ao grafo G;
2. para cada vértice $i \in V - W$, adicione uma aresta $(0, i)$ de custo zero ao grafo;
3. selecione um vértice terminal $w_1 \in W$ e adicione uma aresta $(0, w_1)$ de custo zero ao grafo;
4. considere $V_0 = V \cup \{v_0\}$, $E_0 = E \cup \{(v_0, i) | i \in V - W \cup \{w_1\}\}$ e P_i o conjunto das arestas de E que possuem i como extremo.

A formulação de Lucena & Beasley [23] é descrita a seguir.

$$\text{minimizar} \quad \sum_{(i,j)\in E_0} c_{ij}x_{ij} \tag{3.1}$$

$$\text{sujeito a:} \quad \sum_{(p,q)\in E_0} x_{pq} = |V_0| - 1 \tag{3.2}$$

$$\sum_{p,q\in T \quad (p,q)\in E_0} x_{pq} \leq |T| - 1 \quad \forall T \subseteq V_0 \tag{3.3}$$

$$x_{v_0 i} + x_{pq} \leq 1 \quad \forall (p,q) \in P_i, \forall i \in V - W \tag{3.4}$$

$$\sum_j x_{ij} - 1 \leq \delta(i) * l_i \quad i \in V - W \tag{3.5}$$

$$\sum_j x_{ij} - 2 \leq \delta(i) * r_i \quad i \in V - W \tag{3.6}$$

$$\sum_i r_i \leq R \tag{3.7}$$

$$\sum_i l_i - \sum_i r_i \leq L \tag{3.8}$$

$$x_{ij} \in \{0,1\} \quad (i,j) \in E_0 \tag{3.9}$$

$$l_i \in \{0,1\} \quad i \in V - W \tag{3.10}$$

$$r_i \in \{0,1\} \quad i \in V - W. \tag{3.11}$$

Neste modelo, x_{ij} é uma variável binária que vale 1 se e somente se a aresta (i,j) pertence à solução, l_i é uma variável binária que vale 1 se e somente se duas ou mais arestas são adjacentes ao vértice i na solução e r_i é uma variável binária que vale 1 se e somente se três ou mais arestas são adjacentes ao vértice i na solução. A constante c_{ij} indica o peso da aresta (i,j) e $\delta(i)$ representa o grau do vértice i. A função objetivo (3.1) contabiliza o custo da solução. A restrição (3.2) determina que $|V_0| - 1$ arestas devem pertencer à solução, note que algumas dessas serão arestas artificiais. As restrições (3.3) são eliminações de subciclos, para todo subconjunto T de vértices existe uma restrição que impede que T seja um ciclo, para isso, a quantidade de arestas em T é limitada a $|T| - 1$. Como o número de restrições de subciclo é exponencial, essas restrições serão adicionadas sob demanda pelo algoritmo *Branch-and-Cut* (mais detalhes serão apresentados a seguir). As restrições (3.4) indicam que, para todo vértice i não terminal, se a aresta (v_0, i) pertence a solução, então nenhuma outra aresta adjacente a i poderá pertencer à solução, em outras palavras, i não será um vértice de Steiner. As restrições (3.5) ativam a variável l_i caso a quantidade de arestas adjacentes a i na solução seja maior que dois. Analogamente, as restrições (3.6) ativam a variável r_i caso a quantidade de arestas adjacentes a i na solução seja maior que três. Note que, se o vértice i é um roteador, então $r_i = 1$ e $l_i = 1$. As restrições (3.7) e (3.8) limitam a quantidade de vértices de Steiner, elos e roteadores, respectivamente. Por fim , as restrições (3.9), (3.10) e (3.11) indicam o domínio das variáveis x, y e r, respectivamente. As adaptações ao modelo de Lucena & Beasley [23] correspondem às restrições

Capítulo 4

Resultados

Neste capítulo os resultados computacionais referentes aos testes com o TCP e com o Problema de Steiner serão apresentados.

4.1 Instâncias de Teste

As instâncias de teste utilizadas neste trabalho são as mesmas utilizadas por Martins et al. [24] para avaliar a *Parallel Hybrid GRASP* (PH-GRASP) como solução para o problema de Steiner, retiradas da OR-Library [6]. Adicionalmente, as instâncias de classe B pertencentes aquele conjunto foram utilizadas na avaliação da LNS e da solução exata para o TCP. Para os testes no TCP foram criadas instâncias de diferentes níveis de dificuldade a partir das instâncias do problema de Steiner. As instâncias criadas estão disponíveis em um repositório no GitHub [22].

As instâncias originais se dividem em quatro classes diferentes: B, C, D e E. Cada uma com vinte instâncias diferentes. Estas instâncias possuem as seguintes características:

- Classe B: 18 grafos de com número de vértices entre 50 e 100, número de arestas entre 63 e 200 e número de terminais entre 9 e 50;
- Classe C: 20 grafos de 500 vértices, número de arestas entre 625 e 12.500 e número de terminais entre 5 e 250;
- Classe D: 20 grafos de 1.000 vértices, número de arestas entre 1.250 e 25.000 e número de terminais entre 5 e 500;
- Classe E: 20 grafos de 2.500 vértices, com número de arestas entre 3.125 e 62.500 e número de terminais entre 5 e 1.250.

A partir destas instâncias foram criadas 118 instâncias para o TCP, divididas entre os seguintes níveis, em ordem crescente, de dificuldade: E, 18 instâncias; D, 20 instâncias; C, 20 instâncias; B, 20 instâncias; A, 20 instâncias; e S, 20 instâncias. O processo de

criação destas instâncias consistiu na simples adição das restrições de vértices de Steiner nas soluções. E se deu como descrito a seguir.

Uma versão alterada da implementação da LNS apresentada na Seção 3.1 cujo objetivo é minimizar a soma de elos e roteadores na solução foi utilizada para determinar os valores de l e de r das novas instâncias. A dificuldade de cada grupo de instâncias é determinada pela quantidade de iterações definida para a LNS e pela classe a qual a instância pertencia anteriormente, da seguinte forma:

- Classe E: Criadas a partir das instâncias da classe B da OR-Library, o algoritmo foi executado por 120 iterações.
- Classe D: Criadas a partir das instâncias da classe C da OR-Library, o algoritmo foi executado por 50 iterações.
- Classe C: Criadas a partir das instâncias da classe D da OR-Library, o algoritmo foi executado por 50 iterações.
- Classe B: Criadas a partir das instâncias da classe D da OR-Library, o algoritmo foi executado por 500 iterações.
- Classe A: Criadas a partir das instâncias da classe E da OR-Library, o algoritmo foi executado por 2.000 iterações.
- Classe S: Criadas a partir das instâncias da classe E da OR-Library, o algoritmo foi executado por 10.000 iterações.

4.2 Ferramentas

A meta-heurística LNS foi desenvolvida em C++ 17 e compilada com o Visual C++ 2019. O algoritmo exato foi implementado utilizando a biblioteca de otimização IBM ILOG CPLEX 12.9, com *timeout* de 3600 segundos. Ambos foram executados em um computador com processador Intel Pentium G4560 a 3.5 GHz e 8GB de memória em *single channel* a 2400MHz. A meta-heurística foi executada utilizando o Windows 10 Pro versão 1903 como sistema operacional, já a solução exata foi executada utilizando a distribuição Linux Manjaro, versão 18.0.4.

4.3 Resultados para o Problema de Steiner

Nesta seção serão apresentados os resultados obtidos para o problema de Steiner. Todos os testes foram executados dez vezes, o tempo de execução é a média dos tempos obtidos. Os resultados obtidos pela implementação da LNS, descrita na Seção 3.1, são

comparados aos obtidos por Martins et al. [24]. Para as instâncias da classe C o tempo de execução também é comparado ao da solução exata obtida pelo CPLEX, as outras classes não foram avaliadas pois o uso da solução exata se mostrou inviável para elas. O critério de parada utilizado na LNS foi o número de iterações y. Diferentes valores de y foram testados a fim de avaliar a relação entre tempo de execução e qualidade da solução e determinar qual valor apresenta um bom balanço entre tempo e qualidade.

4.3.1 Resultados da LNS

As Tabelas 4.1, 4.2, 4.3 e 4.4 apresentam os resultados para a LNS com y=2000, este foi o valor de y que apresentou o melhor balanço entre tempo de execução e qualidade do resultado. As tabelas contém os seguintes dados: I, a instância avaliada; *opt*, o valor da solução ótima; min, o custo da melhor solução encontrada; max, o custo da pior solução encontrada; média, o valor médio dos custos das soluções encontradas; mediana, a mediana dos custos das soluções encontradas; t, o tempo total médio de cada execução; t_best, o tempo médio que cada solução levou para encontrar o resultado da execução. Além disso, para cada valor foi calculado o gap que representa a distância relativa entre o respectivo custo para o custo da solução ótima.

O gap entre o custo x e o valor opt é definido como:

$$\text{gap} = \frac{\text{custo} - \text{opt}}{\text{opt}}$$

.

Tabela 4.1: LNS - Resultados para as instâncias da classe C da OR-Library. LNS teve o parâmetro y definido como 2000. O tempo (t e t_best) é dado em segundos e o gap representa a percentagem da taxa de erro.

I	opt	min	gap	max	gap	média	gap	mediana	gap	t	t_best
C1	85	85	0,00	85	0,00	85	0,00	85	0,00	3,65	0,02
C2	144	145	0,69	150	4,17	147,50	2,43	146	1,39	5,72	2,23
C3	754	768	1,86	779	3,32	772,20	2,41	771,50	2,32	10,70	6,46
C4	1.079	1.085	0,56	1.097	1,67	1.090,70	1,08	1.091	1,11	11,06	6,39
C5	1.579	1.597	1,14	1.601	1,39	1.599,60	1,30	1.600	1,33	15,70	10,14
C6	55	55	0,00	55	0,00	55	0,00	55	0,00	4,86	0,13
C7	102	103	0,98	106	3,92	103,60	1,57	103	0,98	8,22	3,51
C8	509	525	3,14	534	4,91	529,70	4,07	529,50	4,03	12,00	6,43
C9	707	721	1,98	728	2,97	723,60	2,35	723,50	2,33	14,98	8,97
C10	1.093	1.100	0,64	1.103	0,91	1.101,40	0,77	1.101,50	0,78	12,28	5,71
C11	32	32	0,00	32	0,00	32	0,00	32	0,00	6,21	0,56
C12	46	46	0,00	46	0,00	46	0,00	46	0,00	6,54	0,07
C13	258	258	0,00	264	2,33	259,90	0,74	260	0,78	11,14	5,21
C14	323	328	1,55	331	2,48	329	1,86	329	1,86	13,84	7,33
C15	556	565	1,62	569	2,34	567,30	2,03	567,50	2,07	24,11	16,95
C16	11	11	0,00	11	0,00	11	0,00	11	0,00	10,54	0,20
C17	18	18	0,00	18	0,00	18	0,00	18	0,00	9,27	0,26
C18	113	113	0,00	114	0,88	113,30	0,27	113	0,00	9,44	2,70
C19	146	146	0,00	149	2,05	147,10	0,75	147	0,68	10,00	3,41
C20	267	267	0,00	267	0,00	267	0,00	267	0,00	8,75	1,09
Média			0,71		1,67		1,08		0,98	10,45	4,39

Tabela 4.2: LNS - Resultados para as instâncias da classe D da OR-Library. LNS teve o parâmetro y definido como 2000. O tempo (t e t_best) é dado em segundos e o gap representa a percentagem da taxa de erro.

I	opt	min	gap	max	gap	média	gap	mediana	gap	t	t_best
D1	106	106	0,00	107	0,94	106,80	0,75	107	0,94	9,30	1,16
D2	220	220	0,00	224	1,82	221,20	0,55	220	0,00	10,72	2,09
D3	1.565	1.619	3,45	1.660	6,07	1.630,80	4,20	1.625,50	3,87	22,33	12,00
D4	1.935	1.985	2,58	2.006	3,67	1.998,20	3,27	2.000	3,36	25,16	13,95
D5	3.250	3.294	1,35	3.316	2,03	3.308,10	1,79	3.309	1,82	29,59	15,98
D6	67	67	0,00	70	4,48	69,40	3,58	70	4,48	12,85	1,12
D7	103	103	0,00	109	5,83	107,20	4,08	108	4,85	17,90	6,27
D8	1.072	1.111	3,64	1.126	5,04	1.118,40	4,33	1.117,50	4,24	33,26	19,13
D9	1.448	1.481	2,28	1.520	4,97	1.492,20	3,05	1.490	2,90	35,37	20,54
D10	2.110	2.142	1,52	2.159	2,32	2.147,90	1,80	2.146	1,71	36,25	19,65
D11	29	29	0,00	29	0,00	29	0,00	29	0,00	16,51	2,98
D12	42	42	0,00	42	0,00	42	0,00	42	0,00	13,21	0,17
D13	500	503	0,60	507	1,40	505,80	1,16	506	1,20	30,21	15,86
D14	667	673	0,90	686	2,85	680,80	2,07	681	2,10	33,61	18,31
D15	1.116	1.130	1,25	1.136	1,79	1.133	1,52	1.133	1,52	36,19	18,26
D16	13	13	0,00	13	0,00	13	0,00	13	0,00	23,38	0,17
D17	23	23	0,00	23	0,00	23	0,00	23	0,00	18,96	0,81
D18	223	225	0,90	230	3,14	227,10	1,84	227	1,79	25,74	10,97
D19	310	314	1,29	318	2,58	316,30	2,03	316,50	2,10	31,15	15,63
D20	537	537	0,00	540	0,56	538,50	0,28	538,50	0,28	29,11	11,30
Média			0,99		2,47		1,81		1,86	24,54	10,32

Como pode ser visto nas Tabelas 4.1, 4.2, 4.3 e 4.4, para o problema de Steiner, a LNS apresentou taxa de erro médio de 1,26% as mínimas dos dez testes realizados, além disso, nas instâncias avaliadas o pior resultado encontrado teve um gap de 8,41% para a instância E2, também é possível observar que a LNS foi capaz de encontrar a solução ótima em vinte e cinco das sessenta instâncias analisadas. Como se pode ver na Tabela 4.4, tanto a taxa de erro quanto o tempo de execução aumentam quanto maiores forem as instâncias.

A Tabela 4.5 apresenta as médias de tempo execução e da melhor taxa de erro para as mesmas instâncias com diferentes valores de y. Como se pode observar aumentar o valor de y para mais de 2000 traz ganhos cada vez menos expressivos, tornando estes valores de y injustificáveis frente ao aumento no tempo de execução. Também é possível notar que mesmo um valor de y igual a 250 é capaz de obter soluções com uma taxa de erro média abaixo dos 2% nas instâncias avaliadas, ao mesmo tempo em que apresenta um tempo de execução 7,4 vezes menor.

Tabela 4.3: LNS - Resultados para as instâncias da classe E da OR-Library. $y = 2000$. O tempo (t e t_best) é dado em segundos e o gap é expresso em %.

I	opt	min	gap	max	gap	média	gap	mediana	gap	t	t_best
E1	111	111	0,00	111	0,00	111	0,00	111	0,00	35,81	0,09
E2	214	220	2,80	232	8,41	226,80	5,98	227,5	6,31	53,76	18,05
E3	4.013	4.190	4,41	4.239	5,63	4.214,40	5,02	4213,5	5,00	97,20	48,50
E4	5.101	5.279	3,49	5.336	4,61	5.312,20	4,14	5313,5	4,17	104,84	50,51
E5	8.128	8.244	1,43	8.278	1,85	8.262,50	1,65	8260,5	1,63	133,28	56,11
E6	73	73	0,00	73	0,00	73	0,00	73	0,00	55,28	0,41
E7	145	147	1,38	151	4,14	149	2,76	149	2,76	60,17	7,57
E8	2.640	2.780	5,30	2.814	6,59	2.798,60	6,01	2800,5	6,08	103,64	38,41
E9	3.604	3.785	5,02	3.843	6,63	3.812,10	5,77	3804	5,55	133,57	61,57
E10	5.600	5.751	2,70	5.784	3,29	5.768,20	3,00	5768	3,00	157,24	59,60
E11	34	34	0,00	34	0,00	34	0,00	34	0,00	60,32	0,27
E12	67	67	0,00	69	2,99	67,70	1,04	68	1,49	58,20	1,71
E13	1.280	1.330	3,91	1.346	5,16	1.336,80	4,44	1335,5	4,34	132,58	63,03
E14	1.732	1.786	3,12	1.830	5,66	1.803,40	4,12	1799,5	3,90	146,21	70,86
E15	2.784	2.841	2,05	2.870	3,09	2.855,20	2,56	2855,5	2,57	184,61	79,23
E16	15	15	0,00	16	6,67	15,30	2,00	15	0,00	89,94	11,65
E17	25	25	0,00	26	4,00	25,10	0,40	25	0,00	66,48	2,70
E18	564	586	3,90	595	5,50	591,10	4,80	591,5	4,88	128,47	62,39
E19	758	771	1,72	781	3,03	776,90	2,49	778	2,64	165,56	92,95
E20	1342	1.345	0,22	1.355	0,97	1.349,70	0,57	1348,5	0,48	235,44	138,73
Média			2,07		3,91		2,84		2,74	110,13	43,22

Tabela 4.4: LNS - Resumo dos resultados para as instâncias da OR-Library. y = definido como 2000. O tempo (t e t_best) é dado em segundos e o gap é expresso em %.

Classe	gap média	gap mínimo	gap máximo	gap mediana	t	t_best
C	1,08	0,71	1,67	0,98	10,45	4,39
D	1,81	0,99	2,47	1,86	24,54	10,32
E	2,84	2,07	3,91	2,74	110,13	43,22
Geral	1,91	1,26	2,68	1,87	48,37	19,31

Tabela 4.5: LNS - Comparação de diferentes valores para y. O gap é expresso em % e o tempo em segundos.

	Classe C		Classe D		Classe E		Geral	
y	gap	t	gap	t	gap	t	gap	t
16	2,32	0,13	3,21	0,28	5,33	1,06	3,62	0,49
250	1,06	1,47	1,65	3,38	3,18	15,04	1,96	6,63
500	0,93	2,89	1,81	6,42	2,67	28,75	1,80	12,69
1.000	0,93	5,45	1,44	12,71	2,30	58,10	1,56	25,42
2.000	0,71	10,45	0,99	24,54	2,07	110,13	1,26	48,37
8.000	0,48	39,32	0,87	93,40	1,89	398,81	1,08	177,18

4.3.2 Comparação dos resultados

A Tabela 4.6 compara os resultados obtidos pela meta-heurística apresentada na Seção 3.1, com y definido como 250 e 2000, com aqueles encontrados pela solução exata apresentada na seção 3.3 e os resultados encontrados por Martins et al. [24] na meta-heurística *Parallel Hybrid* GRASP apresentada por eles. Não foi possível testar a implementação utilizada por Martins et al. [24] na máquina utilizada para os outros testes, os autores não mais a possuem. Porém os resultados para o tempo de execução obtidos por Martins et al. [24] foram mantidos. Para as instâncias das classes D e E, cujos resultados são comparados nas Tabelas 4.7 e 4.8 , não foram incluídos os resultados para a solução exata pois ela não foi capaz de solucionar estas instâncias dentro do tempo limite de 3600 segundos. A Tabela 4.9 mostra estes dados de forma resumida e inclui também os resultados obtidos com y igual a 250.

Tabela 4.6: Comparação entre LNS, CPLEX e PH-GRASP. Resultados encontrados para as instâncias da classe C. O tempo (t) é dado em segundos e o gap representa a percentagem da taxa de erro. Tempos maior que 3600 indicam que o CPLEX não provou a otimalidade no tempo estipulado.

		CPLEX	LNS – y=250			LNS – y=2.000			PH-GRASP		
I	opt	t	custo	gap	t	custo	gap	t	custo	gap	t
C1	85	64.82	85	0,00	0,47	85	0,00	3,65	85	0,00	0,81
C2	144	>3.600	146	1,39	0,90	145	0,69	5,72	144	0,00	1,42
C3	754	>3.600	773	2,52	1,57	768	1,86	10,7	754	0,00	4,83
C4	1.079	>3.600	1089	0,93	1,57	1.085	0,56	11,06	1.079	0,00	3,39
C5	1.579	>3.600	1601	1,39	2,04	1.597	1,14	15,7	1.579	0,00	0,73
C6	55	216,04	55	0,00	0,75	55	0,00	4,86	55	0,00	1,37
C7	102	9,19	103	0,98	0,81	103	0,98	8,22	102	0,00	6,36
C8	509	>3.600	534	4,91	1,60	525	3,14	12	509	0,00	6,37
C9	707	>3.600	727	2,83	1,84	721	1,98	14,98	707	0,00	101,00
C10	1.093	>3.600	1.101	0,73	1,59	1.100	0,64	12,28	1.093	0,00	0,87
C11	**32**	**>3.600**	**32**	**0,00**	**0,85**	**32**	**0,00**	**6,21**	**33**	**3,13**	**1,58**
C12	46	>3.600	46	0,00	0,88	46	0,00	6,54	46	0,00	4,67
C13	258	>3.600	259	0,39	1,58	258	0,00	11,14	258	0,00	98,14
C14	323	>3.600	330	2,17	2,14	328	1,55	13,84	323	0,00	65,58
C15	556	>3.600	569	2,34	2,58	565	1,62	24,11	556	0,00	6,91
C16	11	415,77	11	0,00	1,61	11	0,00	10,54	11	0,00	2,72
C17	18	55,91	18	0,00	1,39	18	0,00	9,27	18	0,00	4,14
C18	**113**	**>3.600**	**113**	**0,00**	**1,63**	**113**	**0,00**	**9,44**	**115**	**1,77**	**155,64**
C19	**146**	**>3.600**	**147**	**0,68**	**1,77**	**146**	**0,00**	**10,97**	**148**	**1,37**	**128,14**
C20	267	>3.600	267	0,00	1,86	267	0,00	8,75	267	0,00	2,53
Média		2.584,56		1,06	1,47		0,71	10,45		0,31	29,86

Tabela 4.7: Comparação entre LNS e PH-GRASP. Resultados encontrados para as instâncias da classe D. O tempo (t) é dado em segundos e o gap representa a percentagem da taxa de erro.

		LNS – y=250			LNS – y=2.000			PH-GRASP		
I	opt	custo	gap	t	custo	gap	t	custo	gap	t
D1	106	106	0,00	1,13	106	0,00	9,3	106	0,00	1,18
D2	220	220	0,00	1,17	220	0,00	10,72	220	0,00	3,88
D3	1.565	1.634	4,41	2,93	1.619	3,45	22,33	1.565	0,00	21,77
D4	1.935	2.003	3,51	3,46	1.985	2,58	25,16	1.935	0,00	2,40
D5	3.250	3.314	1,97	2,76	3.294	1,35	29,59	3.250	0,00	1,15
D6	67	67	0,00	1,61	67	0,00	12,85	67	0,00	2,97
D7	103	105	1,94	2,27	103	0,00	17,9-	103	0,00	3,46
D8	1.072	1.125	4,94	3,45	1.111	3,64	33,26	1.073	0,09	365,94
D9	1.448	1.497	3,38	3,62	1.481	2,28	35,37	1.448	0,00	394,46
D10	2.110	2.151	1,94	4,74	2.142	1,52	36,25	2.110	0,00	15,12
D11	29	29	0,00	1,89	29	0,00	16,51	29	0,00	6,83
D12	42	42	0,00	1,87	42	0,00	13,21	42	0,00	28,07
D13	500	508	1,60	3,99	503	0,60	30,21	502	0,40	553,29
D14	667	678	1,65	5,27	673	0,90	33,61	667	0,00	324,72
D15	1.116	1.133	1,52	4,55	1.130	1,25	36,19	1.116	0,00	10,24
D16	13	13	0,00	3,50	13	0,00	23,38	13	0,00	27,03
D17	23	23	0,00	2,63	23	0,00	18,96	23	0,00	15,55
D18	**223**	**230**	**3,14**	**4,52**	**225**	**0,90**	**25,74**	**229**	**2,69**	**655,37**
D19	**310**	**318**	**2,58**	**5,89**	**314**	**1,29**	**31,15**	**316**	**1,93**	**582,04**
D20	**537**	**538**	**0,19**	**6,31**	**537**	**0,00**	**29,11**	**538**	**0,19**	**66,97**
Média			1,64	3,38		0,99	24,54		0,27	154,12

Tabela 4.8: Comparação entre LNS e PH-GRASP. Resultados encontrados para as instâncias da classe E. O tempo (t) é dado em segundos e o gap representa a percentagem da taxa de erro.

		LNS – y=250			LNS – y=2.000			PH-GRASP		
I	opt	custo	gap	t	custo	gap	t	custo	gap	t
E1	111	111	0,00	4,53	111	0,00	35,81	111	0,00	1,42
E2	214	231	7,94	6,16	220	2,80	53,76	214	0,00	5,99
E3	4.013	4.234	5,51	10,53	4.190	4,41	97,20	4.016	0,07	107,16
E4	5.101	5.322	4,33	11,55	5.279	3,49	104,84	5.101	0,00	21,02
E5	8.128	8.269	1,73	16,63	8.244	1,43	133,28	8.128	0,00	0,67
E6	73	73	0,00	7,32	73	0,00	55,28	73	0,00	8,48
E7	145	150	3,45	7,57	147	1,38	60,17	145	0,00	37,94
E8	2.640	2.805	6,25	13,02	2.780	5,30	103,64	2.648	0,30	6.091,43
E9	3.604	3.845	6,69	15,33	3.785	5,02	133,57	3.608	0,11	2.948,36
E10	5.600	5.789	3,37	19,81	5.751	2,70	157,24	5.600	0,00	188,58
E11	34	34	0,00	7,90	34	0,00	60,32	34	0,00	26,62
E12	67	67	0,00	8,25	67	0,00	58,20	67	0,00	75,85
E13	1.280	1.351	5,55	16,34	1.330	3,91	132,58	1.292	0,93	5.644,48
E14	1.732	1.807	4,33	21,46	1.786	3,12	146,21	1.735	0,17	3.233,90
E15	2.784	2.861	2,77	24,69	2.841	2,05	184,61	2.784	0,00	219,97
E16	15	15	0,00	11,36	15	0,00	89,94	15	0,00	134,64
E17	25	25	0,00	9,86	25	0,00	66,48	25	0,00	288,90
E18	564	599	6,21	22,13	586	3,90	128,47	584	3,55	4.712,09
E19	758	789	4,09	26,85	771	1,72	165,56	770	1,58	3.248,99
E20	1.342	1.362	1,49	39,50	1.345	0,22	235,44	1.343	0,07	601,41
Média			3,19	15,04		2,07	110,13		0,34	1.379,89

Tabela 4.9: Comparação entre LNS e PH-GRASP. O tempo (t) é dado em segundos e o gap representa a percentagem da taxa de erro.

	LNS – y=250		LNS – y=2.000		PH-GRASP	
Classe	gap	t	gap	t	gap	t
C	1,06	1,47	0,71	10,45	0,31	29,86
D	1,64	3,38	0,99	24,54	0,27	154,12
E	3,19	15,04	2,07	110,13	0,34	1.379,89
Geral	1,96	6,63	1,26	48,37	0,31	521,29

Como se pode observar, a solução apresentada por Martins et al. [24] se mostrou superior em qualidade de solução à implementação da LNS apresentada neste trabalho nas instâncias avaliadas. Entretanto, a solução aqui apresentada conseguiu em média uma taxa de erro abaixo dos 2% para valores de y maiores que 250 ao mesmo tempo em que teve um tempo de execução muito menor e que depende apenas do tamanho da instância avaliada, diferentemente do PH-GRASP cujo tempo de execução pode diferir bastante entre duas instâncias de mesmo tamanho. Além disso ele encontrou soluções

algumas ótimas que a *Parallel Hybrid* GRASP não foi capaz de encontrar e encontrou resultados melhores que os do PH-GRASP em outras, mostrando-se uma alternativa viável para os casos em que o tempo de execução seja mais importante que a qualidade da solução. Também vale ser notada a sua superioridade em termos de uso de recursos quando comparada a solução exata implementada com o auxílio do CPLEX, enquanto o CPLEX foi incapaz de solucionar as instâncias das classes D, E e algumas instâncias da classe C no tempo limite a LNS teve como maior tempo de execução os 235.44 segundos necessários para solucionar a instância E20 com valor de y igual a 2.000. Além disso, seu uso de memória ficou abaixo dos 12MB em todas as instâncias, enquanto a solução exata não foi capaz de executar boa parte das instâncias da classe D e E utilizando os 8GB de memória disponíveis na máquina de testes. Apesar de o PH-GRASP de Martins et al. [24] ter obtido resultados melhores que a LNS no problema de Steiner, optou-se por não adaptá-lo ao TCP pois ele usa uma heurística de busca local que deve ser adaptada devido às restrições do número de vértices de Steiner, o que o torna a LNS mais adequada a este problema.

4.4 Resultados para o TCP

Nesta seção serão apresentados os resultados obtidos para o Problema de Conexão de Terminais com Número Restrito de Roteadores e Elos. Todos os testes foram executados dez vezes, o tempo de execução é a média dos tempos obtidos. Das instâncias criadas apenas as da classe E foram executadas com o modelo do CPLEX e possuem solução ótima conhecida. O modelo não foi capaz de encontrar a solução exata para as outras instâncias, sendo assim, o melhor resultado conhecido para elas é o valor mínimo encontrado pela implementação da LNS apresentada neste trabalho. O parâmetro y foi definido como 500 em todos os testes da LNS.

4.4.1 Resultados

As Tabelas 4.10, 4.11, 4.12, 4.13, 4.14 e 4.15 apresentam os resultados para a LNS com y=500. As tabelas contém os seguintes dados: I, a instância avaliada; *opt*, o valor da solução ótima; min, a melhor solução encontrada; max, a pior solução encontrada; média, o valor médio das soluções encontradas; mediana, a mediana das soluções encontradas; t, o tempo total médio de cada execução; t_best, o tempo médio que cada solução levou para encontrar o resultado; e sol a quantidade de vezes em que a LNS conseguiu encontrar uma solução dentre as dez execuções. Além disso, gap representa a distância entre as soluções encontradas e a solução ótima. Na Tabela 4.10 foram incluídos os resultados obtidos pela solução exata apresentada na Seção 3.3, nas outras tabelas a melhor solução conhecida foi a encontrada pela LNS, sendo ela maior ou igual a solução ótima para estas instâncias.

Para os casos em que a LNS não foi capaz de encontrar uma solução em nenhuma das execuções, as métricas de performance foram calculadas sobre os resultados obtidos pelas execuções que obtiveram sucesso. As instâncias em que nenhuma execução da LNS foi capaz de encontrar uma solução têm como valor para min aquele encontrado ao criar a instância.

Tabela 4.10: TCP - Resultados para as instâncias da classe E. a LNS teve o parâmetro y definido como 500. O tempo (t e best) é dado em segundos e o gap representa a percentagem da taxa de erro. sol é a quantidade de vezes em que a LNS foi capaz de encontrar uma solução entre os dez testes realizados e med. é a mediana.

	CPLEX		LNS										
I	opt	t	min	gap	max	gap	média	gap	med.	gap	t	best	sol
E1	83	0,19	83	0,00	83	0,00	83	0,00	83	0,00	0,23	0,12	10
E2	83	0,42	83	0,00	83	0,00	83	0,00	83	0,00	0,14	0,02	10
E3	138	0,24	138	0,00	138	0,00	138	0,00	138	0,00	0,24	0,11	10
E4	62	0,78	62	0,00	63	1,61	62,60	0,97	63	1,61	0,21	0,13	10
E5	62	22,40	62	0,00	62	0,00	62	0,00	62	0,00	0,36	0,23	10
E6	124	0,41	126	1,61	133	7,26	129,60	4,52	129	4,03	0,38	0,31	10
E7	111	0,51	111	0,00	112	0,90	111,50	0,45	111,50	0,45	0,24	0,13	10
E8	104	1,58	107	2,88	107	2,88	107	2,88	107	2,88	0,18	0,00	10
E9	222	0,90	222	0,00	222	0,00	222	0,00	222	0,00	0,43	0,21	10
E10	90	70,80	90	0,00	92	2,22	91,60	1,78	92	2,22	0,47	0,29	10
E11	88	2,42	88	0,00	90	2,27	88,20	0,23	88	0,00	0,49	0,39	10
E12	174	0,68	174	0,00	178	2,30	174,40	0,23	174	0,00	0,50	0,37	10
E13	165	7,42	169	2,42	174	5,45	171	3,64	169	2,42	0,63	0,40	10
E14	235	14,30	239	1,70	240	2,13	239,20	1,79	239	1,70	0,50	0,31	10
E15	318	2,82	324	1,89	324	1,89	324	1,89	324	1,89	0,67	0,35	10
E16	127	18,10	127	0,00	161	26,77	135	6,30	132,5	4,33	0,77	0,72	10
E17	135	25,70	135	0,00	137	1,48	135,80	0,59	136	0,74	0,73	0,66	10
E18	219	4,80	222	1,37	232	5,94	224,70	2,60	223	1,83	0,73	0,72	10
Geral		9,69		0,66		3,51		1,55		1,34	0,44	0,30	100%

Tabela 4.11: TCP - Resultados para as instâncias da classe D. a LNS teve o parâmetro y definido como 500. O tempo (t e t_best) é dado em segundos e o gap representa a percentagem da taxa de erro. sol é a quantidade de vezes em que a LNS foi capaz de encontrar uma solução entre os dez testes realizados.

	TCP: LNS - y=500						
I	**min**	max	média	mediana	t	t_best	sol
D1	**87**	87	87	87	0,89	0,05	10
D2	**153**	160	154,80	153,50	3,14	2,54	10
D3	**765**	787	777,40	777,50	4,90	2,22	10
D4	**1.084**	1.104	1.091,20	1.090	5,69	3,60	10
D5	**1.593**	1.603	1.597,60	1.598	7,47	4,48	10
D6	**55**	55	55	55	1,47	0,32	10
D7	**103**	107	104,80	104	3,69	2,35	10
D8	**523**	545	535	534,50	7,69	5,02	10
D9	**724**	743	730,40	730,50	8,07	5,06	10
D10	**1.105**	1.117	1.109,20	1.108,50	7,90	4,57	10
D11	**32**	35	32,50	32	2,38	1,05	10
D12	**46**	50	47	47	5,04	2,68	10
D13	**260**	268	263,20	263	8,46	5,63	10
D14	**327**	333	329	329	7,17	4,15	10
D15	**557**	568	562,70	563	10,39	6,91	10
D16	**13**	17	14,30	14	3,91	3,41	10
D17	**20**	28	22,20	22	4,57	3,70	10
D18	**115**	120	117,10	116,50	7,59	4,87	10
D19	**150**	157	153,20	153	7,09	4,23	10
D20	**268**	275	271	271	8,03	4,68	10
Geral					5,78	3,58	100%

Tabela 4.12: TCP - Resultados para as instâncias da classe C. a LNS teve o parâmetro y definido como 500. O tempo (t e t_best) é dado em segundos e o gap representa a percentagem da taxa de erro. sol é a quantidade de vezes em que a LNS foi capaz de encontrar uma solução entre os dez testes realizados.

	TCP: LNS - y=500						
I	**min**	max	média	mediana	t	t_best	sol
C1	**107**	113	107,60	107	6,09	2,71	10
C2	**220**	224	222	222	2,29	0,26	10
C3	**1.622**	1.702	1.651,20	1.645,50	20,50	11,92	10
C4	**1.996**	2.016	2.006,90	2.008,50	25,07	14,77	10
C5	**3.295**	3.328	3.314,60	3.314,50	27,62	17,55	10
C6	**72**	86	76,60	73,50	11,13	8,37	10
C7	**103**	110	105,60	105	18,28	11,37	10
C8	**1.137**	1.174	1.154,90	1.154,50	26,16	14,67	10
C9	**1.516**	1.532	1.523,70	1.523	27,73	16,22	10
C10	**2.155**	2.169	2.162,10	2.160,50	27,18	15,21	10
C11	**34**	46	37,90	36,50	15,47	14,14	10
C12	**43**	47	44,20	43	11,90	9,13	10
C13	**524**	551	531,80	529,50	27,25	16,33	10
C14	**679**	699	688,10	687,50	27,53	16,86	10
C15	**1.134**	1177	1.142,80	1.139	30,45	17,65	10
C16	**13**	24	16,10	15	11,88	10,00	10
C17	**25**	29	26,30	26	17,07	13,36	10
C18	**231**	238	233,60	233	36,59	27,28	10
C18	**323**	335	327	325,50	35,40	24,87	10
C20	**544**	552	547,80	547,50	45,59	33,38	10
Geral					22,56	14,80	100%

Tabela 4.13: TCP - Resultados para as instâncias da classe B. a LNS teve o parâmetro y definido como 500. O tempo (t e t_best) é dado em segundos e o gap representa a percentagem da taxa de erro. sol é a quantidade de vezes em que a LNS foi capaz de encontrar uma solução entre os dez testes realizados.

	TCP: LNS - y=500						
I	**min**	max	média	mediana	t	t_best	sol
B1	**107**	126	108,90	107	9,02	4,21	10
B2	**233**	240	234,80	233	9,23	5,23	10
B3	**1.615**	1.631	1.624	1.625	26,07	17,84	10
B4	**1.986**	2.007	1.995,20	1.994,50	30,83	18,29	10
B5	**3.295**	3.331	3.308,20	3.306,50	40,45	25,84	10
B6	**72**	88	76,7	76	12,42	10,49	10
B7	**105**	137	116,9	115	21,28	14,67	10
B8	**1.120**	1.175	1.143,6	1.140	40,62	31,31	10
B9	**1.501**	1.543	1.517	1.518	37,27	23,83	10
B10	**2.158**	2.189	2.172,60	2.171,50	38,78	23,21	10
B11	**38**	42	39,90	39	15,54	10,03	10
B12	**44**	55	49	49,50	16,73	13,20	10
B13	**531**	573	548,80	546	39,65	28,69	10
B14	**687**	709	696,40	694,50	52,95	40,72	10
B15	**1.144**	1.195	1.170,30	1.173	63,32	62,99	10
B16	**18**	24	20,80	20,50	18,43	13,49	10
B17	**24**	29	25,30	24,50	15,22	6,23	10
B18	**231**	242	237,30	237,50	36,16	23,01	10
B19	**321**	334	327,60	328	53,71	40,53	10
B20	**545**	556	548,90	548,50	60,04	44,07	10
Geral					31,89	22,89	100%

Tabela 4.14: TCP - Resultados para as instâncias da classe A. a LNS teve o parâmetro y definido como 500. O tempo (t e t_best) é dado em segundos e o gap representa a percentagem da taxa de erro. sol é a quantidade de vezes em que a LNS foi capaz de encontrar uma solução entre os dez testes realizados.

	TCP: LNS - y=500						
I	**min**	max	média	mediana	t	t_best	sol
A1	**118**	118	118	118	13,05	2,06	10
A2	**227**	247	236,38	234,50	119,33	88,85	8
A3	**4.210**	4.298	4.253,70	4.248	204,31	158,92	10
A4	**5.309**	5.353	5.327	5.323,50	285,59	195,37	10
A5	**8.255**	8.323	8.286,70	8.284	314,93	236,31	10
A6	**78**	103	81,71	78	159,89	140,41	7
A7	**156**	195	172,50	171	146,35	97,00	10
A8	**2805**	2.876	2.845,57	2.855	355,87	342,89	7
A9	**3.820**	3.916	3.868,20	3.864	391,66	313,85	10
A10	**5.775**	5.833	5.806,20	5.802,50	472,11	359,67	10
A11	**43**	67	49,90	49	134,63	103,49	10
A12	**74**	81	77,20	76,50	145,07	97,89	10
A13	**1.376**	1.443	1.417,33	1.421,50	395,61	370,07	6
A14	**1.825**	1.935	1.858,50	1.849,50	472,89	379,55	10
A15	**2.869**	2.999	2.938,57	2.944	997,02	976,42	7
A16	**20**	26	20,67	20	132,84	78,15	9
A17	**26**	37	30,10	29	130,72	112,98	10
A18	**622**	645	632,30	631	524,60	415,80	10
A19	**788**	837	803,50	802,50	582,79	505,14	10
A20	**1.372**	1.388	1.381,80	1.385	1.115,06	985,96	10
Geral					354,72	298,04	92%

Tabela 4.15: TCP - Resultados para as instâncias da classe S. A LNS teve o parâmetro y definido como 500. O tempo (t e t_best) é dado em segundos e o gap representa a percentagem da taxa de erro. sol é a quantidade de vezes em que a LNS foi capaz de encontrar uma solução entre os dez testes realizados.

	TCP: LNS - y=500						
I	**min**	max	média	mediana	t	t_best	sol
S1	**118**	123	118,50	118	12,62	2,01	10
S2	**242**	257	248	247	136,56	121,05	5
S3	**4.194**	4.293	4.250,44	4.258	204,93	168,84	9
S4	**5.317**	5.404	5.353,43	5.355	455,09	430,96	7
S5	**8.274**	8.305	8.287,40	8.284	850,11	820,35	5
S6	**78**	98	80,62	78	182,97	152,87	8
S7	**159**	175	167,33	168	247,75	226,57	3
S8	**2.815**	2.928	2.885	2.899,50	447,39	414,82	6
S9	**3.873**	3.923	3.901,67	3.909	920,23	887,11	3
S10	**5.789**	5.835	5.809,40	5.809	618,34	596,25	8
S11	**56**	61	58,25	58	180,73	180,73	4
S12	**75**	75	75	75	411,64	320,38	1
S13	**1.429**	1.439	1.434	1.434	715,30	666,89	2
S14	**1.847**	1.911	1.879	1.879	551,41	543,22	2
S15	**2.933***	-	-	-	-	-	0
S16	**16**	16	16	16	171,56	105,54	6
S17	**28**	38	32,29	31	209,53	196,56	7
S18	**609**	631	618,50	619	1.691,47	1.578,03	6
S19	**790**	812	800,33	800	1.336,63	1.236,33	9
S20	**1.368**	1.380	1.375,89	1.378	1.516,06	1.353,01	9
Geral					570,03	521,30	55%

*valor encontrado durante a criação da instância

Análise dos resultados

Como visto na Tabela 4.10 a LNS se mostra uma opção interessante para o TCP, nas instâncias da classe E, que puderam ser solucionadas pelo modelo matemático com o CPLEX, a LNS apresentou um gap médio de 0,66% e foi capaz de solucionar todas as instâncias com um tempo de execução médio vinte e duas vezes menor que o da solução exata mesmo nestas instâncias cujos relativamente pequenos. Além disso a LNS foi capaz de encontrar o resultado ótimo em catorze das dezoito instâncias da classe E.

Quanto as instâncias das outras classes, cujos resultados são apresentados nas tabelas 4.11, 4.12, 4.13, 4.14 e 4.15, não foi possível encontrar a solução exata para nenhuma das instâncias das classes mais difíceis que a E dentro do tempo limite de 3.600 segundos. É para estas instâncias que a solução heurística melhor justifica a sua existência mostrando-se muito mais adequado que a solução exata para uso em instâncias grandes e sendo capaz de encontrar uma solução para todas as instâncias das classes E, D, C, B e A e para

dezenove das vinte instâncias da classe S. Além disso a LNS foi capaz de encontrar uma solução em todas as suas execuções para as instâncias das classes E, D, C e B; e pode encontrar uma solução em 92% das execuções para a classe A e 55% para a classe S.

É importante ainda notar que as soluções apresentadas neste trabalho são as primeiras de que se tem conhecimento para o Problema de Conexão de Terminais com Número Restrito de Roteadores e Elos e tanto as soluções apresentadas quanto as instâncias criadas para as testar criam uma base inicial para o desenvolvimento e comparação de soluções para o TCP. As instâncias geradas para este trabalho estão disponíveis em um repositório no GitHub [22] para uso por trabalhos futuros.

Capítulo 5

Conclusões e Trabalhos Futuros

Este trabalho apresentou duas soluções distintas para o Problema de Conexão de Terminais com Número Restrito de Roteadores e Elos, um modelo matemático solucionado com o uso do IBM ILOG CPLEX e uma meta-heurística *Large Neighborhood Search*. Estas são as primeiras soluções de que se tem conhecimento para o problema abordado por tanto para que se pudesse traçar um paralelo a trabalhos da literatura, as soluções apresentadas foram também avaliadas para o problema de Steiner em grafos e a LNS foi comparado a meta-heurística *Parallel Hybrid* GRASP na solução do problema de Steiner.

A solução exata apresentada teve como principal objetivo permitir a avaliação da qualidade das soluções obtidas pela LNS no TCP e cumpriu o seu papel sendo capaz de solucionar as menores instâncias avaliadas neste trabalho, dando assim a possibilidade de se avaliar a taxa de erro e comparar o tempo de execução obtidos pela LNS em relação à solução exata.

A meta-heurística LNS se mostrou uma boa alternativa de baixo custo computacional em termos tanto de uso de memória quanto de tempo de CPU para o problema de Steiner. Apresentando taxas de erro médias de 1.26% com um tempo de execução aceitável, de 48.37 segundos, ou um tempo de execução muito bom, em média 6.63 segundos, para uma taxa de erro de 1.96%. Comparado ao PH-GRASP que apresentou taxa de erro de 0.31%, a LNS mostrou-se inferior como solução para o problema de Steiner em termos de qualidade de solução, entretanto, por apresentar bons tempos de execução que dependem apenas do tamanho da instância, diferentemente do PH-GRASP cujo tempo de execução pode variar bastante entre duas diferentes instâncias de mesmo tamanho, a LNS se mostrou uma boa alternativa para aplicações que priorizem a velocidade em detrimento da qualidade da solução.

Para o TCP a LNS apresentou uma taxa de erro média de 0.66% para as instâncias avaliadas, além de ter sido capaz de encontrar a solução ótima em catorze das dezoito instâncias avaliadas e de ter apresentado um tempo de execução médio vinte e duas vezes menor que o da solução exata nestas instâncias. Além disso, diferentemente da

solução exata, cujo uso de recursos cresce exponencialmente em relação ao crescimento das instâncias, o ritmo de crescimento do tempo de execução da LNS é polinomial tornando-o muito mais interessante para uso em instâncias maiores. Em comparação, a solução exata só foi capaz de resolver as dezoito instâncias da classe de instâncias mais simples analisada, enquanto a LNS foi capaz de encontrar uma solução para cento e dezenove das cento e vinte instâncias avaliadas.

Além das soluções desenvolvidas também foram criadas cento e dezoito instâncias de diferentes níveis de dificuldade para o TCP, instâncias estas que, juntamente as soluções encontradas, poderão servir como base de comparação para trabalhos futuros que venham a apresentar soluções para o problema analisado. Temos então como contribuições deste trabalho:

- As primeiras soluções (heurística e exata) para o Problema de Conexão de Terminais com Número Restrito de Roteadores e Elos
- Um conjunto de instâncias e resultados que poderão ser utilizados na avaliação de soluções apresentadas por trabalhos futuros.

Para trabalhos futuros podem-se avaliar diferentes soluções heurísticas para a fase de reconstrução da LNS bem como solucionar o restante das instâncias geradas para o problema utilizado-se da solução exata apresentada.

Bibliografia

[1] Carnegie Mellon University, School of Computer Science *Combinatorial Optimization.* http://www.cs.cmu.edu/afs/cs.cmu.edu/project/learn-43/lib/photoz/.g/web/glossary/comb.html.

[2] *Euler diagram for NP-Completeness.* https://upload.wikimedia.org/wikipedia/commons/a/a0/P_np_np-complete_np-hard.svg.

[3] *Classification of metaheuristics.* http://nojhan.free.fr/metah/images/metaheuristics_classification.jpeg. Version: 2007.

[4] *Applications of Combinatorial Optimization (Mathematics and Statistics).* Wiley-ISTE, 2014. – ISBN 1848216580.

[5] Ahuja ; K, R. ; Orlin, J. ; B, J. ; Sharma, D. ; ,. New neighborhood search structures for the capacitated minimum spanning tree problem. (1998).

[6] Beasley, J. E. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society* 41, 11 (1990), 1069–1072.

[7] Blum, C. ; Pinacho, P. ; López-Ibáñez, M. ; Lozano, J. A. "Construct, Merge, Solve & Adapt A new general algorithm for combinatorial optimization". *"Computers & Operations Research"* 68 (2016), 75 - 88.

[8] Blum, C. ; Roli, A. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Comput. Surv.* 35 (2001), 268–308.

[9] Cook, S. A. The Complexity of Theorem-proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing.* (1971), pp. 151–158.

[10] Costa, A. M. ; Cordeau, J.-F. ; Laporte, G. Fast heuristics for the Steiner tree problem with revenues, budget and hop constraints. *European Journal of Operational Research* 190, 1 (2008), 68 – 78.

[11] Divya, K. ; Amal, P. M. ; Ajish Kumar, K. S. A degree based approach to find Steiner Trees. *Electronic Notes in Discrete Mathematics* 53 (2016), 383 – 393.

[12] Dourado, M. C. ; Oliveira, R. A. ; Protti, F. ; Souza, U. S. Conexão de Terminais com Número Restrito de Roteadores e Elos. *Simpósio Brasileiro de Pesquisa Operacional* 48 (2014), 2965 – 2976.

[13] Dourado, M. C. ; Oliveira, R. A. ; Protti, F. ; Souza, U. S. Design of connection networks with bounded number of non-terminal vertices. *Matema´tica Contempor^anea, Vol. 42* 42 (2014), 39 — 48.

[14] Fortnow, L. The Status of the P Versus NP Problem. *Commun. ACM* 52, 9 (2009), 78–86.

[15] Fortnow, L. *Golden ticket :.* Princeton: : Princeton University,, 2013.

[16] Gendreau, M. ; Potvin, J.-Y. *Handbook of Metaheuristics.* 2nd. Springer Publishing Company, Incorporated, 2010. – ISBN 1441916636, 9781441916631.

[17] Harifi, S. ; Khalilian, M. ; Mohammadzadeh, J. ; Ebrahimnejad, S. Emperor Penguins Colony: a new metaheuristic algorithm for optimization. *Evolutionary Intelligence* 12, 2 (2019), 211–226.

[18] Karp, R. M. *Reducibility among Combinatorial Problems.* Boston, MA : Springer US, 1972, pp. 85–103.

[19] Kim, M. ; Choo, H. ; Mutka, M. W. ; Lim, H.-J. ; Park, K. On QoS multicast routing algorithms using k-minimum Steiner trees. *Information Sciences* 238 (2013), 190 – 204.

[20] Kleinberg, J. ; Tardos, É. *Algorithm Design.* Pearson/Addison-Wesley, 2006 (Pearson international edition). – ISBN 9780321372918.

[21] Leeuwenvan , J. *Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity.* The MIT Press, 1994. – ISBN 0262720140.

[22] Libório, F. *Repositório contendo os assets do trabalho.* `https://github.com/felipeliborio/Optimization/Graphs/TCP`. Version: 2019.

[23] Lucena, A. ; Beasley, J. E. A branch and cut algorithm for the Steiner problem in graphs. *Networks* 31, 1, 39-59.

[24] Martins, S. ; Resende, M. ; Ribeiro, C. ; Pardalos, P. A Parallel Grasp for the Steiner Tree Problem in Graphs Using a Hybrid Local Search Strategy. *Journal of Global Optimization* 17, 1 (2000), 267–283.

[25] Melo, A. A. d. *Conexão de terminais com limitação de roteadores: complexidade e relação com fluxos e caminhos.* Rio de Janeiro, 2017.

[26] Moscato, P. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms. *Caltech Concurrent Computation Program* (2000).

[27] Neri, F. ; Cotta, C. Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation* 2 (2012), 1 - 14.

[28] Pisinger, D. ; Røpke, S. Large Neighborhood Search. In *Handbook of Metaheuristics* 2, (2010), pp. 399–420.

[29] Prim, R. C. Shortest connection networks and some generalizations. *The Bell System Technical Journal* 36, 6 (1957), 1389–1401.

[30] Rao, S. S. *Engineering optimization: theory and practice.* John Wiley & Sons, 2009.

[31] Shaw, P. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In Maher, M. (Hrsg.) ; Puget, J.-F. (Hrsg.): *Principles and Practice of Constraint Programming — CP98.* (1998), pp. 417–431.

[32] Talbi, E.-G. *Metaheuristics: From Design to Implementation.* Wiley, 2009. – ISBN 0470278587.