Pesquisa Operacional para o Desenvolvimento
Vol. XX, n. YY, pp. ZZ-ZZ, ANO: XXXX

ISSN 1984-3534

FERRAMENTA EM VBA COM INTERFACE EM EXCEL PARA SOLUCIONAR O VRPTW[1]

Bruno Roberto Zanotto Renner[a] [*], **Alexandre Checoli Choueiri**[b]

[a] Universidade Federal do Paraná, Curitiba-PR, Brasil

Departamento de Engenharia de Produção

Recebido xx/xx/xxxx, aceito xx/xx/xxxx

RESUMO

O problema de roteirização de veículos é um dos mais estudados no campo da otimização. Os estudos, no entanto, não se aprofundam nas questões práticas para a aplicação de algoritmos em problemas reais, como a coleta de dados geográficos. Esse descompasso entre pesquisa e prática gera um contingente de empresas que utilizam roteirização em suas operações diárias, porém não conseguem implementar soluções próprias devido a falta de informações, e também não podem arcar com os custos de *softwares* comerciais. Este artigo descreve todas as componentes necessárias para a criação de roteirizadores que possam ser utilizados na prática, sem custos para o usuários. Todas as componentes são implementadas em uma linguagem de programação disponível para usuários do sistema operacional Windows (VBA), de forma que o acesso fica facilitado para pequenas e médias empresas.

Palavras-chave: VRPTW, roteirização, matriz de distâncias, API

[*] Autor para correspondência. E-mail: brunozrenner@hotmail.com

[1] *Todos os autores assumem a responsabilidade pelo conteúdo do artigo.*

1. Introdução

Gerir eficientemente a cadeia de suprimentos é uma atividade indispensável para que os operadores logísticos se mantenham competitivos no mercado (Ballou, 2006). Uma das parcelas mais importantes desta gestão se desdobra nos custos logísticos, sendo grande parte destes incorridos pela atividade de transportes Rodrigue et al. (2006). Por conta disso, otimizar as tomadas de decisões relativas a esta área pode resultar em grandes economias para a empresa.

Uma das decisões mais complexas no transporte se dá na definição de rotas de coleta e entrega para clientes, pois este é um problema de análise combinatória que pode envolver muitas variáveis interligadas. O desafio de determinar as rotas que minimizem tanto os custos de transporte quanto o tempo gasto, é conhecido como "Problema de Roteirização de Veículos"(*Vehicle Routing Problem, VRP*). O VRP é um dos problemas mais estudados na área de otimização logística e possui uma família vasta de problemas derivados, ou seja, problemas que incorporam mais características do mundo real na forma de restrições. Uma dessas variantes derivados do VRP é o "Problema de Roteirização de Veículos com Janela de tempo" (*Vehicle Routing Problem with Time Windows, VRPTW*).

O VRPTW, diferentemente do VRP comum, leva em consideração a janela de tempo em que os clientes podem receber o veículo que fará a entrega ou coleta, assim como o tempo que será despendido para realizar a ação. Caso o veículo chegue ao local antes do horário inicial da janela, o mesmo terá que esperar para realizar a atividade e não existirá a possibilidade de atender um cliente após sua janela ser encerrada. A adição dessa restrição torna o VRPTW mais aderente a problemas reais do que o VRP.

Grandes e médias empresas podem otimizar o VRPTW por meio de algoritmos contidos em softwares comerciais especializados (Routeasy, GraphHopper), incorrendo, no entanto, em custos por essa utilização. Como os dados de entrada necessários para a otimização de qualquer problema de roteirização envolve informações geográficas, e estas, em sua maioria, também devem ser pagas, os custos pela utilização dos softwares são escaldos pelo tamanho dos problemas. Esse cenário pode inviabilizar a utilização de softwares comerciais por uma parcela do mercado, composto por pequenas/ micro empresas e ONGs.

Embora esses agentes possuam demandas no sentido de otimização de rotas, os mesmos não se beneficiam de um otimizador automático (software comercial), o que implica em uma geração manual de todo esse planejamento. Essa atividade demanda tempo do operador que as realiza, e usualmente rotas com poucos clientes (10 a 20) já consomem um tempo considerável. Rotas com mais de 50 clientes se tornam inviáveis para qualquer tipo de planejamento manual. Com base nisso percebe-se uma demanda não atendida por parte desses agentes, em relação a um software mais simples e de baixo custo (de preferencia gratuitos) que seja capaz de otimizar e automatizar essas tarefas de roteirização.

Diferente de muitos problemas de otimização que podem ser resolvidos unicamente por parâmetros passados pelo usuário, problemas de roteirização implicam na coleta de distâncias e tempos entre pontos, que via de regra são coletados por meio de API's. Isso constitui uma camada extra de dificuldade para codificar o problema, além de trazer a necessidade de levar em consideração o fator financeiro do projeto, uma vez que a obtenção desses dados não é gratuita.

Este artigo tem por objetivo então, descrever o processo para a implementação

de um software de otimização para o VRPTW, em um ambiente que seja de fácil acesso para pequenas e médias empresas, como o excel, por meio da linguagem de programação VBA. Além do núcleo do otimizador (algoritmo heurístico), toda a parte de busca dos parâmetros por meio de APIs é desenvolvida na própria linguagem, apontando inclusive opções gratuitas para a coleta desses dados.

O resto do artigo está organizado da seguinte forma: na seção 2 apresentamos uma breve revisão de literatura envolvendo o VRPTW. Na seção 3 os componentes de implementação do software são apresentados (heurística gulosa e coleta de parâmetros) e finalmente, na seção 5 os comentários finais e conclusões são desenvolvidos.

2. Revisão de Literatura

O VRPTW pode ser definido da seguinte maneira: seja $G = (V, E)$ um dígrafo conectado formado por um conjunto de $n + 1$ pontos (V), com 0 sendo o armazém e os outros n vértices os clientes, e um conjunto E de arcos conectando todo par de pontos $(i, j), \forall i, j \in V$. Cada elemento de $V \setminus \{0\}$ (pontos) contém uma janela de tempo tw_i, um tempo de processamento tp_i e uma demanda q_i. A janela de tempo tw_i representa o intervalo em que o vértice pode ser visitado; se o veículo chegar antes do inicio da janela ele deve esperar, caso chegue depois do tempo final ele não poderá atender o cliente (vértice). O tempo tp_i está relacionado ao tempo que o veículo fica no vértice para realizar o processamento (entrega ou coleta). Já a demanda q_i indica uma quantidade (escalar) que deve ser levada ou recolhida do cliente. Cada arco $\in E$ possui uma distância d_{ij} e um tempo t_{ij}, relacionados a distância e ao tempo percorridos para sair do vértice i e chegar ao j. O objetivo do problema pode variar. Na maioria dos casos, no entanto, se baseia em encontrar uma solução que minimize o número de rotas K para uma frota de veículos com a mesma capacidade C, de forma que cada ponto seja visitado apenas uma vez dentro de sua janela de tempo e que a demanda acumulada de cada rota não ultrapasse a capacidade C dos veículos. Outro objetivo do VRPTW pode ser a minimização da distância total percorrida pela frota ou até mesmo do tempo despendido pela frota para a realização das atividades. Além disso, não é permitido satisfazer demandas em entregas divididas e todas as rotas devem iniciar e finalizar no armazém.

O VRP e suas diversas variações têm sido muito estudados e colocados em prática através de diferentes algoritmos nas últimas décadas. Eles podem ser resolvidos tanto por métodos exatos, que garantem a solução ótima de maneira custosa, quanto por métodos aproximativos, preferencialmente escolhidos pelos pesquisadores por chegarem a boas soluções custando muito menos computacionalmente. Um exemplo de método exato é o da programação dinâmica proposto por Kolen et al. (1987) que utiliza o algoritmo *Branch and bound* para chegar ao ótimo. Esse método é uma adaptação do método proposto por Christofides (1976) que resolve o CVRP (*Capacited Vehicle Routing Problem*). Para que fosse possível resolver o VRPTW, Kolen introduziu no modelo o recurso “tempo acumulado das rotas”, porém esse método só é capaz de resolver problemas com no máximo 15 nós (clientes). Já o método exato de geração de colunas proposto por Desrosiers et al. (1984) permite a resolução de problemas maiores e demonstrou possuir um desempenho atrativo em instâncias com até 100 nós. O método consiste na divisão do problema em sub problemas com características combinatórias semelhantes ao problema principal.

Por mais que hoje existam métodos exatos modernos e mais velozes do que antigamente, os métodos aproximativos estão muito a frente no que se refere a performance

em problemas de otimização de média e grande complexidade, dentre os métodos aproximativos para resolução do VRPTW podemos citar: Algoritmo de Recozimento Simulado (*Simulated Annealing*, SA), Algoritmos de Busca Harmonica (*Harmony Search Algorithm*, HAS) Yassen et al. (2017), Algoritmos de Busca Tabu (*Tabu Search*, TS) Hedar e Bakr (2014), Pesquisa de vizinhança variável (*Variable Neighborhood Search*, VNS) de Armas e Melián-Batista (2015), Otimização de Colônia de Formigas (*Ant Colony Optimization*, ACO), Algoritmos Genéticos (*Genetic Algorithms*, AG), dentre outros.

Taha et al. (2017) apresentaram um algoritmo combinado de morcegos com Busca em Vizinhança de Grande Porte (*Large Neighborhood Search*, LNS) para o VRPTW. No algoritmo foi utilizado um mecanismo eficiente chamado *large neighborhood generation*, que resultou na obtenção de soluções de qualidade para o problema. Uma versão hibrida do SA contendo múltiplas temperaturas que resolve o VRPTW com múltiplos objetivos foi apresentado em BañOs et al. (2013). Nesse caso, dois objetivos são considerados simultaneamente. O primeiro leva em consideração minimizar a distância percorrida e equilibrar as rotas, na medida em que o segundo busca a minimização do desequilíbrio das distâncias percorridas entre um cliente e outro dentro de uma rota, assim como o total de itens recebidos pelos clientes.

Uma variação do ACO é apresentado por Khoshbakht e Sedighpour (2011). O algoritmo se tornou mais eficiente do que o original, graças ao acréscimo de uma rotina de resgate e de um mecanismo de troca local e busca local. Essas modificações trouxeram mudanças na configuração do feromônio e o uso de uma lista de candidatos para a escolha da próxima formiga, isso possibilitou um melhor desempenho do algoritmo em problemas mais complexos e diversificados, bem como a prevenção de uma prematura convergência em ótimos locais.

Zhang et al. (2020) utilizou um AG como objeto de pesquisa e propôs um algoritmo evolutivo híbrido multiobjetivo com amostragem rápida e pesquisa global baseada em estratégia e sequência de rotas local.

A Otimização de Enxames de Partículas (*Particle Swarm Optmization*, PSO) é muito aplicado para problemas contínuos e nos últimos anos tem sido adaptado para problemas de otimização combinatória. Ele se baseia no comportamento de bandos de aves quando estão explorando em busca de recursos alimentares. Salehi Sarbijan e Behnamian (2022) propuseram um PSO com múltiplos objetivos e um VNS multiobjetivo de enxame de partículas para resolver o problema da rota do veículo alimentador em tempo real (*real-time collaborative feeder vehicle path problem*, RTCFVRP) com janela de tempo flexível.

3. Desenvolvimento

Nessa seção abordaremos o algoritmo utilizado para a resolução do problema bem como a interface criada e suas funcionalidades, descrevendo em detalhes todas as APIs utilizadas para coleta dos dados.

3.1. O algoritmo: Heurística Gulosa

Para a otimização do VRPTW um algoritmo guloso foi desenvolvido. Vale notar que qualquer método pode ser encaixado no software, de forma a viabilizar o desenvolvimento de algoritmos mais competitivos a qualquer momento. As APIs funcionam de forma modular, e se comunicam com o otimizador por meio de parâmetros em sub-rotinas,

criando uma independência entre as diversas funcionalidades da aplicação. Um pseudocódigo do algoritmo guloso desenvolvido é apresentado em 1.

Algorithm 1 Algoritmo guloso para resolução de VRPTW

Entrada: Vetor C de clientes; Matriz D de distâncias
Saída : Solução LS contendo a lista de rotas

1 $C_s \leftarrow OrdenaVetor(C)$; // Ordena por ordem crescente de Finalização da janela de tempo
2 $LS \leftarrow \emptyset$; // Lista inicial de rota vazias
3 **while** $|C_s| \neq 0$ **do**
4 $r \leftarrow \emptyset$; // Iniciando uma nova rota vazia
5 $r \leftarrow C_s(0)$; // Inserindo o primeiro cliente à rota
6 $C_s \leftarrow \{C_s \setminus C_s(0)\}$; // Removendo o elemento C(0) do vetor C
7 $i \leftarrow 0$
8 **while** $(|C_s| \neq 0) \wedge (i \leq |C_s|)$ **do**
9 $r_f \leftarrow r$; // Criando uma cópia da rota
10 $r_f \leftarrow r_f \cup C_s(i)$; // Adicionando o próximo cliente à rota copiada
11 **if** $(FactTempo(r_f) = T) \wedge (FactCapacidade(r_f) = T)$ **then**
12 $r \leftarrow r_f$; // Substituindo a rota pela rota copiada
13 $C_s \leftarrow \{C_s \setminus C_s(i)\}$; // Removendo o cliente roteirizado do vetor
14 **else**
15 $i \leftarrow i + 1$
16 **end**
17 **end**
18 $LS \leftarrow LS \cup r$; // Adicionando a rota na lista de rotas
19 **end**
20 **return** LS ; // Retorna LS

O algoritmo guloso exige como entrada um conjunto C que contenha todos os clientes que serão roteirizados no problema. O conjunto precisa conter as informações de janela de tempo, demanda e tempo de atendimento. Os algoritmos gulosos constroem uma solução por partes, sempre partindo de um conjunto vazio e adicionando novos componentes de solução a cada iteração. Além disso, todo novo componente é escolhido de forma a maximizar ou minimizar alguma função. Vale notar que essa otimização na escolha do próximo elemento não garante o ótimo da solução final, refletindo a característica principal dos algoritmos gulosos: múltiplos ótimos locais não garantem um ótimo global.

A ideia do algoritmo guloso desenvolvido é a de seleção de elementos para uma rota considerando as suas janelas de tempo: clientes com um término de janela de tempo mais cedo devem ter a preferência na roteirização. Isso se dá pelo fato de que, se deixados para o final, a probabilidade de que o veículo chegue após o término de sua janela de tempo (e portanto não possa atender o cliente) é maior. Dessa forma, o primeiro passo do algoritmo é a ordenação do vetor C de forma crescente em relação aos términos de janela de tempo (linha 1 do Algoritmo 1), gerando o conjunto ordenado C_s.

Todas as rotas criadas são armazenadas na estrutura LS, que inicialmente está vazia (linha 2 do Algoritmo 1). Como no VRPTW existe um número ilimitado de veículos, temos que todos os clientes serão necessariamente atendidos. Isso reflete o critério de parada do algoritmo, nas linhas 3 e 15 de 1: toda vez que um cliente

for alocado a uma rota o mesmo é removido do conjunto de clientes C, de forma que quando o conjunto é vazio, significa que todos os clientes já foram roteirizados e po rtanto, o algoritmo finalizou.

Nas linhas 4 e 5 uma nova rota r é inicializada e o primeiro cliente do conjunto ordenado C_s é alocado a esta rota. Na linha 6 o cliente é removido do conjunto C_s (pois já está alocado em uma rota).

O próximo passo é adicionar os outros clientes a esta rota recém criada. Para isso, será feito uma estrutura de repetição que permita o código verificar todos os clientes ainda existentes no conjunto C. Um contador i (linha 7) sera utilizado para isso, este começa valendo 0 já que o antigo elemento $C(O)$ foi retirado do conjunto ao ser roteirizado. Os clientes são adicionados enquanto 4 condições são satisfeitas (linhas 8 e 11):

1. Existem clientes a serem roteirizados
2. O índice do cliente que está sendo considerado para inserção (i) é menor ou igual ao tamanho de C_s
3. A rota é factível em relação à janela de tempo
4. A rota é factível em relação à capacidade do veículo

Para iniciar a verificação da factibilidade de inserção do cliente $C_s(i)$ na rota r, uma cópia de r é feita em r_f (linhas 9 e 10) e o novo cliente é adicionado à mesma. A rotina $FactTempo(r_f)$ verifica então se a rota r_f satisfaz as restrições de janela de tempo dos clientes enquanto que $FactCapacidade(r_f)$ verifica se a nova demanda da rota está dentro da capacidade do veículo que irá atender a rota (linha 11). Se uma das duas verificações for negativa, o cliente não será adicionado a rota e o código irá acrescentar em 1 o contador i (linha 15), para que na próxima iteração outro cliente seja testado nessa rota. No entanto, se ambas verificações forem verdadeiras, significa que a inserção do cliente $C_s(i)$ satisfaz tanto a restrição de janelas de tempo quanto de carga, de forma que a rota r é atualizada na linha 12 e o cliente $C_s(i)$ é removido do conjunto na linha 13.

Quando o laço da linha 8 é violado (uma rota completa foi gerada), a rota atual r é adicionada na solução LS (linha 18). Se ainda existirem clientes não roteirizados, uma nova rota é inicializada (linha 4) e todo o processo é repetido.

Na próxima seção descrevermos os problemas envolvidos na criação de roteirizadores, bem como a forma de se utilizar as devidas APIs.

3.2. Problemática na implementação de roteirizadores

Como mostrado na seção 2, o VRPTW é um problema muito estudado pela comunidade de pesquisa operacional, sendo que existe uma vasta literatura sobre o mesmo. No entanto, percebe-se que existe uma dissonância entre a criação desses algoritmos de otimização envolvendo rotas, e a utilização dos mesmos em instâncias reais.

Diferentemente de outros problemas de otimização, em que todos os dados necessários para que os algoritmos operem podem ser fornecidos pelo usuário, o VRPTW demanda dados externos, estes eles: pares de latitude e longitude dos endereços de clientes e distância e/ou tempo de percurso entre todos os pontos envolvidos no problema. Essa coleta de dados é feita através de API's (A*plication Programming Interface*) que podem ser definidas como mecanismos que permitem a comunicação

entre dois softwares com funções distintas através de uma dinâmica de requisição e resposta, como mostrado na Figura 1.

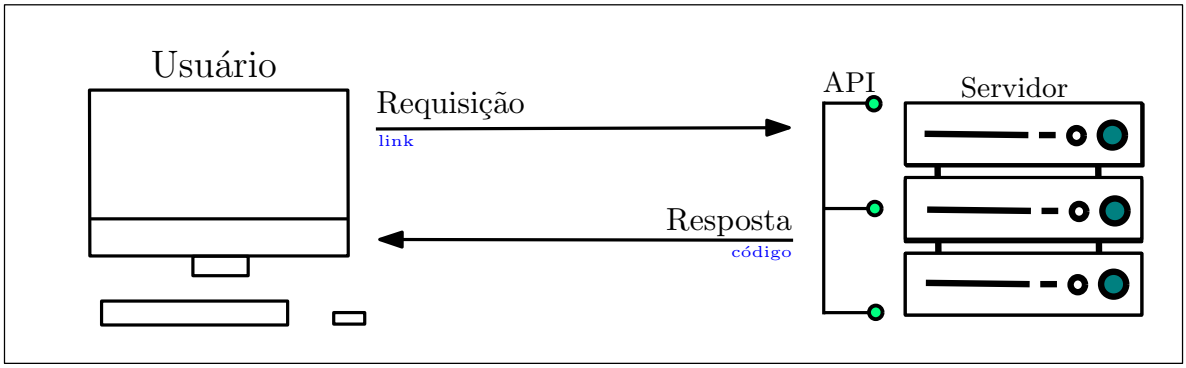


Figura 1: Funcionamento de uma API

A conexão com uma API é feita através do método HTTP (*Hyper Text Transfer Protocol*), um protocolo de comunicação utilizado mundialmente na rede de internet para transferir dados entre um cliente e um servidor que irá se basear em um *link* contendo o serviço a ser requisitado e os parâmetros exigidos (como mostrado na subseção a seguir). De forma simplificada, o usuário "pede"uma informação (Usuário, na Figura 1), que então é processada por um servidor que contém a informação (Servidor, na Figura 1), e esta é então retornada para o usuário. Esse "pedido"é realizado por meio de uma URL por meio do protocolo HTTP.

Existem diferentes tipos de métodos HTTP dependendo do objetivo da conexão, entre eles podemos citar: "GET"para obter dados do servidor sem alterá-los, "POST"para enviar dados a serem criados no servidor, "DELETE"para remover algum recurso do servidor, entre outros. Para o VRPTW, precisamos apenas do método "GET".

O link de requisição gerado irá conter todos os dados que a API precisa para buscar as informações no servidor e devolve-las ao usuário. Atualmente o formato de resposta mais comum das API's é o JSON (*Java Script Object Nation*): notação para a transferência de dados que segue um padrão específico (como é possível verificar na subseção seguinte). Para que o arquivo recebido em JSON seja útil ao usuário, ele precisará receber algum tipo de tratamento para extração dos dados.

Nesse trabalho essa extração foi feita através de uma biblioteca para VBA chamada JsonConverter. Para utilizar esse código, basta copiá-lo e colá-lo em um módulo dentro do seu arquivo VBA e chamar a função passando como parâmetro a variável que contém a resposta obtida da API. O resultado obtido a partir da biblioteca "JsonConverter"é um dicionário contendo os dados do arquivo JSON organizados em uma estrutura hierárquica, facilitando o acesso. Essa dinâmica será explicada na subseção a seguir.

3.2.1. API's utilizadas em VBA

Nesta seção descreveremos quais APIs foram utilizadas para a implementação do software, bem como exemplificaremos a sua utilização. São necessárias API's para duas buscas no software: para obtenção dos pares de latitude e longitude relativas a cada endereço, e para a obtenção de informações a respeito de distâncias e tempos entre cada coordenada.

3.2.2. Busca de Latitude e Longitude - Geocoding free

O objetivo da API geocode é buscar pares de coordenadas (latitude e longitudes) para os endereços que o usuário insere como entrada na ferramenta (chamado de geocodificação), caso esse dado não tenha sido fornecido pelo mesmo. Por exemplo, considerando o seguinte endereço:

R. brg. Franco, 2300, Curitiba

A seguinte URL de requisição deve ser criada:

http://geocode.maps.co/search?q=R.brg.Franco,2300,Centro,Curitiba&api_key=API_KEY

E o servidor do geocode retorna o JSON com as informações como mostrado abaixo:


```
[
    {
        "place_id": 6217389,
        "licence": "Data OpenStreetMap contributors, ODbL 1.0.
        https://osm.org/copyright",
        "osm_type": "way",
        "osm_id": 44393848,
        "boundingbox": [
            "-25.4417003",
            "-25.4402975",
            "-49.2781365",
            "-49.2765937"
        ],
        "lat": "-25.44101455",
        "lon": "-49.27736085423784",
        "display_name": "Shopping Curitiba, 2300, Rua
        Brigadeiro Franco, Centro, Curitiba, Regi o
        Geogr fica Imediata  de Curitiba, Regiao
        Metropolitana de
        Curitiba, Regi o Geogr fica Intermedi ria
        de Curitiba,
        Paran , South Region, 80250-130, Brazil",
        "class": "shop",
        "type": "mall",
        "importance": 0.5300099999999999
    }
]
```


Uma segunda API que desempenha o mesmo papel é nominatim.

3.2.3. Busca de Latitude e Longitude - Google Geocoding

Esta API é fornecida pelo google, portanto existe uma cobrança por sua utilização (Google Geocoding). Como a geocoding é uma API gratuita, pode acontecer de o

servidor não ter a informação de lat/long para determinado endereço, por esse motivo mostramos uma alternativa paga, porém que sempre retorna as informações desejadas. Vale ressaltar que, ao criar uma conta nessa API, existe um limite mensal de requisições gratuitas que podem ser realizadas. Desta forma, na maioria dos casos, se usado juntamente com a geocoding, dificilmente as requisições ultrapassam os limites gratuitos mensais. Como é necessário um cadastro prévio, uma chave é fornecida para cada usuário realizar as suas requisições (é por essa chave que o servidor identifica o número de requisições realizadas, para posterior cobrança). Essa chave deve fazer parte da *request*, nos exemplos abaixo ela aparece como **APIKEY**.

Considerando o seguinte endereço:

> Rua Paraiba, 3290, Guaira, Curitiba

Temos a seguinte URL de requisição:

> http://maps.googleapis.com/maps/api/geocode/json&address=Rua Paraiba+3290+Guaira+Curitiba&key=APIKEY

E o servidor retorna o JSON com as seguintes informações:


```
{
    "results": [
        {
            "address_components": [...],
            "formatted_address": "R. Paraiba, 3290 - Guaira,
            Curitiba - PR, 80630-000, Brasil",
            "geometry": {
                "location": {
                    "lat": -25.4736089,
                    "lng": -49.2851399
                },
                "location_type": "RANGE_INTERPOLATED",
                "viewport": {
                    "northeast": {
                        "lat": -25.4722599197085,
                        "lng": -49.28379091970849
                    },
                    "southwest": {
                        "lat": -25.4749578802915,
                        "lng": -49.28648888029149
                    }
                }
            },
            "place_id": "Ej1SLiBQYXJhw61iYSwgMzI5MCAtIEd1YcOtcmEs
            IEN1cml0aWJhIC0gUFIsIDgwNjMwLTAwMCwgQnJhemlsIjESLwoU
            ChIJOdCiEGDj3JQRhERvEm4Q_7wQ2hkqFAoSCUlGx8We5NyUEXok
            xntVT9aq",
            "types": [
                "street_address"
            ]
```

```
            }
        ],
        "status": "OK"
    }
```

3.2.4. Busca de Matriz de Distâncias - OSRM Table Service

Essa é a API que fornece como entrada um conjunto de latitudes e longitudes e retorna uma matriz de tempos e distâncias entre os mesmos. A API OSRM é gratuita.

Considerando os seguintes pares de latitude e longitude:

> -25.44101455/-49.27736085, -25.4736089/-49.2851399,
> -25.476940/-49.291077

Temos a seguinte URL de requisição:

> http://router.project-osrm.org/table/v1/driving/25.44101455,49.27736085;
> 25.4736089,49.2851399;25.47694,49.291077

E o servidor retorna o JSON com as seguintes informações:

```
{
    "code": "Ok",
    "destinations": [ ],
    "durations": [
        [
            0,
            2277.6,
            1146.8
        ],
        [
            2285.6,
            0,
            1257.6
        ],
        [
            1150.8,
            1247.1,
            0
        ]
    ],
    "sources": [ ]
}
```

O importante dessa resposta são os dados dentro de "durations"que posteriormente serão inseridos em uma matriz dentro do VBA.

3.2.5. Conversão de JSON em dicionário

Em linguagens de programação, dicionários são estruturas de dados que permitem armazenar valores associados a chaves, formando uma espécie de mapeamento onde cada chave é única e retorna apenas um valor. Com o módulo JsonConverter, os valores retornados pelas APIs (JSON) são convertidos em dicionários.

A Figura 2 mostra os dados no dicionário após a requisição feita para o geocode (primeiro exemplo de API da seção anterior). O retorno é feito em uma lista com um objeto do tipo dicionário.

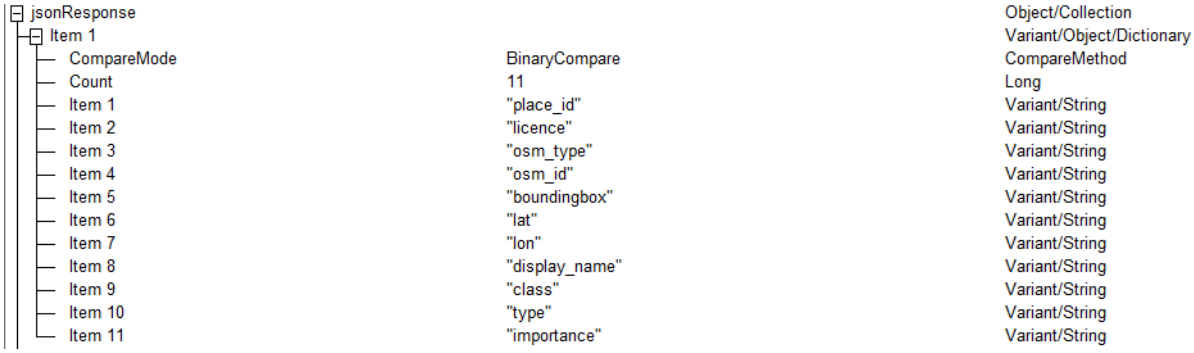


Figura 2: Exemplo de JSON convertido em dicionário - resposta Geocoding API

Considerando que a lista que contém o dicionário tem o nome *jsonResponse*, podemos acessar os dados de latitude e longitude da seguinte forma:

```
jsonResponse(1)("lat")
jsonResponse(1)("lon")
```

A lista possui um item (por isso o acesso por (1)), dentro deste item existe um dicionário com todas as informações ("place_id","license", etc), por isso o acesso é feito pela chave ("lat"e "lon").

4. Interface e Funcionalidades

Nesta seção descreveremos como todos os módulos do software se conectam (entrada de dados, otimizador, APIs e solução). O processo é representado pela Figura 3.

A utilização da ferramenta se da a partir da inserção de dados por uma planilha do excel ou pela leitura de um arquivo CSV (Lista de endereços na Figura 3). Os dados a serem inseridos são:

1. Nome do cliente
2. Rua
3. Número
4. Bairro
5. Cidade
6. Estado
7. País
8. Latitude

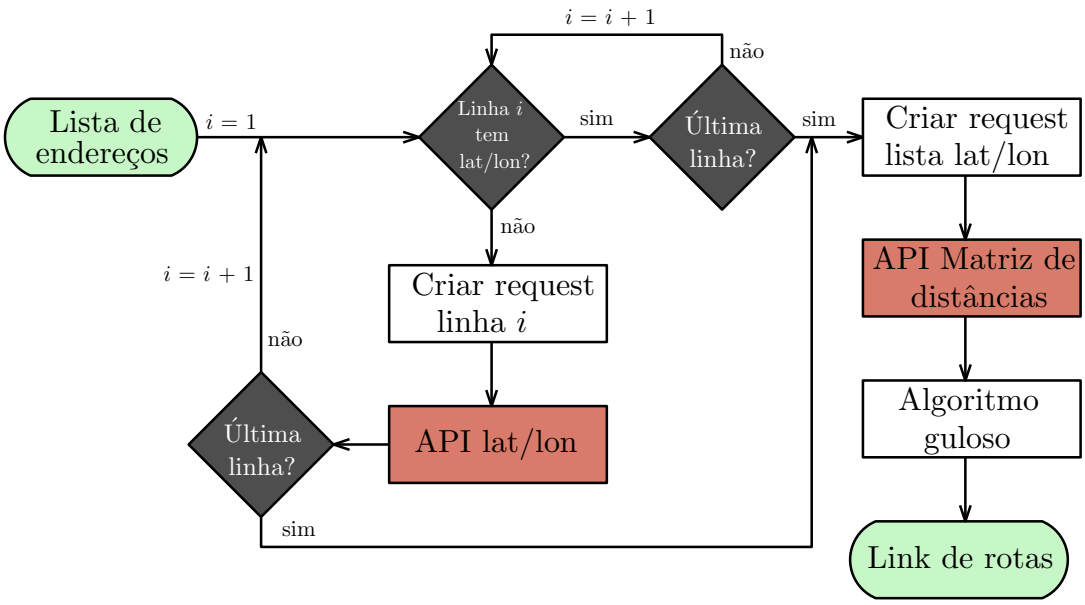
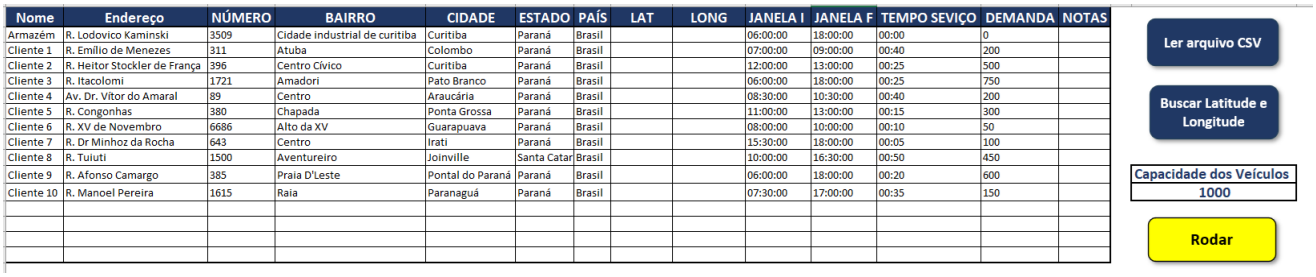


Figura 3: Funcionamento dos módulos

9. Longitude
10. Horário de inicio da janela
11. Horário de finalização da janela
12. Tempo de serviço
13. Demanda

Após todos os dados serem inseridos, é preciso ter certeza que todas as latitudes e longitudes foram fornecidas, caso alguma linha não possua essas informações, o usuário deverá, primeiro, obter esses dados clicando no botão "Buscar Latitude e Longitude"como mostra a Figura 4.



Nome	Endereço	NÚMERO	BAIRRO	CIDADE	ESTADO	PAÍS	LAT	LONG	JANELA I	JANELA F	TEMPO SEVIÇO	DEMANDA	NOTAS
Armazém	R. Lodovico Kaminski	3509	Cidade industrial de curitiba	Curitiba	Paraná	Brasil			06:00:00	18:00:00	00:00	0	
Cliente 1	R. Emílio de Menezes	311	Atuba	Colombo	Paraná	Brasil			07:00:00	09:00:00	00:40	200	
Cliente 2	R. Heitor Stockler de França	396	Centro Cívico	Curitiba	Paraná	Brasil			12:00:00	13:00:00	00:25	500	
Cliente 3	R. Itacolomi	1721	Amadori	Pato Branco	Paraná	Brasil			06:00:00	18:00:00	00:25	750	
Cliente 4	Av. Dr. Vítor do Amaral	89	Centro	Araucária	Paraná	Brasil			08:30:00	10:30:00	00:40	200	
Cliente 5	R. Congonhas	380	Chapada	Ponta Grossa	Paraná	Brasil			11:00:00	13:00:00	00:15	300	
Cliente 6	R. XV de Novembro	6686	Alto da XV	Guarapuava	Paraná	Brasil			08:00:00	10:00:00	00:10	50	
Cliente 7	R. Dr Minhoz da Rocha	643	Centro	Irati	Paraná	Brasil			15:30:00	18:00:00	00:05	100	
Cliente 8	R. Tuiuti	1500	Aventureiro	Joinville	Santa Catar	Brasil			10:00:00	16:30:00	00:50	450	
Cliente 9	R. Afonso Camargo	385	Praia D'Leste	Pontal do Paraná	Paraná	Brasil			06:00:00	18:00:00	00:20	600	
Cliente 10	R. Manoel Pereira	1615	Raia	Paranaguá	Paraná	Brasil			07:30:00	17:00:00	00:35	150	

Figura 4: Endereços inseridos como entrada sem latitude e longitude

Esse botão criará uma requisição e buscará a latitude e longitude de todas as linhas de endereços que não possuírem (loop da Figura 3 com "API lat/lon") como foi explicado na subseção anterior.

Uma vez que todas as latitudes e longitudes estiverem preenchidas, o usuário pode dar início à roteirização clicando no botão "Rodar"como mostrado na Figura 4. A lista de pares de latitude e longitude coletada são usadas como *input* para a busca da matriz de distâncias ("API Matriz de distâncias"na Figura 3).

Com a matriz de distâncias (e tempos), pode-se dar início à criação da solução

passando como parâmetro a matriz juntamente com o vetor de clientes com todos os seus dados e a capacidade dos veículos.

Nesta etapa o algoritmo guloso é iniciado ("Algoritmo guloso"na Figura 3). A solução obtida pelo algoritmo será em forma de lista onda cada item será uma outra lista contendo todos os clientes na ordem em que deverão ser visitados naquela rota. A partir de cada item da lista (ou seja, cada rota), será gerado um link do google *maps* com os endereços, para visualização do trajeto completo. Os links são gerados como URLs, de forma que nesta etapa também não existem custos envolvidos.

Para o exemplo mostrado na imagem 4, a solução gerada pelo código foi uma lista contendo 5 rotas, como mostrado nos links e imagens a seguir:

•$Rota1$

3 Clientes roteirizados: Armazém –> cliente 1 –> cliente 4 –> cliente 2 –> Armazém

https://www.google.com.br/maps/dir/-25.48577,-49.34928/-25.38361,-49.18183/-25.5938,-49.40668/-25.42035,-49.26832/-25.48577,-49.34928?entry=ttu

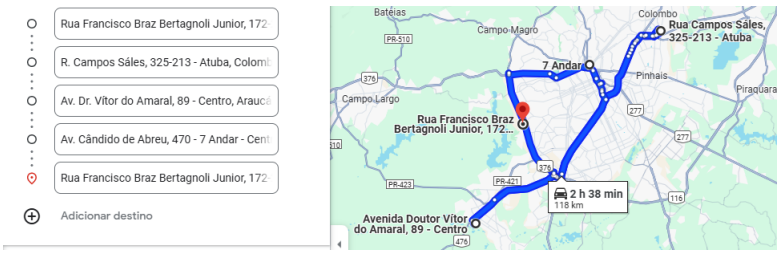


Figura 5: Rota 1 gerada a partir do exemplo

•$Rota2$

3 clientes roteirizados: Armazém –> cliente 6 –> cliente 5 –> cliente 8 –> Armazém

https://www.google.com.br/maps/dir/-25.48577,-49.34928/-25.38793,-51.46054/-25.06693,-50.20116/-26.25803,-48.81964/-25.48577,-49.34928?entry=ttu

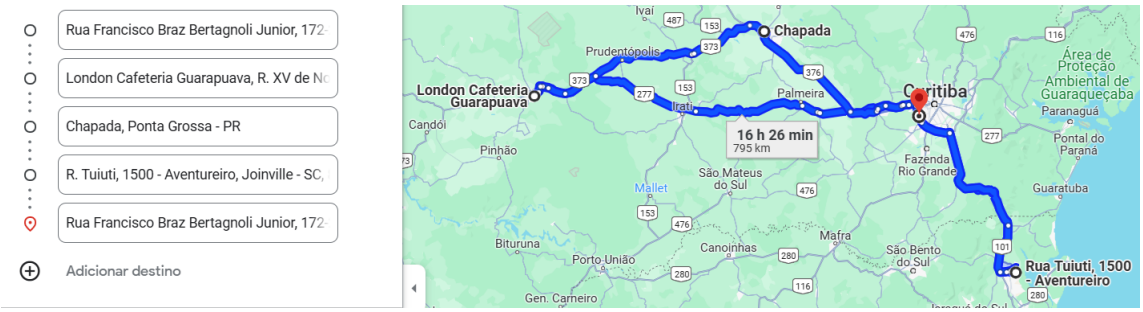


Figura 6: Rota 2 gerada a partir do exemplo

•$Rota3$
2 clientes roteirizados: Armazém –> cliente 10 –> cliente 7 –> Armazém

https://www.google.com.br/maps/dir/-25.48577,-49.34928/-25.51887,-48.51888/-25.46692,-50.64367/-25.48577,-49.34928?entry=ttu

Figura 7: Rota 3 gerada a partir do exemplo

•$Rota4$
1 cliente roteirizado: Armazém –> cliente 3 –> Armazém

https://www.google.com.br/maps/dir/-25.48577,-49.34928/-26.23777,-52.6898/-25.48577,-49.34928?entry=ttu

Figura 8: Rota 4 gerada a partir do exemplo

•$Rota5$
1 cliente roteirizado: Armazém –> cliente 9 –> Armazém

https://www.google.com.br/maps/dir/-25.48577,-49.34928/-25.69843,-48.47396/-25.48577,-49.34928?entry=ttu

5. Conclusão

O problema de roteirização de veículos (VRP) é um dos mais estudados pela comunidade de pesquisa operacional, dada a sua direta aplicação no cenário logístico. Desde o trabalho seminal de Dantzig (**??**) em 1959, a busca por algoritmos eficientes para a resolução do problema não parou, tanto exatos quanto heurísticos.

Esse avanço na pesquisa é focado em questões teóricas em relação a otimização, e técnicas em relação a eficiência computacional, considerando que todos os parâmetros

Figura 9: Rota 5 gerada a partir do exemplo

para que os algoritmos sejam executados já são conhecidos e coletados *a priori*. Essa etapa, no entanto, é essencial para que os algoritmos sejam utilizados, e de forma alguma em aplicações reais se dá de forma trivial. O principal parâmetro para qualquer algoritmo de roteirização é uma matriz de distância entre todos os pontos. Para que essa matriz seja condizente com a realidade (não simplesmente a distância euclidiana), os dados devem ser calculados usando mapas reais, e é neste ponto que reside o descolamento da pesquisa com as aplicações.

Como a pesquisa não aborda como essa coleta deve ser feita, existe um descompasso entre a enorme eficiência dos algoritmos, e a utilização dos mesmos, principalmente por pequenas e médias empresas. Neste artigo criamos um passo a passo descrevendo os principais componentes necessários para que algoritmos de roteirização possam ser utilizados na prática. A principal componente é a utilização de APIs, para que o software possa se conectar com servidores que guardam os dados necessários. Embora a maioria das APIs gere um custo, neste trabalho mostramos opções viáveis para a criação de programas usando apenas componentes gratuitos. Além disso, todos os componentes são desenvolvidos em uma linguagem de programação de fácil acesso para usuários Windows, o *Visual Basic for Application* (VBA).

Com este artigo, então, pequenas e médias empresas que possuem uma demanda por algoritmos de otimização, e no entanto, não podem arcar com os custos de grandes fornecedores de software, podem criar seus próprios roteirizadores de forma independente.

Agradecimentos. Os autores agradecem o apoio da Sociedade Brasileira de Pesquisa Operacional (SOBRAPO). Os agradecimentos não devem ultrapassar 05 linhas de texto.

Referências

BañOs, R., Ortega, J., Gil, C., MáRquez, A. L., e De Toro, F. A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows. Comput. Ind. Eng., v. 65, n. 2, p. 286–296, 2013. ISSN 0360-8352https://doi.org/10.1016/j.cie.2013.01.007.

Ballou, R. Gerenciamento da Cadeia de Suprimentos: Logística Empresarial. Gerenciamento da Cadeia de Suprimentos: Logística Empresarial: Bookman. ISBN 9788536305912, 2006.

Christofides, N. The vehicle routing problem. Revue française d'automatique, informatique, recherche opérationnelle. Recherche opérationnelle, v. 10, p. 55–70, 1976. https://api.semanticscholar.org/CorpusID:209057218.

de Armas, J. e Melián-Batista, B. Variable neighborhood search for a dynamic rich vehicle routing problem with time windows. Computers Industrial Engineering, v. 85, p. 120–131, 2015. ISSN 0360-8352https://www.sciencedirect.com/science/article/pii/S0360835215001151.

Desrosiers, J., Soumis, F., e Desrochers, M. Routing with time windows by column generation. Networks, v. 14, n. 4, p. 545–565, 1984. https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230140406.

Hedar, A.-R. e Bakr, M. A. Three strategies tabu search for vehicle routing problem with time windows. Computer Science and Information Technology, v. 2, n. 2, p. 108–119, 2014.

Khoshbakht, M. e Sedighpour, M. An optimization algorithm for the capacitated vehicle routing problem based on ant colony system. Australian Journal of Basic and Applied Sciences, v. 5, p. 2729–2737, 2011.

Kolen, A. W. J., Kan, A. H. G. R., e Trienekens, H. W. J. M. Vehicle routing with time windows. Operations Research, v. 35, n. 2, p. 266–273, 1987. ISSN 0030364X, 15265463http://www.jstor.org/stable/170698.

Rodrigue, J., Comtois, C., e Slack, B. The geography of transport systems. The geography of transport systems, 2006.

Salehi Sarbijan, M. e Behnamian, J. Real-time collaborative feeder vehicle routing problem with flexible time windows. Swarm and Evolutionary Computation, v. 75, p. 101201, 2022. ISSN 2210-6502https://www.sciencedirect.com/science/article/pii/S2210650222001675.

Taha, A., Hachimi, M., e Moudden, A. A discrete bat algorithm for the vehicle routing problem with time windows. In: 2017 International Colloquium on Logistics and Supply Chain Management (LOGISTIQUA). In: 2017 International Colloquium on Logistics and Supply Chain Management (LOGISTIQUA). IEEE, 2017. p. 65–70.

Yassen, E. T., Ayob, M., Nazri, M. Z. A., e Sabar, N. R. An adaptive hybrid algorithm for vehicle routing problems with time windows. Computers Industrial Engineering, v. 113, p. 382–391, 2017. ISSN 0360-8352https://www.sciencedirect.com/science/article/pii/S0360835217304515.

Zhang, W., Yang, D., Zhang, G., e Gen, M. Hybrid multiobjective evolutionary algorithm with fast sampling strategy-based global search and route sequence difference-based local search for vrptw. Expert Systems with Applications, v. 145, p. 113151, 2020. ISSN 0957-4174https://www.sciencedirect.com/science/article/pii/S0957417419308681.