UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

IGOR LUCIANO CAVALCANTI LIMA

META-HEURÍSTICA GRASP APLICADA AO PROBLEMA DE LOCALIZAÇÃO DE CONTADORES DE TRÁFEGO

São Luís
2018

UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

IGOR LUCIANO CAVALCANTI LIMA

META-HEURÍSTICA GRASP APLICADA AO PROBLEMA DE LOCALIZAÇÃO DE CONTADORES DE TRÁFEGO

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Francisco Glaubos Nunes Clímaco

São Luís
2018

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Cavalcanti Lima, Igor Luciano.
META-HEURÍSTICA GRASP APLICADA AO PROBLEMA DE LOCALIZAÇÃO DE CONTADORES DE TRÁFEGO / Igor Luciano Cavalcanti Lima. - 2018.
43 f.

Orientador(a): Francisco Glaubos Nunes Clímaco.
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luís, 2018.

1. GRASP. 2. Localização de Contadores de Tráfego. 3. Meta-heurística. I. Nunes Clímaco, Francisco Glaubos. II. Título.

IGOR LUCIANO CAVALCANTI LIMA

META-HEURÍSTICA GRASP APLICADA AO PROBLEMA DE LOCALIZAÇÃO DE CONTADORES DE TRÁFEGO

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Aprovada em 12 de Julho de 2018

BANCA EXAMINADORA

Francisco Glaubos

Prof. Me. Francisco Glaubos Nunes Clímaco
Universidade Federal do Maranhão - UFMA
Orientador

Mário Teixeira

Prof. Dr. Mário Antonio Meireles Teixeira
Universidade Federal do Maranhão - UFMA
Membro da Banca

Tiago Bonini Borchartt

Prof. Dr. Tiago Bonini Borchartt
Universidade Federal do Maranhão - UFMA
Membro da Banca

São Luís
2018

À minha mãe.

AGRADECIMENTOS

À minha mãe Fátima e meu padrasto Marco, por todo apoio necessário, sem o qual a realização deste trabalho não seria possível.

À minha fiel companheira Thamiris, por me apoiar e motivar nos momentos difíceis, partilhar e comemorar as felicidades durante toda a minha graduação.

Ao meu orientador Glaubos, por acreditar no meu trabalho, passar seus aprendizados e fornecer todo suporte necessário. Ao professor Alexandre, por toda ajuda, compreensão e empenho nos meus primeiros passos da pesquisa científica.

Aos amigos do LACMO Ramon, Marcelo e Raphael, por toda ajuda e colaboração durante a minha permanência no laboratório.

Aos amigos Lucas, Giovanni, Eduardo, Emanuel, Paulo, Werliton e todos que, direta ou indiretamente, participaram dessa conquista.

"A verdadeira viagem de descobrimento
não consiste em procurar novas paisagens,
mas em ter novos olhos."
(Marcel Proust)

RESUMO

Muitos problemas de otimização combinatória do tipo NP-difícil têm importante aplicações práticas. Apresentar soluções ótimas exatas para esse tipo de problema geralmente requer elevado custo computacional. Esses problemas, em sua maioria, apresentam instâncias reais de grande porte e resolvê-las por meio de algoritmos exatos torna-se inviável. Por esse motivo, nas últimas décadas, diversas pesquisas voltadas para a utilização de meta-heurísticas, como proposta para resolução de problemas desta ordem, foram desenvolvidas. O principal atrativo está na qualidade das soluções apresentadas em tempo computacional aceitável, principalmente para grandes instâncias que normalmente representam dados reais dos problemas. Este trabalho propõe a utilização da meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP) para resolução do Problema de Localização de Contadores de Tráfego (PLCT), um problema do tipo NP-difícil que consiste em determinar a quantidade e localização de contadores numa rede de transporte. A validação da meta-heurística é efetuada realizando-se testes em instâncias reais obtidas a partir de dados do Departamento Nacional de Infraestrutura de Transporte (DNIT) sobre estados brasileiros. Os resultados da abordagem são comparados à outras estratégias presentes na literatura.

Palavras-chave: GRASP. Localização de Contadores de Tráfego. Meta-heurística.

ABSTRACT

Many combinatorial optimization problems of the NP-hard type have important practical applications. Presenting exact optimal solutions for these problems often demands high computational cost. Most part of these problems present large real instances and solve them by exact algorithms becomes impracticable. For this reason, in the last decades, several studies aimed at the use of metaheuristics as a proposal to solve problems of this order were developed. The main attraction is the quality of the solutions presented in acceptable computational time, especially for large instances that usually represent real problems data. This work proposes the use of the Greedy Randomized Adaptive Search Procedure (GRASP) for the resolution of the Traffic Counters Location Problem (TCLP), a NP-hard problem that consists of determining the quantity and location of counters in a transport network. In order to validate the metaheuristic, computational experiments were performed in real instances obtained from data of Departamento Nacional de Infraestrutura de Transportes (DNIT) on Brazilian states. The approach results are compared to other strategies present in the literature.

Keywords: GRASP. Traffic Counters Location. Metaheuristic.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

AG	Algoritmo Genético
DNIT	Departamento Nacional de Infraestrutura de Transporte
FO	Função Objetivo
GRASP	*Greedy Randomized Adaptive Search Procedure*
LRC	Lista Restrita de Candidatos
OC	Otimização Combinatória
OD	Origem-Destino
POD	Par Origem-Destino
PLCT	Problema de Localização de Contadores de Tráfego
PNCT	Plano Nacional de Contagem de Tráfego
VNS	*Variable Neighborhood Search*

LISTA DE SÍMBOLOS

α	Letra grega Alpha
δ	Letra grega Delta
ϕ	Letra grega Fi
ψ	Letra grega Psi
ρ	Letra grega Rô

SUMÁRIO

1 INTRODUÇÃO

Problemas de otimização podem ser encontrados nas mais diversas áreas nos ramos da indústria ou da ciência. Dentre os diversos problemas reais práticos, podemos citar distintas aplicações como problemas de roteamento, escalonamento, localização de instalações, entre outros. Encontrar a solução ótima destes problemas geralmente pode consumir um elevado tempo de processamento, inviabilizando métodos exatos para resolvê-los. Na prática, nós normalmente estaremos satisfeitos com "boas"soluções, as quais podem ser obtidas por meio de heurísticas e meta-heurísticas (TALBI, 2009).

Para problemas de Otimização Combinatória (OC) do tipo NP-difícil, nos quais soluções ótimas são computacionalmente difíceis de serem encontradas ou suficientemente grandes para considerar técnicas exatas, as meta-heurísticas são empregadas para encontrar boas soluções, mas não necessariamente ótimas, com um menor custo computacional. Por esse motivo, há um significativo crescimento de interesse no domínio de métodos que empreguem técnicas heurísticas. Na literatura encontramos diversos métodos consolidados e comprovadamente eficientes utilizando meta-heurísticas, dentre eles podemos destacar: GRASP (FEO; RESENDE, 1995), Algoritmos Genéticos (HOLLAND, 1992), *Simulated Annealing* (KIRKPATRICK; GELATT; VECCHI, 1983) entre outros. A efetividade destes métodos depende da capacidade de se adaptar a um problema particular, evitar armadilhas de ótimos locais e explorar a estrutura básica do problema (FESTA; RESENDE, 2009a).

Meta-heurísticas podem ser definidas como procedimentos de alto nível que guiam e modificam heurísticas subordinadas a fim de produzir soluções aproximadas de boa qualidade. São essencialmente empregadas na resolução de problemas de otimização e geralmente utilizam algoritmos mais simples que os métodos exatos. Para serem efetivas, as meta-heurísticas proveem ferramentas rápidas e robustas que garantem estas soluções em tempo computacional razoável (RIBEIRO; HANSEN, 2012).

Este trabalho aborda o Problema de Localização de Contadores de Tráfego (PLCT), um problema de OC do tipo NP-difícil no qual a sua solução consiste em determinar a quantidade e a localização de contadores em uma rede de transportes. Dado um conjunto de pares Origem-Destino (*OD*) e dados todos os caminhos possíveis entre dois pares, qualquer fluxo entre uma origem e um destino deverá passar obrigatoriamente por, pelo menos, um contador.

1.1 Motivação

O planejamento de melhorias e ações nas redes de transporte necessitam de informações como volume de tráfego, tráfego médio anual, tráfego médio diário, entre outros. Essas informações são obtidas a partir da matrizes *OD* que podem ser utilizadas para uma representação espacial das viagens de veículos entre diferentes zonas em uma rede de transporte. A sua estima-

ção, porém, exige grande esforço e estudos em larga escala. Uma maneira simples e eficaz de efetuar essa estimação está na utilização de contadores de tráfego para o monitoramento do fluxo e atualização das matrizes *OD*. O monitoramento por meio de contadores permite descrever características do tráfego de uma região que ajudam no planejamento de melhorias.

Entre os métodos de contagem de tráfego, podemos destacar as contagens manuais, semiautomáticas e completamente automáticas (GONZÁLEZ et al., 2016). As contagens manuais são normalmente realizadas por apenas um curto período de tempo e possuem limitações por falhas humanas. As contagens semiautomáticas contam com o auxílio de softwares, transmissores e outros a fim de reduzir esses erros e tornar mais ágil o processo de contagem. Por fim, contagens totalmente automáticas evitam o envolvimento humano e utilizam sensores que detectam a passagem do veículo e enviam a informação a um contador. Segundo González (2016), independente do método de contagem utilizado, as pesquisas de tráfego são custosas, necessitam de mão-de-obra específica e possuem orçamento reduzido.

O PLCT permite a redução de custos envolvendo a contagem de tráfego ao determinar a quantidade e a localização dos contadores. Uma maneira de determinar a solução para o problema consiste em enumerar todas as combinações possíveis de pontos de localização de contadores, porém o custo computacional associado a essa enumeração cresce exponencialmente com respeito ao tamanho da rede. Portanto, dadas as características do PLCT e a dificuldade em determinar soluções ótimas exatas para o problema, principalmente para instâncias de grande porte, a utilização de meta-heurísticas para a sua resolução é encorajada.

Neste sentido, este trabalho tem como objetivo principal propor uma alternativa aos métodos construtivos encontrados na literatura para a resolução do PLCT. Para tal, utilizou-se a meta-heurística GRASP (Procedimento de Busca Guloso, Aleatório e Adaptativo do inglês *Greedy Randomized Adaptive Search Procedure*). O GRASP é uma estratégia multipartida proposta inicialmente por Resende (1996) que sugere sua definição como um procedimento guloso, aleatório e adaptativo que repetidamente aplica uma busca local a soluções construídas com um algoritmo guloso e aleatório. O GRASP se encaixa na categoria de meta-heurística de única solução ou caminho, na qual a melhor solução encontrada pela meta-heurística é guardada e retornada como resultado.

A fim de validar a abordagem proposta, experimentos computacionais foram realizados a partir de instâncias reais do PLCT, fornecidas pelo Departamento Nacional de Infraestrutura de Transportes (DNIT), e seus resultados comparados a uma outra abordagem presente na literatura.

1.2 Organização do trabalho

O restante deste trabalho está organizado da seguinte maneira. No Capítulo 2 o problema tratado é apresentado em mais detalhes, inclusive com sua definição matemática. O Capítulo 3 apresenta a meta-heurística GRASP e seu funcionamento, bem como suas particularidades,

técnicas e melhorias. O Capítulo 4 apresenta a aplicação do método GRASP ao PLCT, formulando uma construção gulosa e aleatória adequada juntamente com um método de busca local. O Capítulo 5 contém os experimentos computacionais realizados e seus resultados. Finalmente, no Capítulo 6, é apresentada a conclusão deste trabalho e propostas de trabalhos futuros.

2 APRESENTAÇÃO DO PROBLEMA

O Problema de Localização de Contadores de Tráfego é um problema do tipo NP-Difícil e trata do problema de posicionar contadores em uma rede de transporte tendo como objetivo utilizar o número mínimo de contadores que seja suficiente para realizar a contagem do fluxo de tráfego entre pares de Origem-Destino (*OD*) previamente definidos. Dessa forma, para garantir a contagem, qualquer caminho tomado entre um par *OD* deverá passar obrigatoriamente por, pelo menos, um contador.

Uma rede de transporte pode ser definida como um grafo $G = (V,A)$ não orientado, onde A é o conjunto de arestas e V o conjunto de vértices. W representa o conjunto de pares origem-destino. Para cada par $\{i, j\} \in OD$, temos pelo menos um caminho simples a partir da origem i com destino em j.

Deste modo, considere então $x_a = \{0,1\}$ com $a \in A$ uma variável binária que determina a presença ou não de um contador. Portanto, $x_a = 1$ indica que um contador está presente na aresta a e 0 caso contrário. Yang *et al*. (1992) propuseram um modelo matemático para o PLCT que pode ser definido da seguinte maneira:

$$Minimizar \;\; Z = \sum_{a \in A} x_a \tag{2.1}$$

sujeito a

$$\sum_{a \in A} \delta_{ra}^{w} x_a \geq 1 \quad r \in R_w, w \in W \tag{2.2}$$

$$x_a \in \{0,1\} \quad a \in A \tag{2.3}$$

sendo R_w o conjunto de caminhos entre cada par Origem-Destino $w \in W$, e $\delta_{ra}^{w} x_a = 1$ caso a aresta a esteja no caminho r entre o par w e 0 caso contrário. A função objetivo 2.1 é responsável por calcular a quantidade de contadores necessários para a cobertura total do fluxo entre cada par *OD* no grafo. A restrição 2.2 garante que todo par *OD* terá, para cada caminho que liga uma origem a um destino, pelo menos um contador em alguma das arestas que compõem este caminho. Por fim, a restrição 2.3 define o domínio da variável de decisão.

O PLCT surge a partir do problema de estimar matrizes *OD* que podem ser utilizadas para uma representação espacial das viagens de veículos entre zonas em uma rede de transporte. Segundo Mauri *et al*. (2017) as matrizes *OD* são fontes de informação para diversos estudos de transporte, como previsão de demanda futura de viagens, gestão e controle. A estimação dessas matrizes exige normalmente estudo em grande escala que pode ser custoso em termos de tempo e trabalho. Porém, a estimação dessas matrizes a partir da definição da localização de contadores é uma forma conveniente e prática, a partir da contagem de tráfego, de estimar estas matrizes.

O PLCT é formulado como um problema de programação inteira e sua complexidade é do tipo NP-difícil. Uma abordagem para a resolução deste problema consiste em enumerar

todos os caminhos possíveis para cada um dos pares *OD*. Contudo, esse procedimento requer um tempo computacional inviável para redes de grandes dimensões, uma vez que o número de caminhos entre cada par *OD* cresce exponencialmente com respeito ao tamanho da rede (MAURI et al., 2017). Portanto, o uso de meta-heurísticas para a resolução do problema é incentivado.

Na Figura 1 é ilustrado um grafo que representa uma instância do problema. O conjunto de vértices destacados $V_{OD} = \{A, G, H\}$ combinados dois a dois representam os pares *OD*, enquanto as arestas sinalizadas indicam a presença de um contador. Para esse grafo, os contadores de tráfego estão posicionados nas arestas de modo a cobrir todo o fluxo entre os pares *OD*. Isso significa que, dado um par (i, j) do conjunto V_{OD} com $i \neq j$, todos os caminhos possíveis com origem em i e destino em j passará por, pelo menos, um contador.

O problema do PLCT tem como objetivo não apenas a determinação da localização dos contadores, mas também diminuir a quantidade utilizada. Na Figura 1 é possível observar que esta configuração é ideal e portanto ótima, pois representa a quantidade mínima necessária de contadores para cobertura de todo o fluxo na rede entre qualquer par *OD*. Apesar de mínima, esta não é a única configuração possível. Desta maneira, diferentes configurações de soluções ótimas podem ser encontradas para uma mesma instância do PLCT, mas todas com a mesma quantidade de contadores posicionados.

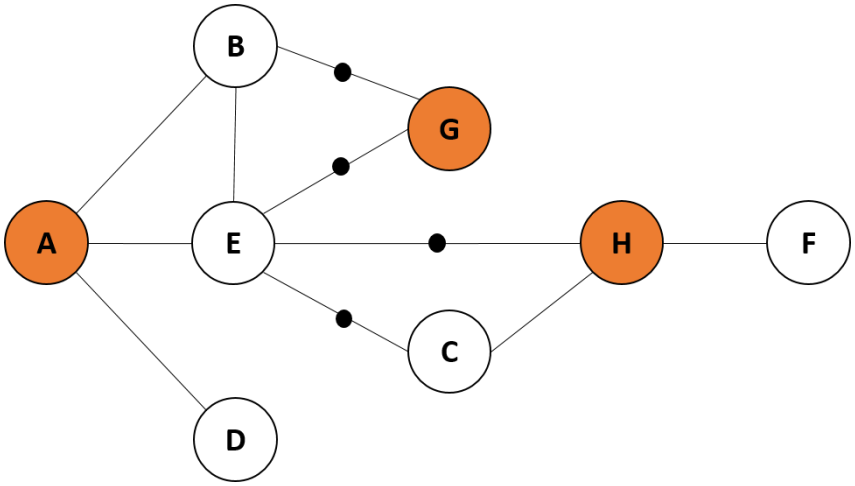

Figura 1 – Ilustração de um grafo que representa uma instância do PLCT.

Neste trabalho, utilizaremos como instâncias do problema uma representação de dados reais baseados nos estados brasileiros. Os vértices representam os nós das rodovias, os municípios são um tipo especial de nó e representam os pares *OD*, enquanto as arestas são representações de segmentos das rodovias federais ou estaduais. Esses dados são fornecidos pelo Departamento Nacional de Infraestrutura de Transportes (DNIT) através do Plano Nacional de Contagem de Tráfego (PNCT), que tem por objetivo planejar o sistema rodoviário a fim de prover informações sobre demandas de tráfego, avaliações de fluxos, necessidades de melhorias, entre outros (DNIT, 2018).

Na Figura 2 podemos observar o mapa do Brasil com as representações das rodovias

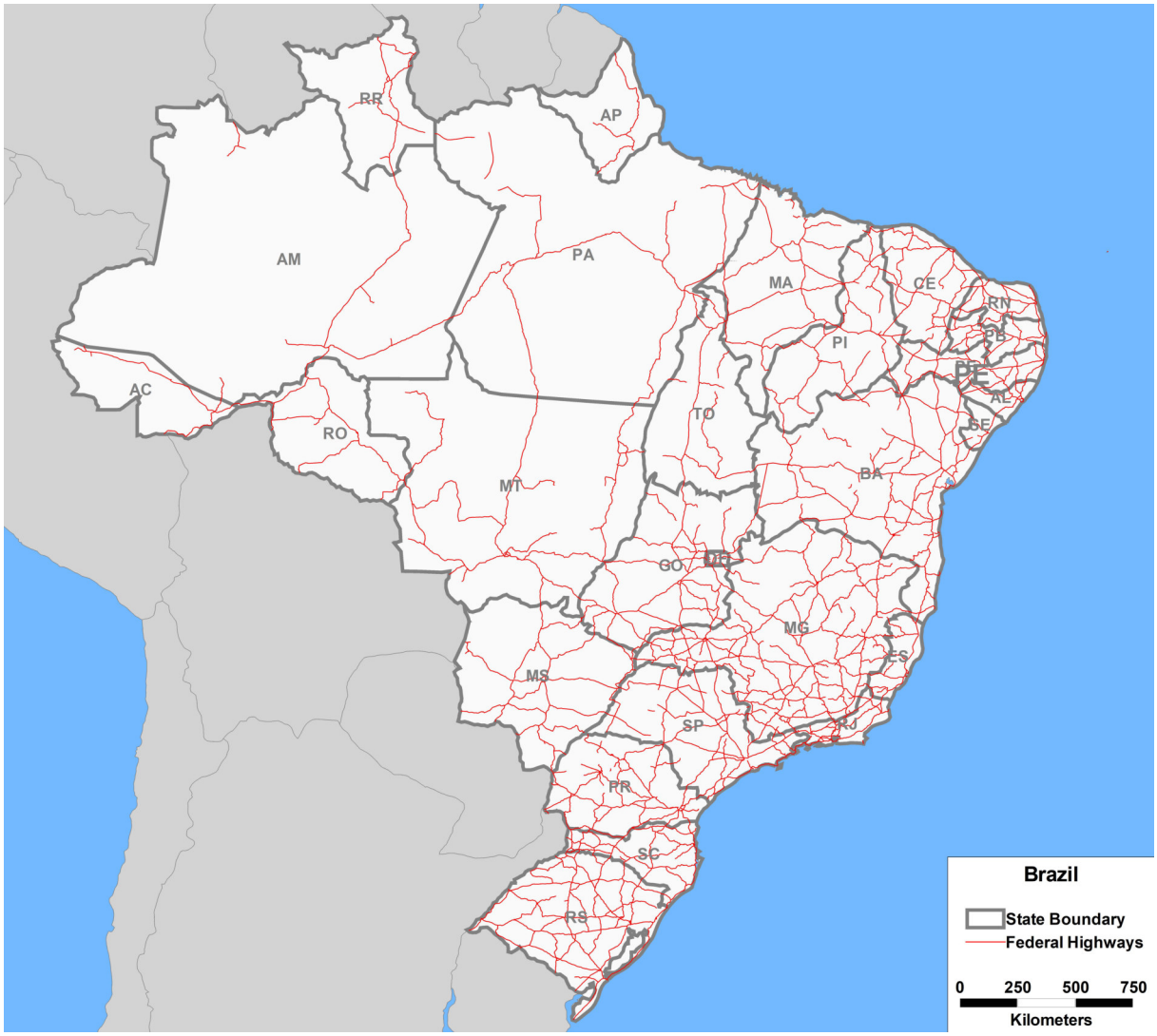

Figura 2 – Mapa do Brasil com a representação das rodoviais federais que interligam aproximadamente 5600 municípios brasileiros.

interligando aproximadamente 5600 municípios. O PNCT visa instalar contadores de tráfego para a estimação de fluxo entre pares de Origem-Destino compostos pelos municípios brasileiros, porém a rede completa contém mais de 24 mil arestas e mais de 20 mil nós. Portanto, encontrar a melhor configuração para a instalação de contadores de tráfego nas rodovias facilita a estimação e atualização das matrizes *OD* necessárias ao estudo dos fluxos de transporte.

Dado que o Brasil inteiro contém uma rede extremamente grande e difícil de se trabalhar, é possível dividi-la utilizando as fronteiras entre os estados como forma de diminuir a quantidade de nós e arestas envolvidos. Na Figura 3, é ilustrada uma solução com contadores de tráfegos posicionados para observar o fluxo entre todos os pares *OD*, compostos pelos municípios do estado do Rio de Janeiro.

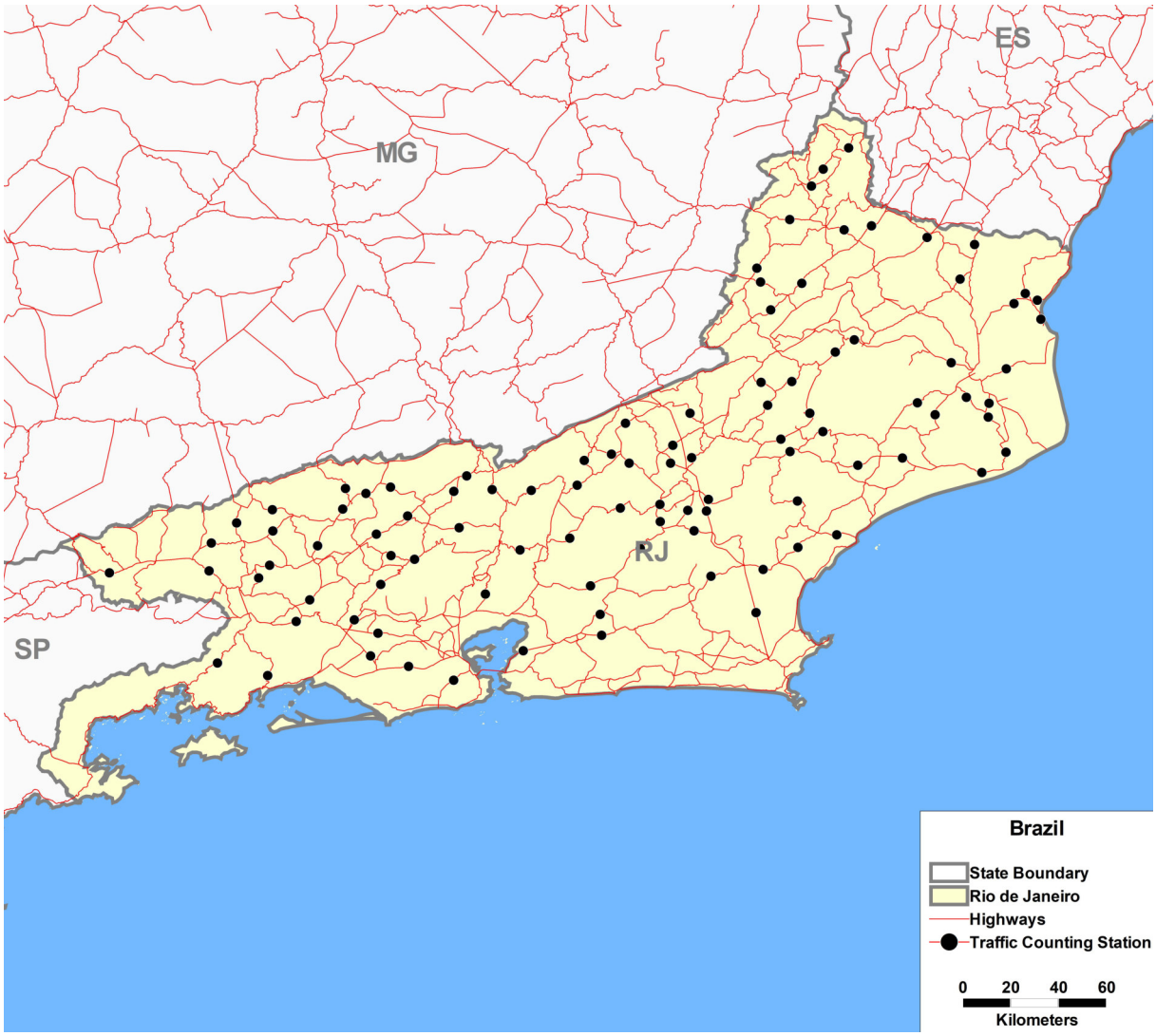

Figura 3 – Representação das rodoviais e contadores de tráfego posicionados para o estado do Rio de Janeiro.

3 FUNDAMENTAÇÃO TEÓRICA

Muitos problemas de otimização com importantes implicações práticas e teóricas possuem como decisão definir a melhor configuração de um conjunto de parâmetros para atingir determinado objetivo. Ao longo de décadas, muitos problemas de otimização surgiram nas mais diversas áreas da indústria e ciência, como problemas de escalonamento, roteamento, localização de instalações, entre outros. Estes problemas de otimização foram divididos em dois grandes grupos: problemas com variáveis contínuas e problemas com variáveis discretas (PAPADIMITRIOU; STEIGLITZ, 1998). Estes últimos, definidos como tipos de Problemas de Otimização Combinatória.

Mateus, Resende e Silva (2010) sugerem a definição de um problema de otimização combinatória como um conjunto finito $E = \{1, ..., n\}$, um conjunto de soluções viáveis $F \in 2^E$ e uma função objetivo $f : 2^E \rightarrow \mathbb{R}$, todos definidos para cada problema específico. Problemas de otimização podem ser do tipo maximização ou minimização. Pretendemos otimizar o valor de f, tal que a solução ótima é definida como $S^* \in F$ e $f(S^*) \geq f(S), \forall S \in F$ para problemas de maximização e $f(S^*) \leq f(S), \forall S \in F$ para os problemas de minimização.

Entre as classes de problemas de Otimização Combinatória, encontramos um tipo denominado NP-difícil. Problemas dessa ordem geralmente possuem alta dificuldade de resolução, uma vez que a quantidade de combinações de soluções é muito grande e geralmente escalam de maneira exponencial. Portanto, algoritmos que geram soluções exatas para esse tipo de problema tornam-se inviáveis devido ao custo computacional elevado. Apesar do progresso no desenvolvimento de técnicas exatas para a resolução desses problemas, as heurísticas ainda são muito utilizadas devido à sua agilidade e robustez na geração de soluções viáveis, mas não necessariamente ótimas. Exemplos de meta-heurísticas comprovadamente eficientes são encontradas na literatura, tais como *Simulated Annealing*, Algoritmos Genéticos, GRASP, entre outros.

As próximas seções detalham os conceitos de meta-heurísticas, algumas abordagens comumente empregadas e um estudo detalhado sobre a meta-heurística tema deste trabalho, o Procedimento de Busca Guloso, Aleatório e Adaptativo (GRASP).

3.1 Meta-heurísticas

Encontrar a solução ótima é intratável para muitos dos problemas de otimização existentes. Na prática, estaremos satisfeitos com boas soluções, as quais podem ser obtidas por heurísticas ou meta-heurísticas. As meta-heurísticas representam uma família de técnicas de otimização aproximadas que proveem uma solução "aceitável"em tempo computacional razoável. Diferentemente de técnicas exatas, as meta-heurísticas não garantem que suas soluções encontradas sejam ótimas, tampouco garantem a proximidade, tais como algoritmos de aproximação.

A palavra "heurística"deriva do grego *heuriskein*, que significa "a arte de descobrir novas estratégias para resolver problemas". O prefixo *meta* também deriva do grego e significa sua colocação implica em algo que está em "alto nível"(TALBI, 2009). Portanto, meta-heurísticas podem ser definidas como procedimentos de alto nível que guiam e modificam heurísticas subordinadas a fim de produzir soluções aproximadas de boa qualidade. Nas últimas décadas, houve um intenso crescimento no domínio das meta-heurísticas devido à facilidade de adaptá-las a problemas específicos e sua capacidade em gerar soluções de alta qualidade, até mesmo ótimas, para muitos problemas conhecidos.

Uma meta-heurística terá sucesso para um dado problema de otimização se esta consegue prover um balanço entre exploração e diversificação (BOUSSAÏD; LEPAGNOT; SIARRY, 2013). A exploração tem por objetivo maximizar o espaço de busca de modo a diversificar a qualidade das soluções. A intensificação tem por objetivo buscar ótimos locais e melhorar uma solução encontrada. A diferença entre as diversas meta-heurísticas encontradas na literatura está em como cada uma delas consegue encontrar este balanceamento.

As meta-heurísticas podem ser classificadas em relação a vários aspectos, como uso de memória, exploração do espaço de busca ou número de soluções correntes mantidas durante suas iterações. Este último aspecto divide as meta-heurísticas em dois grandes grupos: meta-heurísticas baseadas em solução única e meta-heurísticas baseadas em população.

As meta-heurísticas baseadas em solução únicas tem por objetivo melhorar uma única solução e são consideradas meta-heurísticas de trajetória ou caminho. A cada iteração, uma nova solução melhor que a anterior é procurada no espaço de busca. A solução corrente é então substituída pela melhor encontrada. Esse processo se repete até um dado critério de parada. Exemplos populares desse tipo de meta-heurísticas são: Busca Local, *Simulated Annealing*, Busca Tabu, entre outros.

Já as meta-heurísticas baseadas em população trabalham com um conjunto de soluções em vez de uma única solução e podem ser vistas como processo de melhoria da população iterativo. Inicialmente uma população é gerada por meio de alguma técnica de geração de população. Entre as técnicas mais utilizadas, estão as construções gulosas e inicializações aleatórias. A partir da solução inicial, uma nova população é gerada por meio de técnicas de combinações de soluções a fim de procurar outras melhores que as atuais. Finalmente, essa nova população é integrada à população corrente de acordo com os mecanismos de seleção definidos por cada método. Este processo é então repetido até um dado critério de parada ser atingido. Alguns exemplos de meta-heurísticas baseadas em população são: Algoritmos de Estimação de Distribuição, Colônia de Abelhas, Colônia de Formigas, Algoritmos Genéticos, entre outros.

Muitos trabalhos presentes na literatura apresentam soluções baseadas em meta-heurísticas que unem aspectos de diferentes abordagens, são as chamadas meta-heurísticas híbridas. Este tipo de meta-heurística procura unir duas ou mais técnicas conhecidas a fim de prover um único algoritmo para resolver um dado problema. Um exemplo clássico desta hibridização está na

união do GRASP ao AG (Algoritmo Genético). Geralmente o AG gera uma solução inicial de maneira aleatória, porém esta geração pode comprometer o comportamento do método uma vez que boas soluções podem não ser encontradas. Neste caso, o GRASP se apresenta como uma alternativa melhor de gerar boas soluções iniciais. O GRASP deverá ser rápido o suficiente para não comprometer o desempenho da estratégia, visto que construções gulosas e inicializações aleatórias são muito menos custosas, além de gerar boas soluções que servirão de ponto de partida para as seguintes iterações do AG.

3.2 GRASP

Procedimento de busca guloso, aleatório e adaptativo (do inglês *Greedy Randomized Adaptive Search Procedure* - GRASP) foi apresentado inicialmente por Feo e Resende (1995) e, desde então, tem sido largamente aplicado com sucesso na resolução de problemas de Otimização Combinatória (FESTA; RESENDE, 2009a; FESTA; RESENDE, 2009b). A meta-heurística consiste num método de busca local que atua repetidamente a partir de soluções construídas por um algoritmo guloso aleatório. Um aspecto fortemente atraente do GRASP é a facilidade com que pode ser implementado, dado que poucos parâmetros devem ser inicializados e ajustados.

O GRASP é um procedimento iterativo, onde ao final de cada iteração uma nova solução é gerada e consiste em duas fases: uma fase construtiva e uma busca local. A fase construtiva gera uma solução viável adicionando elementos que são avaliados e ordenados por uma função *f(s)*. Em seguida, uma busca local é aplicada a fim de explorar a vizinhança da solução construída até encontrar um ótimo local. A melhor solução entre todas encontradas é guardada como resultado. O Algoritmo 1 apresenta o pseudo-código de um GRASP genérico.

As iterações do algoritmo são encontradas entre as linhas 2 e 6 e terminam quando um critério de parada é atingido. Este critério é um dos dois parâmetros essenciais ao GRASP, sobre os quais iremos falar mais adiante. Na linha 3 ocorre a fase de construção da solução, enquanto a linha 5 é responsável por efetuar um procedimento de reparo quando a solução construída não é viável. Na linha 6 a fase de busca local é iniciada e, por fim, se uma solução melhor foi encontrada, na linha 9 esta solução é guardada. A melhor solução é então retornada na linha 13. Uma revisão detalhada do algoritmo é descrita em (FESTA; RESENDE, 2011).

Apenas dois parâmetros são necessários na implementação do GRASP da maneira mais convencional. O primeiro parâmetro define o número de iterações a serem efetuadas pelos métodos construtivos e busca local. Este parâmetro pode ser, por exemplo, uma quantidade definida de iterações ou um critério de parada baseado em melhora de solução. O segundo parâmetro, denominado α, define o compromisso entre uma estratégia mais gulosa ou mais aleatória durante o método construtivo. Outras implementações do GRASP, como métodos híbridos, podem definir outros parâmetros a serem implementados.

A despeito de sua simplicidade e facilidade de implementação, o GRASP é uma meta-

Algoritmo 1: GRASP

```
   Entrada: Critério de parada
   Saída: Solução x*
1  início
2      enquanto critério de parada não satisfeito faça
3          x ← ConstrucaoGulosaAleatoria();
4          se x não for viável então
5              x ← reparo(x);
6          fim
7          x ← BuscaLocal(x)
8          se f(x) < f(x*) então
9              x*← x;
10         fim
11     fim
12 fim
13 retorna x*;
```

heurística muito eficaz e produz as melhores soluções para muitos problemas (MATEUS; RESENDE; SILVA, 2010). Aplicações recentes do GRASP que obtiveram sucesso podem ser encontradas nos trabalhos de Lima (2014) e García-Archilla (2013).

3.2.1 Construção Gulosa Aleatória

Esta fase da meta-heurística GRASP visa construir soluções diversificadas para então serem aprimoradas durante a fase de busca local. O elemento de aleatoriedade incluído neste momento garante variabilidade de soluções, enquanto a qualidade dessas soluções está relacionada à função gulosa utilizada para avaliar e selecionar indivíduos.

Partindo inicialmente de uma solução vazia, uma solução completa é construída iterativamente elemento por elemento. A cada iteração, a escolha do próximo elemento a ser adicionado é determinado ordenando todos os elementos candidatos numa lista de acordo com uma função gulosa que calcula o grau de adaptação de cada um dos elementos candidatos (FESTA; RESENDE, 2011). A qualidade de um elemento é determinada pela contribuição, naquela iteração, para a solução em construção.

A heurística de construção gulosa é adaptativa devido aos benefícios associados à inserção de cada um dos candidatos serem recalculados a cada nova iteração. Já a caracterização aleatória da heurística gulosa está presente na escolha randômica de um novo elemento a ser inserido na solução que está em construção. Os melhores candidatos a serem inseridos, definidos a partir do parâmetro α, compõem uma *Lista Restrita de Candidatos* (LRC). Destes, apenas um elemento, escolhido aleatoriamente, será o novo elemento a entrar na solução.

O pseudo-código genérico de uma construção aleatória gulosa é apresentado no Algoritmo 2 (MATEUS; RESENDE; SILVA, 2010). Nas linhas 3 a 6 a solução é construída

Algoritmo 2: CONSTRUÇÃO GULOSA ALEATÓRIA

Entrada: E = {conjunto discreto finito}
Saída: Solução *S*

```
1  início
2      S ← 0;  C ← 0;
3      enquanto solução não construída faça
4          Para todo c ∈ C computar o valor da função gulosa g(c);
5          Construir a lista restrita de candidatos LRC com base no parâmetro α;
6          Selecionar aleatoriamente c*∈ LRC(C);
7          Adicionar c* à solução parcial: S ← S∪{c*};
8          Seja C o conjunto de elementos que podem ser adicionados à S;
9      fim
10 fim
11 retorna S;
```

iterativamente. A cada iteração, na linha 4 será calculado o valor da função gulosa para cada um dos candidatos determinando seu grau de adaptação. Na linha 5 ocorre a construção da lista restrita de candidatos. Um candidato é selecionado aleatoriamente na linha 6. Na linha 7 o candidato é adicionado à solução e, por fim, na linha 8 o conjunto de candidatos *C* é reconstruído.

Na literatura, outros métodos de construção tem sido propostos. Por exemplo, um método de construção denominado reativo, onde o parâmetro α sofre uma variação de valor durante as iterações do GRASP. Em seu trabalho, Prais e Ribeiro (PRAIS; RIBEIRO, 2000) mostraram que a utilização de um α fixo na construção da LRC pode muitas vezes impedir que melhores soluções sejam encontradas quando eventualmente poderiam ser utilizados valores diferentes para este parâmetro. Outra variação do método construtivo é a perturbação de custos, onde perturbações aleatórias são introduzidas com o objetivo de aleatorizar o processo de construção da LRC. Estas duas construções são detalhadas nas subseções a seguir.

3.2.1.1 Construção Reativa

O GRASP Reativo, inicialmente proposto por Prais e Ribeiro (2000), tem por objetivo suprir a falta de memória de longo prazo da meta-heurística durante o processo de construção da solução. Enquanto no GRASP tradicional as solução anteriormente encontradas não influenciam nas próximas a serem construídas, o GRASP Reativo é guiado pela qualidade destas soluções anteriores para auto-ajustar o valor de α a ser utilizado na construção da solução durante uma iteração.

O parâmetro α é basicamente o único parâmetro a ser ajustado na implementação do GRASP. Este parâmetro define o comportamento do método em termos de qualidade e diversidade da fase construtiva e consequente impacto na melhor solução encontrada ao fim da execução da meta-heurística. Em vez de utilizar um α fixo durante toda a execução, o GRASP reativo propõe uma escolha randômica a partir de um conjunto discreto $\psi = \{\alpha_1, ..., \alpha_m\}$ pré-determinado com

valores aceitáveis para α. O uso de diferentes valores de α a cada iteração permite que diferentes soluções de alta qualidade sejam construídas, as quais poderiam nunca ser encontradas caso fosse utilizado um único valor fixo.

Seja $\psi = \{\alpha_1, ..., \alpha_m\}$ definido como o conjunto de possíveis valores para α. Na primeira iteração do GRASP, todos os valores de α têm a mesma probabilidade de escolha (Equação 3.1).

$$p_i = \frac{1}{m}, \quad i = \{1, ..., m\} \tag{3.1}$$

A cada iteração subsequente, seja z^* a melhor solução encontrada até aquele ponto e seja A_i a média de todas as soluções encontradas utilizando $\alpha = \alpha_i$. As probabilidades de seleções são periodicamente avaliadas seguindo a equação 3.2.

$$p_i = \frac{q_i}{\sum_{j=1}^{m} q_j} \tag{3.2}$$

$$q_i = \frac{z^*}{A_i}, \quad i = \{1, ..., m\} \tag{3.3}$$

O valor de q_i será maior para os valores de $\alpha = \alpha_i$, com $i = \{1, ..., m\}$, associados às melhores soluções em média. Deste modo, q_i corresponde aos valores mais apropriados para α e suas probabilidades serão aumentadas quando forem reavaliadas. Por outro lado, as probabilidades associadas aos piores valores de α terão seus valores reduzidos.

Nem sempre a escolha do parâmetro α a ser utilizado no GRASP para determinado problema será a melhor para todas as instâncias. Por vezes, diferentes instâncias podem gerar melhores resultados com um valor diferente do fixado. A utilização do GRASP Reativo dispensa um tempo de razoável que é consumido no GRASP tradicional para encontrar a calibração deste parâmetro. Enquanto na abordagem padrão o valor definido para α deve balancear o algoritmo entre enfoque guloso e aleatório por meio da realização de vários testes, o reativo faz um auto-ajuste deste valor, dispensando este tempo e tomando o valor de α mais apropriado a cada iteração.

Devido à grande diversificação trazida pelo método e bons resultados encontrados, o GRASP Reativo é uma das mais fortes melhorias aplicadas ao GRASP básico em termos de robustez e qualidade de solução, tendo sido aplicado com sucesso em alguns problemas, como podemos observar nos trabalhos de Boudia, Louly e Prins 2007 e também no trabalho de Alvarez-Valdés, Parreño e Tamarit 2008.

3.2.1.2 Construção com Perturbação de Custos

Uma outra maneira de melhorar a fase construtiva do GRASP é adicionar algum "ruído"aos custos originais do problema de maneira a obter mais aleatoriedade na construção da solução. Segundo Festa e Resende (2011), a perturbação de custos é efetiva em casos

onde a fase construtiva não é muito sensível à estratégias aleatórias padrão, tal como a seleção aleatória de um elemento numa lista restrita de candidatos, esquema discutido na subseção 3.2.1.

A perturbação de custos foi inicialmente proposta por Ribeiro, Uchoa e Werneck (2002) como um dos componentes da fase construtiva de um GRASP híbrido para um problema de Steiner em grafos. O objetivo era gerar diferentes soluções em diferentes iterações a partir da perturbação de custos das arestas durante a fase construtiva, permitindo deste modo uma maior intensificação de busca e maior aleatoriedade.

Esta melhoria proposta para o GRASP tem o objetivo de dinamizar o método construtivo e apresentar diferentes alternativas aos esquemas padrões. O método reativo, perturbação de custos e outros podem ser encontrados com mais detalhes nos trabalhos de Festa e Resende (2011) e Mateus, Resende e Silva (2010). Um caso de aplicação do esquema de perturbação de custos pode ser encontrado no trabalho de Canuto, Resende e Ribeiro (2001).

3.2.2 Busca Local

Assim como a maioria dos métodos determinísticos, as soluções geradas pela fase de construção do GRASP não têm garantias de serem ótimas (FEO; RESENDE, 1995). Porém, o GRASP poderá ser muito beneficiado caso aplique-se uma busca local a fim de melhorar a qualidade da solução gerada.

Segundo Talbi (2009), a busca local pode ser considerada como a mais antiga e simples meta-heurística onde, partindo de uma solução inicial, a cada iteração o procedimento substitui a solução corrente por uma solução vizinha que melhore o valor da função objetivo. Este procedimento é interrompido quando os vizinhos encontrados são piores que a solução corrente, significando que um ótimo local foi atingido. Para grandes vizinhanças, o conjunto de soluções candidatas poderá ser um subconjunto da vizinhança a fim de preservar a performance do método em termos de velocidade. O sucesso do algoritmo de busca local consiste numa escolha adequada de uma estrutura de vizinhança, técnicas eficientes de procura de vizinhos e uma boa solução inicial (FEO; RESENDE, 1995).

Seja $G(V,A)$ um grafo correspondente ao espaço de busca de soluções vizinhas, onde o conjunto de vértices V representa todas as soluções viáveis no espaço de busca e A o conjunto de arestas que representam o movimento conectando soluções vizinhas. No grafo G, uma aresta (i, j) irá conectar quaisquer soluções vizinhas s_i e s_j. Dada uma solução s, o número de arestas associadas será $|N(s)|$, que é o número de vizinhos de s.

A busca local clássica pode apresentar alguns problemas para determinados específicos devido à sua simplicidade e por isso Talbi (2009) propõe algumas estratégias que podem ser aplicadas à busca local a fim de prover um bom esquema de vizinhança, são elas:

- **Melhor vizinho:** nessa estratégia o melhor vizinho, ou seja, aquele que apresenta melhor

custo para a função objetivo, entre todos os presentes no espaço de busca, é selecionado. Neste caso todas os movimentos possíveis entre soluções são testados a fim de trazer a melhor solução. Para problemas em que avaliar uma função é muito custoso ou problemas em que a vizinhança da solução inicial é muito grande, esta busca pode se tornar extremamente custosa, tomando um tempo de processamento muito elevado.

- **Primeira melhoria:** essa estratégia consiste em retornar como resultado da busca local o primeiro vizinho que apresentou uma melhora da função objetivo em relação à solução inicial. Uma vez encontrado um vizinho com melhor custo que a solução inicial, essa solução é imediatamente substituída por seu vizinho e o procedimento é encerrado. Essa estratégia envolve uma exploração parcial do espaço de busca onde, no pior caso, todos os movimentos são explorados e nenhum vizinho melhor que a solução inicial é encontrado.
- **Seleção randômica:** nessa estratégia um vizinho que melhore a função objetivo é aleatoriamente selecionado. Esta escolha aleatória pode ser feita a partir de um ou mais movimentos na solução inicial e os vizinhos buscados podem envolver todo ou apenas uma porção do espaço de busca.

Para as soluções iniciais que são aleatoriamente geradas, a estratégia de primeira melhoria deverá ser uma boa escolha quando se deseja manter o compromisso entre uma busca local rápida que gere soluções de boa de qualidade. Por outro lado, soluções construídas por meio de uma algoritmo guloso geralmente se beneficiam da estratégia de melhores vizinhos, onde a solução inicial será melhorada ao máximo, buscando entre seus vizinhos o melhor ótimo local. O GRASP por sua vez utiliza uma construção aleatória e gulosa, onde neste caso a busca local poderá ser facilmente aplicada com bons resultados tanto nas duas estratégias mencionadas quanto nas estratégia baseada em seleção randômica. O pseudo-código de uma busca local genérica é encontrado no Algoritmo 3 (MATEUS; RESENDE; SILVA, 2010).

Algoritmo 3: BUSCA LOCAL

Entrada: $s_0 \in S$
Saída: Ótimo local t

```
1 início
2 |   t ← s0;
3 |   enquanto existe s ∈ N(t) tal que f(s) < f(t) faça
4 |   |   t ← s;
5 |   fim
6 fim
7 retorna t;
```

Na linha 2 a variável t é inicializada com o valor da solução inicial, que é o parâmetro de entrada do algoritmo. Na linha 4 a solução t recebe uma solução s, vizinha de t, que seja melhor que a solução t corrente. O procedimento se repete enquanto existir solução vizinha melhor que a solução corrente. Finalmente, na linha 7 a melhor solução t, vizinha de s_0,, é retornada.

3.3 Abordagem heurística C1

Outro método de construção de solução para o PLCT pode ser encontrado no trabalho de González *et al*. (2016), no qual os autores propõem uma heurística de construção gulosa denominada C1. Essa heurística, desenvolvida para resolução do PLCT, baseia-se no uso do limite superior para o problema de fluxo máximo.

Deste modo, dada uma rede, para cada par $k \in W$, sendo W o conjunto de pares OD, o método verifica qual dos extremos (o_k ou d_k) possui o menor número de arestas incidentes. Definido o extremo com menos número de arestas, todas as suas arestas são adicionadas a solução posicionando um contador em cada uma dessas arestas. O processo se repete até que todos os pares $k \in W$ sejam testados. O Algoritmo 4 apresenta o pseudo-código da heurística de construção C1.

Algoritmo 4: HEURÍSTICA C1

```
   Saída: Solução s
 1 início
 2 |   s ← ∅;
 3 |   para cada k ∈ W faça
 4 |   |   se #N(o_k) ≤ #N(d_k) então
 5 |   |   |   s ← s ∪ N(o_k);
 6 |   |   senão
 7 |   |   |   s ← s ∪ N(d_k);
 8 |   |   fim
 9 |   fim
10 fim
11 retorna s;
```

Neste algoritmo, $N(.)$ representa o conjunto de arestas incidentes ao vértice passado como argumento, enquanto $\#N(.)$ representa a quantidade de arestas desse conjunto. A instrução $s \leftarrow s \cup N(.)$ representa a adição à solução s de todas as arestas pertencentes ao conjunto $N(.)$. Finalmente, o_k e d_k representam o vértice de origem e destino do par Origem-Destino k, respectivamente.

4 METODOLOGIA

Este trabalho tem por objetivo propor a utilização da meta-heurística GRASP na resolução do PLCT. Para tanto, algumas definições precisam ser dadas a fim de adaptar o algoritmo ao problema em questão. A escolha dos parâmetros, método de construção gulosa e aleatória e esquemas de busca local são componentes do GRASP que precisam ser ajustadas e adaptadas ao problema de maneira que a aplicação da meta-heurística se torne viável.

Nas próximas seções estas definições são fornecidas bem como uma discussão sobre a escolha de cada uma delas.

4.1 Parâmetros de Entrada do GRASP

Como discutido na seção 3, dois parâmetros essenciais ao GRASP devem ser definidos. O primeiro diz respeito ao número de iterações, enquanto o segundo controla o comportamento da exploração do espaço de busca.

Neste trabalho, para a definição do número de iterações, utilizaremos um número inteiro positivo fixo para toda e qualquer instância. Este número diz respeito à quantidade de soluções que serão geradas ao fim da execução do algoritmo GRASP. Quanto maior o número de iterações, mais oportunidades o GRASP terá de explorar ainda mais o espaço de busca e melhores soluções podem ser encontradas, uma vez que o elemento de aleatoriedade garante que diferentes soluções sejam geradas a cada iteração. Porém, um número elevado de iterações pode também causar um elevado custo computacional. A definição deste parâmetro deve levar em consideração um balanceamento entre exploração do espaço de busca e o tempo de execução total do algoritmo.

O segundo parâmetro desempenha um importante papel para a meta-heurística. Denominado α, este regula o enfoque da estratégia que deve variar entre mais gulosa ou mais aleatória, impactando diretamente na qualidade das soluções geradas. Por este motivo, a performance do GRASP é muito sensitiva a este parâmetro. Partindo disto, diferentes estratégias de inicialização deste parâmetro são consideradas, entre elas: inicialização estática, inicialização dinâmica e inicialização adaptativa (TALBI, 2009).

- **Inicialização estática:** aqui o parâmetro é definido estaticamente antes do início da execução da meta-heurística e esse valor é tomado para a construção da LRC em todas as iterações até o fim da execução do algoritmo.
- **Inicialização dinâmica:** nessa estratégia o parâmetro é inicializado randomicamente a partir de uma lista de possíveis valores para α a cada iteração.
- **Inicialização adaptativa:** nesse tipo de inicialização, o valor de α é auto-ajustável a cada iteração e leva em consideração os custos de soluções obtidas em soluções anteriores de

acordo com o α escolhido. Essa estratégia foi discutida em detalhes na subseção 3.2.1.1

O modelo aqui proposto apresenta α como um parâmetro de inicialização auto-ajustável. Seu valor é tomado, a cada iteração, como a melhor escolha baseada nas soluções anteriormente geradas. Entre as estratégias apresentadas, essa foi a que melhor se adaptou ao problema e permitiu maior variabilidade de soluções na exploração do espaço de busca.

4.2 Algoritmo de Construção Gulosa

O próximo passo para a utilização da meta-heurística consiste em determinar uma função gulosa que será utilizada para avaliar e ordenar uma lista de candidatos que irão compor a solução. Desta, um subconjunto denominado LRC (Lista Restrita de Candidatos) é gerado. A LRC é componente chave do GRASP, uma vez que esta representa o aspecto probabilístico do método (TALBI, 2009).

A qualidade de um elemento candidato é determinada por sua contribuição, naquele ponto, para o custo da solução sendo construída (MATEUS; RESENDE; SILVA, 2010). Isso significa dizer que a função gulosa deverá ser adaptativa e, além disto, ser capaz de avaliar, segundo um critério guloso, bons candidatos a compor a solução. A definição de um critério guloso pode variar de acordo com o problema e suas características. Portanto, a definição da função gulosa depende exclusivamente da maneira que se pretende avaliar os candidatos e do problema que está sendo tratado.

Uma função gulosa e aleatória define um valor, aqui denominado grau de adaptação, para uma dada aresta. A cada iteração, durante a construção gulosa e aleatória de uma solução, o algoritmo irá avaliar o grau de adaptação de cada uma das arestas candidatas. Para aquelas arestas que não entraram na solução, seu grau de adaptação medido pela função irá ser alterado em cada uma dessas iterações, por este motivo nossa função é dita adaptativa. Dentre todas as arestas, um percentual das arestas mais promissoras é selecionada com base no parâmetro α. Neste momento, contemplamos no algoritmo de construção gulosa o critério de aleatoriedade.

Para o problema tratado neste trabalho, as arestas candidatas serão avaliadas pela função gulosa $f(x_{ij})$, onde x_{ij} é uma aresta que liga os vértices i e j, definida na Equação 4.1. Desta equação, temos que C é um subconjunto das arestas A, $g(k)$ uma função que determina o grau do vértice k, ϕ_k um parâmetro binário que determina se a aresta k pertence ao conjunto dos pares POD e δ um valor fixo para o ganho da aresta.

$$f(x_{ij}) = \sum_{k}^{\{i,j\}} [g(k) + (\phi_k \times \delta)] \tag{4.1}$$

$$x \in C, \quad \phi \in \{0,1\} \tag{4.2}$$

A função gulosa $f(x_{ij})$ tem por objetivo selecionar boas arestas candidatas com base em algum critério guloso. Para o nosso problema, o critério para seleção das arestas leva em consideração o grau de seus vértices de origem e destino. Quanto maior o grau dos vértices de origem e destino de uma dada aresta, maior será sua adaptação medida pela função. Esta medida tem por objetivo selecionar arestas que possam servir como "pontes" (FEOFILOFF, 2018). Além disto, quando um de seus vértices, seja origem ou destino, pertence ao conjunto W de pares OD, essa aresta é bonificada.

A fim de diversificar a seleção de arestas, foi adicionada à função gulosa uma perturbação no grau de adaptação da aresta definida na seção 3.2.1.2. Durante a avaliação da aresta pela função gulosa $f(x)$, um custo c é adicionado ao valor do seu grau de adaptação caso esta aresta seja escolhida dada uma probabilidade s. Esse mecanismo é incorporado à função gulosa para permitir uma maior variedade de soluções geradas pela meta-heurística. Portanto, a equação pode ser definida como

$$f(x_{ij}) = \sum_{k}^{\{i,j\}} [g(k) + (\phi_k \times \delta)] + [c \times \rho_{ij}] \tag{4.3}$$

$$x \in C, \quad \phi \in \{0,1\}, \quad \rho_{ij} \in \{0,1\} \tag{4.4}$$

sendo ρ um parâmetro binário que assume o valor 1 quando a aresta x_{ij} deve receber uma perturbação com custo c e 0 caso contrário.

Deste modo, a execução da fase de construção dar-se-á da seguinte maneira: todas as arestas serão avaliadas pela função gulosa acrescentada de uma perturbação de custo; destas, apenas um percentual α irá compor LRC; aleatoriamente seleciona-se uma aresta de LRC para entrar na solução, ou seja, a aresta que receberá um contador; a aresta escolhida sairá do conjunto C de candidatos; o processo então se repete até que uma solução viável seja construída.

4.3 Busca Local para o PLCT

O processo de definição do método de exploração do espaços de soluções pela busca local é também determinado exclusivamente pelas características do problema. Para o PLCT, a estrutura de vizinhança é aquela que, movimentando-se uma aresta (seja retirando ou incluindo) temos a formação de uma nova solução viável. Como nosso problema é de minimização, a busca local deverá procurar por soluções parecidas com a atual retirando-se uma aresta aleatória da solução em que busca-se melhorar.

Ao efetuar movimentações na solução deveremos preservar seu aspecto de viabilidade para o problema. Portanto, a cada nova busca local sobre uma solução, o valor da Função Objetivo (FO) é recalculado para esta solução e sua viabilidade é checada. Apenas soluções viáveis de qualidade melhor que a solução atual são procuradas pela busca local.

Uma proposta de busca local para o PLCT é escolher uma aresta ao acaso e retirá-la da solução. Após a retirada, deveremos testar se a nossa solução continua viável, ou seja, verificar se todos os caminhos possíveis entre cada par *OD* tem pelo menos um contador em uma de suas arestas. Caso a viabilidade seja verificada, uma solução melhor foi encontrada pela busca local. Sobre a solução encontrada é aplicada uma nova busca local até que não se encontre uma melhora de solução.

Uma outra proposta de busca local para o PLCT envolve um novo esquema de busca, ainda não proposto na literatura, observando características na construção da solução efetuada pelo GRASP. Observou-se durante os testes que muitas vezes a solução construída pelo algoritmo era tal que algumas das arestas poderiam ser facilmente removidas sem perder sua viabilidade. A cada remoção de aresta, um teste era efetuado para garantir que esta continuaria viável. Porém, quando a busca local é grande o suficiente para remover muitas arestas, cada um desses testes torna-se então cada vez mais custoso, inviabilizando assim uma busca com muitas retiradas de aresta, uma vez que o tempo computacional despendido seria extremamente grande. Dito isto, uma maneira conveniente de busca local que foi aplicada com sucesso ao GRASP envolveu uma exploração de vizinhanças com n movimentos de retiradas de arestas simultâneos.

Algoritmo 5: BUSCA LOCAL APLICADA AO PLCT

Entrada: $s_0 \in S$
Saída: Ótimo local t

```
1  início
2      t ← s_0;
3      n ← β × s_0.size();
4      l ← lista_vazia;
5      enquanto n > 0 faça
6          l ← removeArestasAleatorias(t, n);
7          se t é viável então
8              continue;
9          senão
10             insereArestas(t, l);
11             n ← n − (ε × n);
12         fim
13     fim
14 fim
15 retorna t;
```

Inicialmente n arestas aleatórias são removidas; um teste de viabilidade da nova solução é então efetuado; caso esta solução continue viável, a busca continua na vizinhança removendo-se então mais n arestas. A qualquer momento esta solução poderia tornar-se inviável, então as últimas n arestas removidas são retornadas à solução. O valor de n é decrementado e os passos se repetem. A busca é encerrada quando n é igual a 0 e a solução final com as arestas removidas da solução inicial é então retornada. O Algoritmo 5 mostra a busca local desenvolvida para o

GRASP aplicado ao PLCT.

Entre as linhas 2 e 4 são efetuadas as inicializações necessárias à execução da busca, no qual t é um vetor que inicialmente corresponde à solução de entrada que se deseja melhorar; n um valor inteiro que controla o número de buscas locais e inicialmente corresponde a uma porcentagem β do tamanho da solução inicial s_0; e finalmente l que corresponde a uma lista de arestas, inicialmente vazia, que guarda as arestas removidas da solução t. Na linha 6, n arestas aleatórias são removidas de t e guardadas em l. Na linha 7 a viabilidade de t é testada e, caso seja viável, a execução volta para a linha 6 de acordo com a instrução na linha 8. Quando a solução torna-se inviável, as últimas arestas removidas em l são retornadas à solução na linha 11 e o valor de n é então decrementado na linha 12. Ainda na linha 12, um parâmetro ε, definido entre 0 e 1, é inserido de modo a garantir que o valor de n seja decrementado gradualmente a cada iteração. Por fim, na linha 16 a solução t é retornada.

5 RESULTADOS COMPUTACIONAIS

Neste capítulo, são apresentados e discutidos os experimentos computacionais obtidos pela meta-heurística GRASP [3] e seus resultados são comparados a outras abordagens da literatura.

5.1 Ambiente de execução e instâncias utilizadas

Os métodos abordados nesta seção foram todos implementados na linguagem C++ e compilador GCC versão 6.3 com auxílio de uma biblioteca para modelagem e otimização eficiente em grafos chamada *LEMON* (do inglês *Library for Efficient Modeling and Optimization in Networks*) e disponibilizada gratuitamente no *website* da biblioteca (EGRES, 2018). Os experimentos computacionais foram executados em um computador Intel Core^{TM}i7 CPU 6700k@4.60 GHz com 16GB de RAM, utilizando o sistema operacional Ubuntu 16.04.

Tabela 1 – Características das instâncias utilizadas.

Estado	Vertices	Arestas	M
AC	61	84	20
AL	169	219	97
AM	74	77	37
AP	52	77	13
BA	812	1113	395
CE	401	612	177
ES	283	394	75
GOeDF	799	1165	241
MA	257	355	163
MG	1474	1917	803
MS	343	497	76
MT	711	1069	140
PA	289	370	122
PB	384	480	213
PE	362	472	172
PI	405	550	212
PR	780	1083	381
RJ	502	721	86
RN	333	433	160
RO	185	258	50
RR	75	97	13
RS	675	859	391
SC	481	604	266
SE	183	248	74
SP	1280	1683	606
TO	372	524	134

As instâncias testadas contém dados reais a partir dos 26 estados brasileiros, nas quais os pares *OD* foram definidos a partir dos municípios de cada estado e cada aresta representa um segmento de rodovia federal ou estadual presente no estado de cada instância. Estes dados foram extraídos de uma base georreferenciada do ano de 2015 disponibilizada pelo Departamento Nacional de Infraestrutura de Transportes (GONZÁLEZ et al., 2017). A Tabela 1 apresenta as principais características das 26 instâncias utilizadas, sendo M o número de municípios que constituem o conjunto W. Como cada um dos municípios são origem ou destino para todos os outros, o número de pares *OD* pode ser calculado como $|W| = \frac{M \times (M-1)}{2}$. A maior instância é a de Minas Gerais (MG) que possui 1917 arestas, 1474 vértices, 803 municípios e pouco mais de 322 mil pares *OD*.

Um exemplo dos dados de entrada para a execução dos algoritmos é ilustrado no Anexo A, no qual uma instância é criada a partir de 18 vértices, dos quais 4 são *OD*, e 25 arestas representando uma rede de transporte.

5.2 Resultados da meta-heurística GRASP

Para a calibração do GRASP, foram realizados experimentos com três instâncias de médio porte (AL, MS e RS) propostas no trabalho de Mauri *et al*. (2017) por serem, segundo o autor, representativas o suficiente. Instâncias menores poderiam levar a incorretas interpretações, enquanto instâncias de grande porte levariam demasiado tempo para serem testadas. A Tabela 2 apresenta os valores definidos após o processo de calibragem. Com os parâmetros definidos, cada instância foi executada 5 vezes e cada execução com 250 iterações.

Tabela 2 – Parâmetros do GRASP.

Parâmetro	Descrição	Valor final	Valores testados
β	Parâmetro para a busca local.	0,3	0,2 e 0,3
δ	Ganho fixo da aresta.	100	10, 20, 50, 100
ψ	Conjunto de alphas iniciais.	{0,001, 0,1, 0,2, 0,3}	{0,001, 0,1, 0,2, 0,3, 0,4, 0,5, 0,6, 0,7} e {0,001, 0,1, 0,2, 0,3}
c	Custo aleatório adicional.	{1, 2, 3, 4, 5, 6, 7, 8, 9}	{1, 2, 3, 4, 5, 6, 7, 8, 9} e {10, 20, 30, 40, 50, 60, 70, 80, 90}

Na Tabela 3 é possível observar os resultados obtidos com o GRASP utilizando as técnicas desenvolvidas na Seção 4. Apesar de não alcançar o ótimo em nenhuma das instâncias, como será discutido mais adiante, o GRASP apresentou resultados aproximados em tempo de execução viável. A primeira coluna indica o nome de cada instância, a segunda coluna indica a melhor solução encontrada, enquanto a terceira e quarta colunas mostram a solução média e tempo médio respectivamente. Estas duas últimas colunas são necessárias visto que

cada execução do GRASP é aleatória, portanto diferentes soluções podem ser encontradas com diferentes sementes. Podemos observar que o maior tempo de execução foi para a instância de MG com um tempo de execução total de 9.693,77 segundos.

Tabela 3 – Resultados das execuções do GRASP.

Instância	Melhor Sol.	Média Sol.	T(s) Médio
AC	36	37,00	0,48
AL	169	171,00	12,41
AM	46	47,00	2,41
AP	29	29,80	0,24
BA	800	805,00	1115,38
CE	424	426,80	129,58
ES	185	187,20	19,34
GOeDF	580	581,60	522,08
MA	310	312,80	53,03
MG	1423	1428,25	9693,77
MS	199	202,80	17,09
MT	355	357,20	140,99
PA	229	232,40	26,56
PB	368	369,80	53,83
PE	316	321,80	36,65
PI	405	409,00	66,22
PR	764	765,20	481,12
RJ	200	201,20	39,44
RN	287	291,60	30,39
RO	109	111,80	3,64
RR	21	21,40	0,27
RS	676	679,20	335,57
SC	476	478,80	215,60
SE	145	147,40	4,99
SP	1062	1066,40	1766,76
TO	295	298,20	29,07

5.3 Comparações com estratégias da literatura

A fim de mensurar o desempenho do GRASP, na Tabela 4 são apresentados os resultados do GRASP, da heurística C1 e da abordagem exata da literatura *Branch-and-cut* (GONZÁLEZ et al., 2017). Os algoritmos comparados nessa seção, em seus trabalhos originais, foram executados no mesmo ambiente mencionado anteriormente para as mesmas instâncias. Para a execução da heurística C1 nenhum ajuste de parâmetro foi necessário, enquanto o único parâmetro ajustado para o *Branch-and-cut* diz respeito ao tempo limite de execução do algoritmo, definido em 3600 segundos. Em negrito, destacamos as soluções ótimas encontradas.

Na Tabela 4 podemos observar que apenas 8, das 26 instâncias testadas, possuem solução ótima conhecida, nas quais todas foram alcançadas somente pelo algoritmo exato *Branch-and-cut*. Com relação à qualidade das soluções, os resultados mostram que, para todas as instâncias, o GRASP obteve melhor desempenho quando comparado à heurística C1. Porém, a abordagem teve pior resultado quando comparado ao algoritmo exato. Com relação ao tempo de execução, com exceção das instâncias em que o ótimo foi alcançado, para todas as outras instâncias o algoritmo *Branch-and-cut* excedeu o tempo limite de execução. O GRASP, entretanto, teve certa dificuldade apenas para a instância de MG, sendo esta a única para a qual o tempo de execução excedeu 3600 segundos.

Tabela 4 – Comparativo entre diferentes estratégias para o PLCT.

	GRASP			**C1**		**Branch-and-cut**	
Instância	**Melhor Sol.**	**Média Sol.**	**T (s) Médio**	**Sol**	**Tempo (s)**	**Sol**	**Tempo (s)**
AC	36	37,00	0,48	40	0,001	***30***	0,23
AL	169	171,00	12,41	178	0,002	***137***	1,20
AM	46	47,00	2,41	57	0,001	***39***	0,10
AP	29	29,80	0,24	31	0,001	***22***	0,05
BA	800	805,00	1115,38	822	0,188	630	3600
CE	424	426,80	129,58	438	0,011	331	3600
ES	185	187,20	19,34	205	0,001	144	3600
GOeDF	580	581,60	522,08	593	0,031	483	3600
MA	310	312,80	53,03	321	0,010	***250***	3,51
MG	1423	1428,25	9693,77	1468	3,193	1133	3600
MS	199	202,80	17,09	210	0,002	152	3600
MT	355	357,20	140,99	369	0,008	313	3600
PA	229	232,40	26,56	247	0,003	174	3600
PB	368	369,80	53,83	382	0,016	296	3600
PE	316	321,80	36,65	332	0,008	252	3600
PI	405	409,00	66,22	418	0,016	316	3600
PR	764	765,20	481,12	790	0,135	599	3600
RJ	200	201,20	39,44	210	0,001	169	3600
RN	287	291,60	30,39	301	0,009	233	3600
RO	109	111,80	3,64	119	0,001	***88***	604,03
RR	21	21,40	0,27	25	0,001	***19***	0,01
RS	676	679,20	335,57	699	0,185	545	3600
SC	476	478,80	215,60	486	0,049	371	3600
SE	145	147,40	4,99	151	0,001	***112***	35,73
SP	1062	1066,40	1766,76	1095	1,026	885	3600
TO	295	298,20	29,07	299	0,005	232	3600

6 CONCLUSÃO

Este trabalho apresentou uma nova abordagem para resolução do Problema de Localização de Contadores de Tráfego (PLCT). O PLCT é um problema de Otimização Combinatória do tipo NP-Difícil e resolvê-lo requer grande esforço, uma vez que a complexidade cresce exponencialmente com respeito ao tamanho da rede.

A fim de resolver o PLCT, foi proposta uma aplicação da meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*), uma abordagem robusta e intuitiva que obteve êxito na aplicação para diversos outros problemas de otimização combinatória. Uma função gulosa e aleatória foi desenvolvida baseada nas características do PLCT para construção de uma solução inicial e um novo esquema de busca local foi sugerido e implementado com base nos experimentos realizados.

A validação da abordagem foi realizada por meio de experimentos computacionais em 26 instâncias que utilizam dados reais baseados nos estados brasileiros. Os resultados mostraram que o GRASP é capaz de construir soluções viáveis para todas as instâncias em tempos melhores que o algoritmo exato e de melhor qualidade que a heurística comparada.

Como proposta de trabalhos futuros, pretende-se utilizar da característica do GRASP de gerar uma nova solução viável a cada iteração. De acordo com os resultados obtidos, percebeu-se que diferentes soluções viáveis e de boa qualidade são alcançadas muito rapidamente, neste sentido a aplicação de estratégias que utilizem o GRASP como gerador de soluções iniciais, ou mesmo a incorporação de um método que utilize memória de longo prazo, poderá aproveitar essa forte vantagem do GRASP.

O GRASP como gerador de soluções iniciais pode ser incorporado a um Algoritmo Genético (AG), por exemplo. Neste tipo de aplicação, a população inicial do AG é formada a partir de diversas iterações da meta-heurística GRASP. Uma outra proposta de melhoria para o GRASP consiste na hibridização do método com técnicas de *Data Mining*. O uso das soluções encontradas após um significante número de iterações do GRASP, como memória de longo prazo, pode ser explorado por meio de técnicas de mineração de dados a fim de descobrir padrões de soluções. Estes padrões podem apresentar características próximas do ótimo e servir como guia para as iterações seguintes da meta-heurística (BARBALHO et al., 2013).

Outra possível melhoria para o GRASP seria a utilização de outros esquemas de buscas locais, como a utilização de um VNS (do inglês *Variable Neighborhood Search*) (MLADENOVIĆ; HANSEN, 1997) a fim de diversificar a busca. Uma nova função gulosa, que avalie outros aspectos das arestas e não só o grau de incidência de seus vértices, também pode ser uma opção.

REFERÊNCIAS

ALVAREZ-VALDÉS, R.; PARREÑO, F.; TAMARIT, J. M. Reactive grasp for the strip-packing problem. **Computers & Operations Research**, Elsevier, v. 35, n. 4, p. 1065–1083, 2008.

BARBALHO, H. et al. A hybrid data mining grasp with path-relinking. **Computers & Operations Research**, Elsevier, v. 40, n. 12, p. 3159–3173, 2013.

BOUDIA, M.; LOULY, M. A. O.; PRINS, C. A reactive grasp and path relinking for a combined production–distribution problem. **Computers & Operations Research**, Elsevier, v. 34, n. 11, p. 3402–3419, 2007.

BOUSSAÏD, I.; LEPAGNOT, J.; SIARRY, P. A survey on optimization metaheuristics. **Information Sciences**, Elsevier, v. 237, p. 82–117, 2013.

CANUTO, S. A.; RESENDE, M. G.; RIBEIRO, C. C. Local search with perturbations for the prize-collecting steiner tree problem in graphs. **Networks**, Wiley Online Library, v. 38, n. 1, p. 50–58, 2001.

DNIT. **Plano Nacional de Contagem de Tráfego**. 2018. Disponível em: <http://www.dnit.gov.br/planejamento-e-pesquisa/planejamento/contagem-de-trafego>.

EGRES. **LEMON Graph Library**. 2018. Disponível em: <http://lemon.cs.elte.hu/trac/lemon>.

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. **Journal of global optimization**, Springer, v. 6, n. 2, p. 109–133, 1995.

FEOFILOFF, P. **Pontes em grafos não-dirigidos**. 2018. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/bridges.html>.

FESTA, P.; RESENDE, M. G. An annotated bibliography of grasp–part i: Algorithms. **International Transactions in Operational Research**, Wiley Online Library, v. 16, n. 1, p. 1–24, 2009.

FESTA, P.; RESENDE, M. G. An annotated bibliography of grasp–part ii: Applications. **International Transactions in Operational Research**, Wiley Online Library, v. 16, n. 2, p. 131–172, 2009.

FESTA, P.; RESENDE, M. G. Grasp: basic components and enhancements. **Telecommunication Systems**, Springer, v. 46, n. 3, p. 253–271, 2011.

GARCÍA-ARCHILLA, B. et al. Grasp algorithms for the robust railway network design problem. **Journal of Heuristics**, Springer, v. 19, n. 2, p. 399–422, 2013.

GONZÁLEZ, P. H. et al. Heurísticas para o problema de localização de contadores de tráfego em redes de transporte. In: **Anais do XXX ANPET-Congresso de Pesquisa e Ensino em Transportes**. [S.l.: s.n.], 2016.

GONZÁLEZ, P. H. et al. Um algoritmo branch-and-cut para o problema de localização ótima de contadores de tráfego em redes de transporte. **XLIX Simpósio Brasileiro de Pesquisa Operacional**, 2017.

HOLLAND, J. H. **Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence**. [S.l.]: MIT press, 1992.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **science**, American Association for the Advancement of Science, v. 220, n. 4598, p. 671–680, 1983.

LIMA, I. L. et al. Grasp aplicado ao problema de alocaç ão de berços em terminais portuários graneleiros. Jornada de Informática do Maranhão, 2014.

MATEUS, G. R.; RESENDE, M. G.; SILVA, R. M. Grasp: Procedimentos de busca gulosos, aleatórios e adaptativos. **2a Escola Luso-Brasileira de Computação Evolutiva**, 2010.

MAURI, G. R. et al. Meta-heurística cs para o problema de localização de contadores de tráfego em redes de transporte. **XLIX Simpósio Brasileiro de Pesquisa Operacional**, 2017.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. **Computers & operations research**, Elsevier, v. 24, n. 11, p. 1097–1100, 1997.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial optimization: algorithms and complexity**. [S.l.]: Courier Corporation, 1998.

PRAIS, M.; RIBEIRO, C. C. Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. **INFORMS Journal on Computing**, INFORMS, v. 12, n. 3, p. 164–176, 2000.

RESENDE, M. G.; FEO, T. A. A grasp for satisfiability. In: CITESEER. **CLIQUES, COLORING, AND SATISFIABILITY: THE SECOND DIMACS IMPLEMENTATION CHALLENGE, VOLUME 26 OF DIMACS SERIES ON DISCRETE MATHEMATICS AND THEORETICAL COMPUTER SCIENCE**. [S.l.], 1996.

RIBEIRO, C. C.; HANSEN, P. **Essays and surveys in metaheuristics**. [S.l.]: Springer Science & Business Media, 2012. v. 15.

RIBEIRO, C. C.; UCHOA, E.; WERNECK, R. F. A hybrid grasp with perturbations for the steiner problem in graphs. **INFORMS Journal on Computing**, INFORMS, v. 14, n. 3, p. 228–246, 2002.

TALBI, E.-G. **Metaheuristics: from design to implementation**. [S.l.]: John Wiley & Sons, 2009. v. 74.

YANG, H. et al. Estimation of origin-destination matrices from link traffic counts on congested networks. **Transportation Research Part B: Methodological**, Elsevier, v. 26, n. 6, p. 417–434, 1992.

ANEXO A – EXEMPLO DE INSTÂNCIA PLCT

```
NbNode  NbEdge  POD
18  25  4
1
2
5
8
1  1  12
2  1  11
3  1  3
4  2  3
5  2  7
6  3  4
7  3  5
8  3  6
9  3  11
10  4  5
11  4  9
12  5  9
13  5  10
14  6  7
15  6  8
16  8  10
17  9  16
18  11  12
19  11  17
20  12  13
21  13  14
22  14  18
23  14  15
24  15  16
25  16  17
```