EAComp
Encontro Acadêmico de Computação

VIII Encontro Acadêmico de Computação - UFMA
Copyright (c) EAComp. ISSN: 2594-486X

Meta-heurística GRASP aplicada ao problema do roteamento de veículos capacitado: um estudo de caso com a empresa Loggi

Mayrla Jansen[1], **Glaubos Clímaco**[1]

[1]Universidade Federal do Maranhão (UFMA)
Av. dos Portugueses, 1966 - Vila Bacanga, Sao Luís - MA, 65080-805

mayrla.jansen@discente.ufma.br, francisco.glaubos@ufma.br

Resumo. *Neste trabalho, é proposto um método para a resolução do problema de roteamento de veículos capacitado (PRVC). O PRVC é uma extensão do problema de roteamento de veículos, que por sua vez deriva do clássico problema do caixeiro viajante. Para a resolução desse problema, é proposta uma heurística baseada na meta-heurística GRASP no intuito de se obter soluções de boa qualidade em um tempo computacional razoável. O GRASP trata-se de um algoritmo iterativo composto por uma fase de construção de uma solução viável, e outra fase na qual ocorre uma busca local promovendo um refinamento dessa solução. Para a validação da proposta, foram realizados experimentos computacionais em instâncias geradas a partir de uma base de dados real, disponibilizada pela empresa Loggi.*
Palavras-chave*: GRASP, meta-heurística, PRVC, Loggi.*

Abstract. *In this work, a method for solving the capacitated vehicle routing problem (CVRP) is proposed. CVRP is an extension of the vehicle routing problem, which in turn derives from the classic traveling salesman problem. To solve this problem, a heuristic based on the GRASP meta-heuristic is proposed in order to obtain good quality solutions in a reasonable computational time. GRASP is an iterative algorithm composed of a construction phase for an initial solution, and another phase in which a local search takes place, promoting a refinement of this solution. To validate the proposal, computational experiments were carried out on instances generated from a real database, made available by the company Loggi.*
Key-words*: GRASP, meta-heuristic, CVRP, Loggi.*

1. Introdução

Com o avanço das demandas por bens e serviços, aliada ao grande crescimento da competitividade presente no mercado, se torna cada vez mais notório que os custos que antes poderiam ser relevados agora se tornam de fundamental importância para se tornarem objetos de estudo e diminuição de valores. Nesse ponto, em especial, as despesas com as operações. Essas devem ser cada vez mais pensadas do ponto de vista logístico, para que o produto final obtenha o menor valor de custo possível. Dessa forma, a utilização de Pesquisa Operacional para este fim, se torna de grande valia, visto que a aplicação de suas técnicas buscam o máximo desempenho operacional, vantagens estratégicas e contabilizam o impacto financeiro [de Araújo et al. 2018].

Nesse contexto, surge o problema de roteamento de veículos (PRV). O PRV é um problema de otimização combinatória que pergunta "Qual é o conjunto ideal de rotas para uma frota de veículos percorrer a fim de realizar entregas a um determinado conjunto de clientes?". O PRV tem muitas aplicações diretas na indústria, e o uso de *softwares* de otimização pode proporcionar economia de 5% para uma empresa [Hasle et al. 2007], já que o transporte é geralmente um componente significativo do custo de um produto (10%) [Rodrigue et al. 2016].

O PRV diz respeito ao serviço de uma empresa de entrega. As entregas são realizadas a partir de um ou mais depósitos que têm um determinado conjunto de veículos operados por um conjunto de motoristas que podem se deslocar em uma determinada rede rodoviária para um conjunto de clientes. O objetivo do PRV é determinar um conjunto de rotas, S, (uma rota para cada veículo que deve começar e terminar em seu próprio depósito) de forma que todos os requisitos dos clientes e restrições operacionais sejam atendidos e o custo global de transporte seja minimizado. Este custo pode ser monetário, distância ou outro [Toth e Vigo 2002].

A rede de estradas pode ser descrita usando um grafo onde os arcos são estradas e vértices são junções entre eles. Os arcos podem ser direcionados ou não direcionados devido à possível presença de ruas de mão única ou custos diferentes em cada direção. Cada arco possui um custo associado que geralmente é sua duração ou tempo de viagem, que pode depender do tipo de veículo [Toth e Vigo 2002]. Neste trabalho é abordado o problema de roteamento de veículos capacitado (PRVC). O PRVC é uma variante do PRV na qual os veículos possuem uma capacidade de carga limitada das mercadorias que devem ser entregues (ver Figura 1).

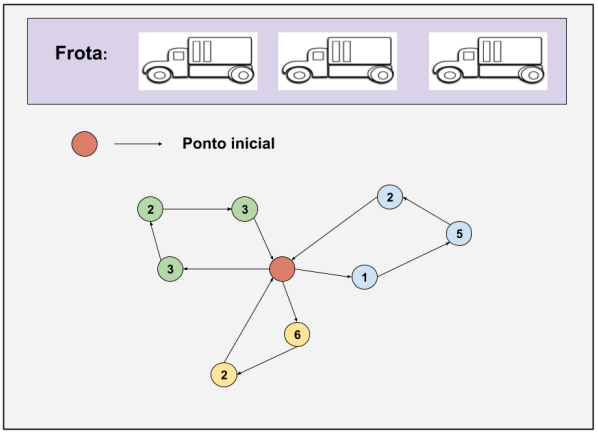

Figura 1. Representação em grafo de uma solução para o PRVC para 8 vértices

Determinar a solução ótima para PRVC é NP-difícil [Toth e Vigo 2002], então o tamanho dos problemas que podem ser resolvidos, de forma otimizada, usando programação matemática ou otimização combinatória pode ser limitado. Portanto, os solucionadores comerciais tendem a usar heurísticas devido ao tamanho e à frequência dos PRVC do mundo real que precisam resolver. Esse fato motivou os autores deste trabalho a proporem um método baseado na metaheurística GRASP (*greedy randomized adaptive search prodecures)* para a resolução do problema, a partir de dados reais da empresa

Loggi [LOGGI 2021].

O restante deste artigo é organizado da seguinte maneira. A Seção 2 descreve os principais trabalhos relacionados ao PRVC. Na Seção 3 é apresentado o desenvolvimento do algoritmo GRASP, assim como os algoritmos utilizados em suas fases: guloso e de busca local. A Seção 4, descreve a instanciação dos dados e a análise feita com os resultados obtidos pelo algoritmo GRASP. Finalmente, na Seção 5 é apresentada a conclusão deste trabalho.

2. Trabalhos relacionados

O PRVC é amplamente estudado na literatura, e foi estudado pela primeira vez em um artigo de [Dantzig e Ramser 1959], no qual a primeira abordagem algorítmica foi escrita e aplicada a entregas de gasolina. Esse problema pode ser resolvido por métodos exatos (modelos matemáticos e algoritmos de branch-and-bound) ou abordagens heurísticas. Métodos exatos podem obter soluções ótimas, mas sua complexidade computacional resultará em tempo exponencial quando o tamanho do problema for grande. As abordagens heurísticas podem ser divididas em abordagens heurísticas clássicas e abordagens metaheurísticas.

Entre os procedimentos heurísticos clássicos, o algoritmo construtivo de [Clarke e Wright 1964] (CWS) é provavelmente a heurística mais citada para resolver o PRVC. Muitas variantes e melhorias do CWS foram propostas na literatura. Por exemplo, [Mole e Jameson 1976] generalizaram a definição da função de economia, introduzindo dois parâmetros para controlar o comportamento da economia. Da mesma forma, [Holmes e Parker 1976] desenvolveram um procedimento baseado no algoritmo CWS, utilizando a mesma função de economia, mas introduzindo um esquema de perturbação de solução para evitar rotas de baixa qualidade.

Utilizando heurísticas construtivas como base, as meta-heurísticas se tornaram populares para o PRV durante os anos noventa. Esses procedimentos são capazes de encontrar soluções quase ótimas e até ótimas globais para problemas combinatórios. Alguns exemplos iniciais aplicados para resolver o PRVC são o método *Tabu Route* de [Gendreau et al. 1994] ou o método Boneroute de [Tarantilis e Kiranoudis 2002]. Algoritmos de busca tabu (TS), como os propostos por [Taillard 1993] ou [Toth e Vigo 2003], estão entre as meta-heurísticas mais populares. Por fim, os algoritmos genéticos (AG) também têm desempenhado um papel importante no desenvolvimento de abordagens eficazes para o PRV. Alguns exemplos são os estudos de [Berger e Barkaoui 2003] e [Mester e Bräysy 2007].

É importante ressaltar que, após a revisão bibliográfica para este trabalho, foi observado que apesar de vários trabalhos na literatura relatarem heurísticas/meta-heurísticas interessantes para a resolução do PRVC, poucos trabalhos abordam instâncias ou cenários baseados em dados geográficos reais. Em muitos casos, os autores posicionaram pontos aleatoriamente em um plano cartesiano e adotaram a distância euclidiana entre dois pontos.

3. Meta-heurística grasp

Neste trabalho, é proposto um método de solução baseado na meta-heurística GRASP. GRASP do inglês, *Greedy Randomized Adaptive Search Procedure*, é uma meta-

heurística proposta por [Resende e Ribeiro 2016] e utilizada para resolver problemas complexos e de grande porte. Essa meta-heurística é executada iterativamente, e cada iteração é composta por duas fases: construção e busca local. Uma solução viável S é gerada na fase de construção, e então sua vizinhança é explorada por uma busca local, a fim de encontrar uma solução melhor. Esse processo iterativo é repetido até que um critério de parada seja alcançado, que pode ser um número limitado de iterações permitidas, um número de iterações sem melhoria da solução, entre outros. Por fim, a melhor solução encontrada em todas as iterações é retornada.

O Algoritmo 1 apresenta um pseudo-código do GRASP. Ele recebe como parâmetros o $MaxIterations$ e o α, sendo $MaxIterations$ o número de iterações a serem executadas e o α determina o quão guloso é o algoritmo, variando de zero a um, em que quanto mais próximo de zero mais aleatório e quanto mais próximo de um, mais guloso. Na linha 1, o valor da melhor solução é iniciado com um valor muito alto (∞), em seguida, das linhas 2 a 8, são executadas $MaxIterations$ iterações, no qual uma solução é construída de maneira gulosa e refinada com busca local. A cada iteração é feito um teste que verifica se a solução atual é melhor que a melhor solução, caso positivo, o valor da melhor solução é atualizado.

Algorithm 1 GRASP $(\alpha, MaxIterations)$

```
1: bestSolution ← ∞
2: for i = 0, 1, ..., MaxIterations do
3:     solutionGreedy ← greedySearch(α)
4:     solution ← localSearch(solutionGreedy)
5:     if solution < bestSolution then
6:         bestSolution ← solution
7:     end if
8: end for
9: return bestSolution
```

3.1. Algoritmo de busca gulosa

Para a fase de construção do GRASP, uma busca gulosa foi implementada considerando as seguintes condições do problema: i) um vértice (cliente) deve ser visitado por apenas um único veículo que deve ter capacidade maior ou igual à quantidade de entregas (demanda) desse vértice; ii) todos os veículos da frota partem de um mesmo vértice origem (depósito); e iii) cada vértice deve ser visitado exatamente uma vez.

Diferentemente de um algoritmo guloso tradicional, geralmente aplicado em situações de tomada de decisão, neste trabalho foi implementado uma busca gulosa mais complexa. Tal busca não considera simplesmente a menor distância de um vértice para outro, mas sim uma lista de candidatos restrita (LCR); e para pertencer a essa LCR, o custo incremental da visita de um vértice por determinado veículo deve considerar um valor limite, estipulado pela Equação (1).

$$\boxed{limite = menor\_distancia + \alpha * (maior\_distancia - menor\_distancia)} \quad (1)$$

Nesse caso, a LCR será composta por vértice que ainda não foram visitados e que possuam um custo incremental menor que o valor limite da Equação (1). Mais detalhes sobre a busca gulosa implementada são apresentados no pseudo-código do Algoritmo 2.

Algorithm 2 greedySearch $(\alpha, n\_veiculos, n\_vertices)$

```
S ← ∅                                              ▷ inicializando a solução
for k = 0 até k < n_veiculos do
    LC ← i ∈ V : i é um vértice ainda não visitado
    LRC ← criar_LRC(α, LC, k)                      ▷ a partir da Equação (1)
    v ← random(LRC)                ▷ escolhendo um vértice para ser visitado por k
    S[k] ← S[k] ∪ v
end for
return S
```

No Algoritmo 2, iteramos tanto sobre os veículos quanto sobre os vértices, de modo a obter soluções considerando a quantidade de veículos e suas capacidades. Para cada veículo, inicialmente são considerados todos os vértices ainda não visitados (conjunto LC). Em seguida, é cria uma lista de candidatos restrita (LCR), que é composta pelos vértices que obtêm custo incremental menor ou igual ao *limite* e que a demanda não ultrapassa a capacidade atual do veículo. Após a lista finalizada, aleatoriamente é escolhido um dos vértices de LRC que será incorporado à solução gulosa que está sendo montada. Ao final do laço que percorre todos os veículos, a solução construída é retornada.

3.2. Algoritmo de busca local

Buscas locais, também referidas na literatura como buscas na vizinhança, são procedimentos utilizados para melhorar uma solução viável obtida, por exemplo, através de uma heurística construtiva. Esses métodos são mais genéricos e podem auxiliar uma ampla gama de problemas sem a necessidade de uma compreensão mais profunda do mesmo [Sauer 2007].

Após a fase de construção, já é possível ter uma solução gulosa, contudo, como foi visto anteriormente, essa solução apresenta desvantagens por não considerar o problema como um todo. Com a busca local, é realizado um refinamento na solução obtida, de maneira a melhorar a qualidade da solução construída. Ou seja, com essa fase, pretende-se reduzir o custo das rotas, caso seja possível. O Algoritmo 3 é um pseudo-código da busca local realizada neste estudo de caso.

Para este trabalho a busca local ocorreu da seguinte forma. Para cada veículo e para cada vértice pertencente à rota desse veículo, é verificado se uma troca de ordem de visita entre dois vértices melhora o custo da solução. Caso melhore, a solução refinada da busca local é atualizada. Por fim, a solução $S\_refinada$ é retornada.

4. Experimentos computacionais

Para a realização dos experimentos deste trabalho, a linguagem escolhida foi o python3.8 e o ambiente foi um computador pessoal Intel® Core™ i5 CPU@1.60GHz com 8GB RAM, utilizando o sistema operacional Linux Ubuntu 18.04.5.

Algorithm 3 LocalSearch (S)

```
1: S_refinada ← S
2: for k = 0 até k < n_veiculos do
3:     for v = 0 até v < tamanho(S[k]) − 1 do
4:         custo_k_antes ← custo(S[k])
5:         troca(S[k][v], S[k][v + 1])          ▷ trocando a ordem de visita na rota k
6:         custo_k_depois = custo(S[k])
7:         if custo_k_depois ≥ custo_k_antes then
8:             troca(S[k][v + 1], S[k][v])      ▷ desfazendo a troca
9:         else
10:            S_refinada ← S
11:        end if
12:    end for
13: end for
14: return S_refinada
```

4.1. Criação das instâncias

Os dados referentes aos pontos de entrega (longitude e latitude), quantidade de entrega por pedido e capacidade dos veículos, cedidos pela empresa Loggi para análise e testes, estavam em formato *JSON* [LOGGI 2021], e para fins desse estudo a quantidade de pontos usada foi de 100 e 200. *JSON* é um formato de texto derivado da linguagem Javascript sendo representado por uma coleção de objetos compostos por pares de chave-valor [Silva e Silva Júnior 2018].

Para fins de implementação, optou-se por transformar os dados presentes no JSON, em um novo arquivo no formato `.txt`, tal que, nesse novo arquivo, estaria elencado a quantidade de pontos de entregas, capacidade dos veículos, quantidade de pedidos e por fim, uma matriz que representa a distância de um ponto para os demais como mostrado na Figura (2). Esse arquivo será o grafo, usado na entrada para o algoritmo GRASP, pois nele conterá todas as informações da instância.

A capacidade dos veículos foi dada com base no total das demandas. Considerando K o número de veículos e Q a capacidade dos veículos, a razão (soma de todas as demandas)$/K \times Q$ deve ser o mais próximo possível de 1, tendo como consequência o número mínimo de veículos para se obter uma solução viável.

Para construir a matriz de distância entre os pontos, foi empregada a Distance Matrix API do Google [GOOGLE 2021]. O Algoritmo (4) mostra o pseudo-código da utilização dessa API neste estudo de caso. A API é conectada com o projeto (linha 1) e necessita dos seguintes parâmetros: longitude e latitude do ponto X e do ponto Y, onde X é o ponto de origem e Y é o ponto de destino (linha 2); e a partir dessas coordenadas, a API retorna um conjunto de dados sobre esses pontos. Para este trabalho foi necessário apenas registrar a distância real entre esses pontos (linhas 3 e 4).

Vale ressaltar que o retorno da API foi padronizado para a unidade de medida em quilômetros, de modo a facilitar a manipulação dos dados, visto que assim, todos estão na mesma unidade de medida. E a distância de um ponto para ele mesmo, foi padronizada em 10000.0, com o intuito de representar uma distância excessiva para que sempre fosse excluída nos algoritmos de construção de rotas, mostrando que não se deve ir de um ponto

Copyright (c) EAComp. ISSN: 2594-486X

instancias_5.txt

src > matriz_distancias > arquivos > distancia > instancias_5.txt

```
5 <- nº de cidades
0
6
8
12
4
1000.0          0.3          1.1          1.5          9.0
0.3          1000.0          1.2          1.6          9.1
1.1             1.2       1000.0          1.7          9.2
1.5             1.6          1.7       1000.0          8.3
9.0             9.1          9.2          8.2       1000.0
```

Figura 2. Exemplo de instância com cinco pontos de entrega

Algorithm 4 GetDistance

1: $gmaps \leftarrow googlemaps.Client(key =' xxx')$
2: $getDistance \leftarrow gmaps.distance\_matriz((latX, longX), (latY, longY))$
3: $distance \leftarrow getDistance['distance']['text']$
4: $distance \leftarrow (distance.replace(",",".") * 1000)$
5: **return** $distance$

para ele mesmo.

4.2. Análise dos resultados

Para os fins deste trabalho, foi avaliado tanto se as soluções atendiam os critérios, quanto se o tempo de solução era viável. Além disso, comparou-se os resultados obtidos do algoritmo GRASP com os resultados obtidos utilizando apenas o algoritmo de Busca Gulosa. Considerando que, geralmente, quando não se faz uso de metaheurísticas, um algoritmo guloso é o mais utilizado por empresas ao tomar decisões acerca do PRVC.

Iniciou-se realizando testes com o GRASP para refinar os parâmetros utilizados e obter a melhor proporção de menor custo x tempo computacional aceitável. Tal que, para este trabalho o custo representa a distância total a ser percorrida para que todos os vértices sejam visitados. Logo após, utilizou-se a mesma matriz de distância para computar os resultados obtidos com a Busca Gulosa.

Os resultados para esta fase inicial de refinamento de parâmetros, neste caso, o parâmetro α, estão dispostos na Tabela 1. Sendo a quarta coluna referente à quantidade de iterações dentro do GRASP fixa em 10 e o α utilizado para a fase de construção que varia entre 0.70, 0.80 e 0.90. Da quinta a sétima coluna é mostrado os resultados e tempo referente ao algoritmo GRASP e a oitava e nona mostram os resultados obtidos a partir da Busca Gulosa. Outro ponto importante é frisar que o tempo médio está em milissegundos.

Com base nos primeiros resultados obtidos, na maioria dos casos, o $\alpha = 0.7$ mostra resultados bem melhores em relação aos outros parâmetros. Além disso, é perceptível que os resultados obtidos com o GRASP são bem mais otimizados que os resultados da Busca Gulosa, em um tempo computacional similar. Por exemplo, na instância PA com

				GRASP			Busa Gulosa	
Qtd. Pontos	**Cap. veículo**	**UF**	**Parâmetros**	**Custo médio**	**Menor custo**	**Tempo médio**	**Menor custo**	**Tempo médio**
100	50	RJ	iterações = 10, α = 0,70	190,2	186,5	0,54	229,7	0,06
		DF		1.123,6	1.118,5	0,50	1.444,2	0,07
		PA		536,1	534,3	0,59	611,1	0,06
200	100	RJ		720,9	719,2	1,82	854,1	0,20
		DF		1.054,0	1.050,9	1,82	1.751,4	0,20
		PA		1.252,6	1.250,0	1,93	1.503,4	0,21
100	50	RJ	iterações = 10, α = 0,80	200,3	199,5	0,53	226,8	0,07
		DF		992,5	990,7	0,49	1.584,2	0,06
		PA		556,0	553,2	0,53	1.464,2	0,08
200	100	RJ		762,1	759,9	1,80	973,0	0,20
		DF		1.309,7	1.308.2	1,75	1.677,1	0,19
		PA		1.239,4	1.238,7	1,89	1.527,0	0,22
100	50	RJ	iterações = 10, α = 0,90	212,3	210,6	0,66	269,9	0,08
		DF		1.193,0	1.191,4	0,51	1.368,2	0,06
		PA		575,9	574,3	0,64	694,3	0,05
200	100	RJ		858,7	857,1	1,87	999,2	0,21
		DF		1.394,3	1.392,1	1,76	1.498,0	0,24
		PA		1.366,2	1.364,2	1,93	1.554,7	0,23

Tabela 1. Tabela de resultados (α) dos testes com o algoritmo GRASP e da Busca Gulosa

200 pontos, o GRASP empregou uma redução de aproximadamente 16% no custo da solução.

Contudo, ainda existem formas de melhorar o resultado obtido com o algoritmo GRASP, por exemplo, refinando o parâmetro de iterações ($MaxIterations$). Os resultados para o refinamento das iterações estão dispostos na Tabela 2.

		Melhor Caso		
Qtd. pontos	**UF**	**Iteração = 10**	**Iteração = 20**	**Iteração = 30**
100	RJ	186,5	186,5	184,3
	DF	1.118,5	922,7	878,6
	PA	534,3	537,8	534,0
200	RJ	719,2	723,6	699,9
	DF	1.050,9	1.077,7	1.069,1
	PA	1.250,0	1.216,9	1.150,9

Tabela 2. Tabela de resultados dos testes do refinamento das iterações

Com esses novos resultados ficam claros que o aumento da quantidade de iterações, torna os resultados cada vez melhores. No entanto, a relação de aumento de interações é diretamente proporcional ao aumento do tempo computacional. Dessa maneira, a solução de melhor qualidade que possui um custo computacional aceitável, fica com os parâmetros: α igual a 0.70 e quantidade de iterações igual a 30.

5. Conclusão

Neste trabalho aplicamos a meta-heurística GRASP para a resolução do Problema de Roteamento de Veículos Capacitado. Para a validação da proposta de solução, foram geradas instâncias a partir de uma base de dados real fornecida pela empresa Loggi, contendo as informações: quantidade de pontos, demandas, e matriz de distâncias.

O método proposto neste trabalho mostrou-se como viável para resolução do problema do roteamento de veículos capacitado. Apresentou resultados satisfatórios e conseguiu-se atender todos os critérios elencados tanto pelo problema, quanto pela própria técnica e os recursos computacionais.

A grande vantagem do método proposto está em sua facilidade de adaptação a diversos projetos que busquem esta resposta ao PRVC. Entretanto, a API do Google utilizada para cálculo das distâncias não é gratuita, o que gerou algumas dificuldades na geração das instâncias.

Ao se resolver o problema abordado, "manualmente" ou por algum algoritmo mais simples que o GRASP, como um algoritmo puramente guloso, pode-se correr o risco de facilmente cair em ótimos locais. Nos experimentos computacionais realizados, foi observado que o GRASP é bem mais robusto nesse sentido. Desse modo, acredita-se que a solução apresentada neste trabalho pode auxiliar o setor logístico de empresas, a fim de se ter um melhor controle sobre a operação de transporte com base em dados reais.

É fundamental a realização de estudos complementares sobre a técnica e o problema. Além disso, é de suma importância que essas técnicas sejam cada vez mais incorporadas no âmbito empresarial, de modo a testá-las no contexto de uso diário para que assim, a técnica possa ser cada vez mais aperfeiçoada.

Como trabalhos futuros, dado que o GRASP é um método multi-partida e "sem memória", pois informações de uma iteração não transmitidas a iterações futuras, pode-se combinar o GRASP proposto com mineração de dados para que se possa re-utilizar informações obtidas em iterações passadas [Santana et al. 2020]. Neste trabalho foi explorado apenas uma vizinhança na fase de busca local, então um caminho natural para investigações futuras seria elaborar um maior número de vizinhanças e combiná-las em um algoritmo VND (*Variable Neighborhood Descent*) [Clímaco et al. 2021].

Referências

Berger, J. e Barkaoui, M. (2003). A hybrid genetic algorithm for the capacitated vehicle routing problem. In *Genetic and evolutionary computation conference*, pages 646–656. Springer.

Clarke, G. e Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581.

Clímaco, G., Rosseti, I., Silva, R., e Guerine, M. (2021). Reactive grasp for the prize-collecting covering tour problem. *RAIRO: Recherche Opérationnelle*, 55:1441.

Dantzig, G. B. e Ramser, J. H. (1959). The truck dispatching problem. *Management science*, 6(1):80–91.

de Araújo, F., Lima, A. A., e Lima, M. d. A. C. (2018). Otimização de rota e redução dos custos logísticos: estudo de caso em uma empresa de contabilidade. *Brazilian Journal of Development*, 4(1):136–144.

Gendreau, M., Hertz, A., e Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management science*, 40(10):1276–1290.

GOOGLE (2021). Distance matrix api. `https://console.cloud.google.com/marketplace/product/google/distance-matrix-backend.googleapis.com?q=search&referrer=search`. Acesso 07 de outubro de 2021.

Hasle, G., Lie, K.-A., e Quak, E. (2007). *Geometric modelling, numerical simulation, and optimization*. Springer.

Holmes, R. e Parker, R. (1976). A vehicle scheduling procedure based upon savings and a solution perturbation scheme. *Journal of the Operational Research Society*, 27(1):83–92.

LOGGI (2021). Github loggi. `https://github.com/loggi/loggibud`. Acesso 08 de outubro de 2021.

Mester, D. e Bräysy, O. (2007). Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers & operations research*, 34(10):2964–2975.

Mole, R. e Jameson, S. (1976). A sequential route-building algorithm employing a generalised savings criterion. *Journal of the Operational Research Society*, 27(2):503–511.

Resende, M. G. e Ribeiro, C. C. (2016). *Optimization by GRASP*. Springer.

Rodrigue, J.-P., Comtois, C., e Slack, B. (2016). *The geography of transport systems*. Routledge.

Santana, Í., Plastino, A., e Rosseti, I. (2020). Improving a state-of-the-art heuristic for the minimum latency problem with data mining. *International Transactions in Operational Research*.

Sauer, J. G. (2007). *Abordagem de Evolução diferencial híbrida com busca local aplicada ao problema do caixeiro viajante*. PhD thesis, Master's thesis, Pontificia Universidade Católica do Paraná.

Silva, P. C. d. e Silva Júnior, J. B. d. (2018). Análise da representação semântica de modelos de dados do formato json. *Revista de Sistemas e Computação-RSC*, 8(1).

Taillard, É. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673.

Tarantilis, C. D. e Kiranoudis, C. T. (2002). Using a spatial decision support system for solving the vehicle routing problem. *Information & Management*, 39(5):359–375.

Toth, P. e Vigo, D. (2002). The vehicle routing problem monographs on discrete mathematics and applications siam. *Philadelpia, Pennsylvania*.

Toth, P. e Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on computing*, 15(4):333–346.