Integrated Decision Support System for Rich Vehicle Routing Problems

Philippe Lacomme, Gwénaël Rault, Marc Sevaux

Integrated Decision Support System for Rich Vehicle Routing Problems

Philippe Lacomme[a], Gwénaël Rault[b,c], Marc Sevaux[b]

[a] *Université de Clermont-Ferrand, LIMOS, Clermont-Ferrand, France*
[b] *Université Bretagne Sud, Lab-STICC, UMR 6285, CNRS, Lorient, France*
[c] *Mapotempo, Bordeaux, France*

Email addresses: placomme@isima.fr (Philippe Lacomme), gwenael@mapotempo.com (Gwénaël Rault), marc.sevaux@univ-ubs.fr (Marc Sevaux)
URL: www.mapotempo.com (Gwénaël Rault)

Abstract

Recent economic and environmental constraints push supply chain management systems to adopt closed-loop supply chain operating modes that have to address very complex problems including the end-user quality of services, environmental considerations, and daily transportation time variations. Relevant and challenging research areas require a proper coordination between the data provider software (Transport Management Software) and the operational research tool in charge of trip definition.

This paper proposes a decision support system applied to the Vehicle Routing Problem able to tackle very large instances with real-life constraints. Our contribution is to propose an architecture that handle both static resolution prior to the completion of routes and update them in a dynamical context during their completions. This is implemented through a REST based API using numerous state-of-the-art operational research methods. Moreover, this system in used in practice by the Mapotempo company.

Keywords: Supply Chain, Transportation System, REST API, Rich VRP

1. Introduction

Recent economic and environmental constraints increase the pressure on the supply chain management systems to integrate multi-attribute decisions both for the last mile and closed-loop of the supply chain (Cardenas et al., 2017; Macharis & Melo, 2011). Due to the wide development of e-commerce, and the quick growth of online retailers (Chen et al., 2018) the demand has been fragmented. Previously the supply activities were performed in shopping centers or within few shops, with only few contacts with the final user. Nowadays, every house, office or shop is a potential point of interaction between the supply chain and the customer.

The last mile is about delivering goods provided by the whole supply chain to specific customers, where the closed-loop, is more about taking back products

from customers to recover additional value from the product itself or some of its components. The objectives of the supply chain model can include profit maximization, environmental impact customer quality of service. This is placed in a dynamic urban environment with transportation conditions which variate from day to day and along the day. The road network may change, and traffic jams may appear. This requires most of the time robust solutions or dynamic systems to update iteratively a solution. Decision making of the supplier and manufacturer in a supply chain management to tackle these real-world situations require efficient Operations Research (OR) tools dedicated to the Vehicle Routing Problem (VRP) (Toth et al., 2014) to obtain an efficient solution regarding model objective. These tools can be based on numerous exact methods, heuristics, or meta-heuristics depending by the size of the instances, which exhibit the complexity of the problem, and the end-user constraint on computational time.

The future of transportation depends on several relevant and challenging research areas that are impacted by 1) environmental constraints, 2) digitization of the services, and of 3) new modes of transport more connected due to the service digitization. The research community needs to collaborate with practitioners to efficiently manage the global transportation that requires a proper coordination between the data provider software (Transport Management Software) and the Operational Research tool in charge of trip definition addressing the whole constraints.

This paper presents an original contribution by making easily reachable OR resolution methods for the VRP and its variants for software engineers through a high level API which handles an entire problem and selects by itself the best suited methods to solve it. Moreover, some low level APIs are available giving access to the set of data necessary to solve such problems and allow the user to interact with the data and the solution and to display these on a map.

This work is motivated by the specific needs in the last mile of the supply chain. It requires fine data to be relevant in such dynamic context. The traffic and the position of vehicles evolve all along the day. Missions to perform may change during the realization of tours. This is particularly the case in the

dynamic or multi period variants of the VRP.

The OR methods embedded within the system are selected both for their flexibility and fastness. Evolving in a dynamic context requires to be able to start from an existing state and give limits in the neighborhood which can be reached through the resolution. It may not be feasible to entirely change the route of a driver while he is already performing his tour. The dynamic context of city logistic requires robust solutions in order to keep an high level customer satisfaction.

The proposed decision support systems introduce low-level systems such as Geocoder and Router services which give basic bricks used by others systems, respectively to convert addresses into coordinates and provide data about the path between points. Some high-level systems as Optimizer and Fleet services give access to more advanced features. A Web App is as well available to display an overview of the capabilities provided by the combination of whole services.

The architecture is scalable, each service can be replicated and eventually implements mechanism to delegate operations to other machines.

The system is expected to propose good and robust solutions in a very short amount of time fitted to the dynamic environment of the city logistic.

This paper is organized following the next structure. Section 2 will provide an overview of the evolutions in the OR research field to solve the VRP and its variants. Section 3 introduce the concept of Application Programming Interface (API). The following Section 4 propose an application of the API to the problematic of the Decision Support System applied to the last mile. Section 5 provides a particular focus on the implementation of Optimizer-API using the Representational State Transfer (REST) standard. Section 7 will provide a comparison of the Optimizer-API against the alternative available on the market in term of capability. Then a benchmark on various instances of the literature will be presented. Some particular points of the paper will be discussed in Section 8. Finally, Section 9 presents concluding remarks and provide some future perspective to this work.

2. Related work

More than 60 years have elapsed since Dantzig & Ramser (1959) have introduced the VRP and open a new research field for the operation researchers. The community have been largely inspired by the need of the supply chain, this can be shown through the number of variants addressed. This emulation has create a large basis of instance sets on which the community can rely on to compare the efficiency of the resolution methods. Among others we can mention the work of Solomon (1987) and Gehring & Homberger (1999) on the VRP with Time Windows (VRPTW). Cordeau et al. (1997) has introduced various instances on the Multi-Depot VRP (MDVRP) and the Periodic VRP (PVRP). And lately add the Time Windows attributes to theses variants MDVRPTW and PVRPTW (Cordeau & Laporte, 2001). The combination of attributes expresses the need of the community to get closer of the reality. Duhamel et al. (2011) in this trend have introduce instances for the Heterogeneous Fleet VRP (HFVRP) where distance matrices were calculated using the road network instead of traditional measures such as euclidean and Manhattan distance. On another side, as the methods became more refined and their performances have improved, the size of the problems which could be handled increases. This has lead the community to update the classical instances to increase their size such as Vidal et al. (2013a) on the MDVRPTW, PVRPTW and Site Dependent VRP (SDVRP).

In the same time, as the contribution to new variants have been shared. A need of classification of the different attributes has been expressed (Vidal et al., 2013b; Caceres-Cruz et al., 2014) with the idea of exploring which combinations were already explored, which were the current trends and which were the unknown land. Some recent contributions have rushed down this path. That is the case of Penna et al. (2019). They have added various attributes to he HFVRP and provide an hybrid heuristic able to solve efficiently rich variants of the HFVRP. More recently Sadykov et al. (2020) have proposed an exact method based on bucket graph labeling algorithm able to solve a large set of

variants of the VRP.

As a regard to the need to challenge even more the resolution methods with instances of bigger size Arnold et al. (2017) provide both instances of large scale for the Capacitated VRP (CVRP), up to 12000 nodes to serve and an efficient heuristic to solve these. Even if the field is highly active both on the research and the industrial fields, the contribution in the creation of commercial plateforms are rarely shared with the community. The only previous contribution we have found comes from Welch (2017)

3. Web API definition

The first developments of distributed environments for optimization took place in the 1990s, and were based on a client-server architecture that allowed end-users to submit problems and receive solutions using protocols including FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), and even on the lowest protocols TCP-IP (Transmission Control Protocol (TCP) and Internet Protocol (IP)). This first generation of servers rapidly evolved to take advantage of the power of expression offered by the new protocols developed for the World Wide Web (WWW).

Web-based technologies have transformed the design, development, implementation and deployment of decision support systems. As stressed by Cohen et al. (2001) with the definition of web-based decision support systems in Supply Chain Management. The interconnection of modules in the business process models (Wang et al., 2018) of the Supply Chain Management is based on application programming interfaces (API) that define a set of procedure and methods that can be executed by a module through an internet connection. It can be written into any programming language that manages internet access using TCP-IP. Those interfaces are called remotely through internet protocols and allow them to build applications with distant services on servers. Such APIs are designed to provide specific data or services in order to delegate parts of much larger applications. They give access to black-boxes on which developers

can rely on to provide an expected behavior. Managing data and performing complex tasks are often involved to specialized entities as it requires otherwise to have developers with appropriate skills or to have access to large volume of data to feed such systems.

APIs are one way to implement the Service-Oriented Architecture (SOA) paradigm and are designed to tackle the problem of integration of heterogeneous sources and to make heterogeneous systems inter-operable. A key to design such API is how requests are processed on the server side (processing model) and how clients invoke and use this service (interaction model).

An API service processing model may be business object (or method) centric or it may be document centric.

- the **business object-centric** approach is driven by a series of method calls. These successive method calls apply the business logic of the service to a set of business objects hold by the API and containing the information required for processing the requests.

- a **document-centric** API keeps the business logic separates from the document content. The service receives a document that only contains data and no explicit binding to the business logic that has to be applied. There is no mapping of the request to the business logic; that is, specific methods are not invoked by the client. The processing workflow is deducted from the document content.

Despite the effort of numerous studies in recommending adequate API structure, several relevant shortcomings can be disputed including but not limited to the following remarks:

- most of the service recommendation approaches focus on recommending services in **isolation** to obtain an availability rank that matches the customer requirement (Almarimi et al., 2019).

- the **confidentiality** becomes more challenging especially with the increasing number of available services interconnected. And even more, regarding

the GDPR.

- a **reasonable access** across the service users depending on the usage in the services ecosystem and on the commercial usage contract that provide equity between customers.

- **avoid unfair API usage** pattern which is a sequence of method calls which are not compliant with the end-user user permissions

- **define synchronous and asynchronous** abstraction layer with a service composition that has to be proved to be correct and complete. Specific techniques for planning execution and allowing the combination of asynchronous services according to behavioral requirements have been provided (Zhao et al., 2012).

The current trend for Web Services is to rely on various services available through the network. It relies on the Service-Oriented Computing (SOC) paradigm and supports the development of rapid, low-cost, inter-operable, evolvable, and massively distributed systems. The promise of SOC is to easily assemble loosely coupled micro-services to build dynamic business processes (Papazoglou et al., 2007; Karakoc & Senkul, 2009). It comes with another important fundamental concept, which is quality of service (QoS). It is used to drive the selection of candidate service through the evaluation of various attributes (Parejo et al., 2014; Ramírez et al., 2017) : cost, availability, response time, reliability, security, latency, documentation.

4. Definition of the Mapotempo Transportation System

Decision support systems (DSS) are information systems that provide assistance to humans involved in complex decision-making processes. Unlike online retail services, the supply chain management requires, especially in its last mile part, to consider various sources of change in the plan. As the situation on the field may evolve quickly it has to consider many feedback loops at every step

of the decision process. Even if a first plan could be defined from scratch, it should anticipate the traffic state to return a sufficiently robust solution.

The vehicle fleet may be heterogeneous in term of physical vehicle, which influence the road network that can be traveled through. It influences also the travel speed, that could also variate from one driver to another due to their experience or their behavior on the road. The time spend at customer's site may variate from one route to another has it depends on the type of vehicle. A truck has more difficulties to find parking places, it is also longer to open the trailer or take out the parcels. The time spend at customer's site also depend on the driver and the politic of quality of service set up by the company. These parameters are usually known in advance.

Some parameters are fixed in advance, it is often the case in the press portage, the routes are defined in advance and send to subcontractors as it. But to improve the margin of such routes external newspaper clippings are introduced to the existing route if they do not improve the total cost of the route and generate no detours.

Furthermore, once a plan has been established it may be hard to change it entirely. At this step, searching an optimal solution is irrelevant, the need is to find a solution of good quality with a small "distance" to the existing one. That is particularly true when the shiploads are prepared in advance to ease the load of the trucks when the drivers arrive to the loading docks. It may also be difficult to dismiss a driver because his route has been dispatched through multiple other routes. Whenever the route has begun, some missions may be assigned to other drivers, if it doesn't depend on the shipload, due to event on the route.

Such DSS must provide an adapted solution from the current state of one or multiple routes with various levels of preservation of the previous solution.

The modules that operate in this context must consider these constraints, states, priority and preferences. The tools at the disposal of the operator should give him correct feedback on the progress of the plan. The route manager may have to adjust the solution or deploy it to its vehicle fleet directly on the field.

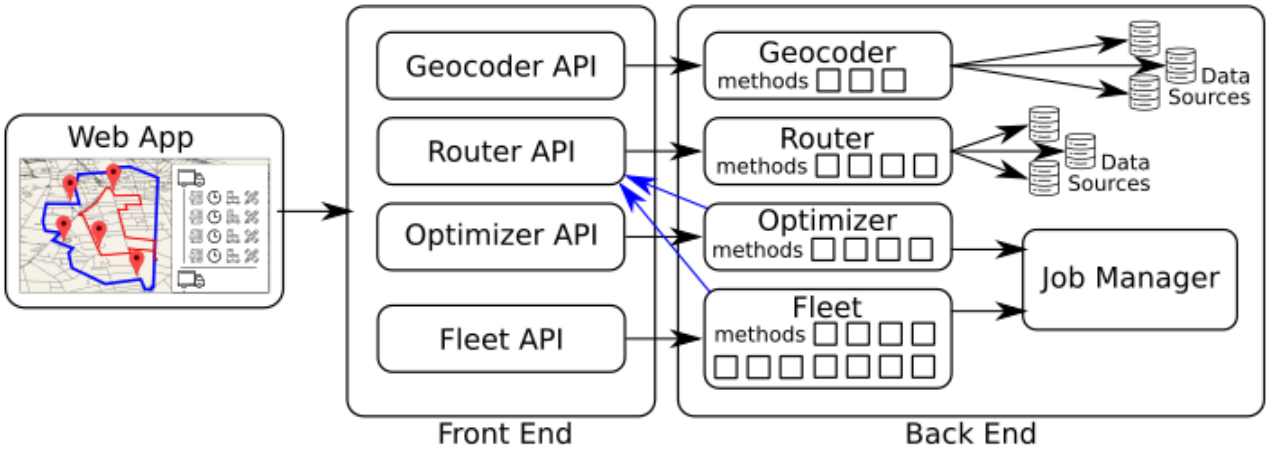

Figure 1: Mapotempo Transportation System Architecture

4.1. Web Services

Mapotempo Transportation System (MTS), as displayed in Figure 1, provides two levels of services. First, a set of low-level web services relative to the management of data (cartography and route optimization) dedicated to computer science engineers. Secondly, a high-level web application that provides a user-friendly graphical interface.

Moreover, the web services are divided in two main layers. The front end assembles all the methods available to computer science engineers who come into direct contact with the various functional bricks. The back end is a structured collection of modules which are dedicated to specific goals to create additional values from the routing data.

The MTS is compartmentalized. Every main functional block is self-contained. Thus, every service may evolve independently from the rest of the global project.

The client, here the Web rely on the services behind and contact them whenever it requires a specific action on the data. Each service implements a set of methods exposed through its API as a set of resources. These have been summarized within Table 1. APIs may communicate between themselves. In particular both Optimizer and Fleet serviced rely on the paths calculated by Router.

Project	Resource	Create	Read	Update	Delete	Synchronous
Web Application	depot	✓	✓	✓	✓	✓
	unit	✓	✓	✓	✓	✓
	vehicle	✓	✓	✓	✓	✓
	customer	✓	✓	✓	✓	✓
	plan	✓	✓	✓	✓	✓
	plan/tour		✓	✓		✓
	plan/optimize	✓				
Geocoder	geocode		✓			✓
	reverse		✓			✓
Router	route		✓			✓
	matrix		✓			✓
	isoline		✓			✓
Optimizer	solve	✓				
	solution		✓		✓	✓

Table 1: Front End resources

4.2. Geocoder

Mailing addresses are usually provided as a text split in multiple fields such as a street, a zip code, and a city. But such addresses could be of poor quality or containing exotic data. The geocoder service has to interpret this information, to find the best source of data which could convert it into coordinates, to evaluate the quality of this data and eventually interrogate another source of data. Mailing address standards change from one country to another and even sometimes from one region to another, mainly because of the syntax construction of the official language of the given locality. Some countries or newly constructed boroughs may not have proper addresses.

Another feature is to translate coordinates into mailing address and is commonly called reverse geocoding. Both of these operations are performed synchronously.

4.3. Router

The last mile part of the supply chain belongs to a dense environment with many regulation rules on the road network. Various means of transportation operate in this context with rules varying from one business field to another and eventually from one driver to another. In this context determining the shortest path between delivery points requires to put in correlation the road network, the vehicle features, the driver behavior, and the geographic points to pass through to complete the expected activities.

The router does not take as input mailing addresses but expects proper coordinates. The geocoding result is entirely dependent on the quality of the data. This quality of the obtained coordinates can only be qualified with a specific expertise. As Geocoder only returns the most probable coordinates corresponding to the information, it could be dangerous and poorly relevant to directly return route data. This choice of separating these two phases also has the advantage to increase the isolation the service.

The *Router* web service has three main features. First, it must determine the shortest path for a given vehicle to navigate through an ordered set of point. Then, the matrix operation has to return the route time and distance between each pair of points. Finally, the isoline feature give the area which could be reached within a given amount of time or within a limited distance. These operations are synchronously performed. The matrix operation may take an important amount of time for a large set of points, but most of the requests are performed for a very small set of points and could be answered in a very short time.

It has to be noted, that it uses by default Open Source and Open Data layers to provide such data. The road network data are coupled with a land use data set in order to adapt the speed profiles accordingly to the population density. The method is currently only available for occidental countries and could be extended as these data are progressively aggregated worldwide (Terroso-Saenz & Muñoz, 2020).

4.4. Optimizer

Like the router service, the optimizer expects already geocoded data. Nevertheless, it performs requests to the router. Indeed, the Vehicle Routing Problem sent usually does not contain its own matrix between all the points of the problem. Unlike the geocoding which could be easily validated by a human operator. Travel time and distance matrices in their entirety are difficult to evaluate. Even if an individual itinerary could be humanly evaluated. The volume of data, as they are dependent on a large set of parameters, is left to the discretion of the project, if not provided by the client itself.

Within this ecosystem, the web service dedicated to the resolution of Vehicle Routing Problems has received a particular attention, as it receives and gathers all the data collected from the others services and must provide an adapted route plan regarding all the constraints described by the client.

Figure 2 gives the conceptual model behind the Optimizer web service. This web service does not have as a goal to store data and let users to add, edit, or delete single customers within a given problem to solve. The client is supposed to send a fully formed problem in a single row, containing all the data of the problem (except the matrix data) with the solve operation. This operation would not answer a solution synchronously, first because the computation of matrices may be time-consuming and secondly because the resolution itself requires heavy computations. Then, the solve operation will return an *id*. This *id* could be sent to the solution resource to check the current status of the resolution, and when terminated the final result. Eventually, the client may decide that a resolution is no more required, he could with this *it* interrupt the resolution. Currently no mechanism of notification has been put in place. Some questions have to be answered first, as we have no assumption on the use case of this API. The VRP could be sent from one device (the route manager desktop) and the solution could be expected both on the same device or on another one (a driver app).

The Optimizer Back End includes every operation performed behind the hood to answer client requests. It is structured into multiple modules. When a

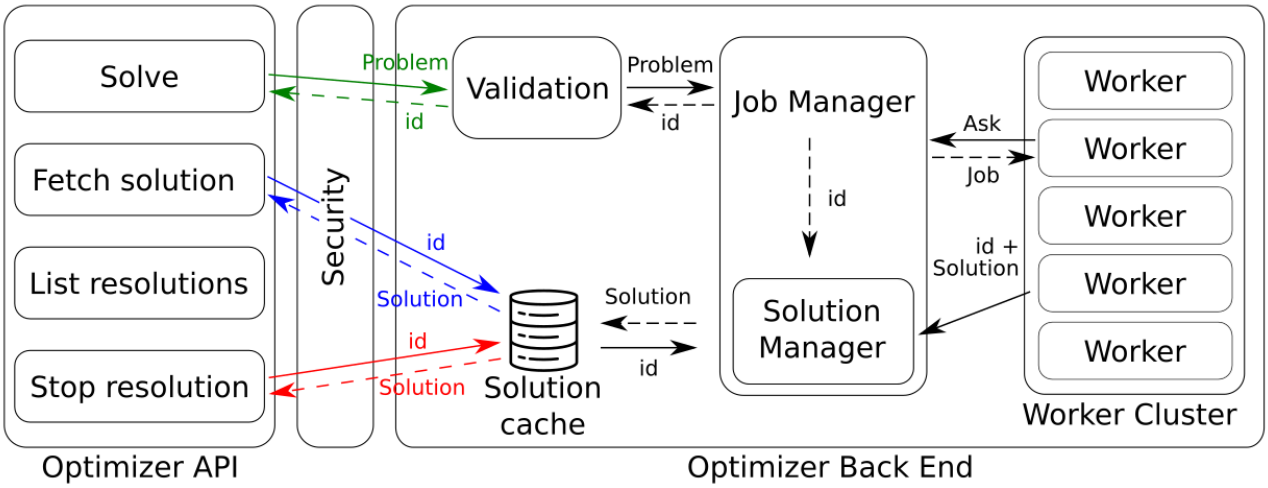

Figure 2: Optimizer Web Service

problem is posted, it must be validated to verify if the data are both relevant individually and consistent regarding the problem in its entirety. Note that, a single problem may provide different solutions on different runs. It may depend among other things on the performances currently available. Indeed, due to the diffusion of cloud computing (Choi et al., 2018), the resolution may be performed on a dedicated server or on a server requested on the fly, which is nowadays convenient and allows to deploy resources to match the expected quality of service. The task assignment is delegated to a *Job Manager*. It has been assumed that users of such a web service are supposed to be software engineers or advanced users. With this assumption and having in mind that the solutions provided are dependent of the current state of the road network and the performances of the available machines. A single problem may obtain different results depending on the moment it is sent. Even if the differences are minor, every problem received by the server leads to a new resolution. If such differences are insignificant from the point of view of the user, the cache can be implemented on the client-side. That is the case in the calls of the Web Application, a given problem will only be solved once.

Concerning the deployment of servers on the fly or the affectations of ded-

icated servers, any problem must be evaluated in order to determine what resources have to be devoted to its resolution. Indeed, this question is complex (Rasley et al., 2016) and has to take into account a large panel of business considerations, some parameters are displayed within Figure 3. The main objective is to find a balance between resource allocation, profitability, and the delay to answer. This subject has become more critical with the development of Infrastructure as a Service (IaaS) providers on the market (Heilig et al., 2020).

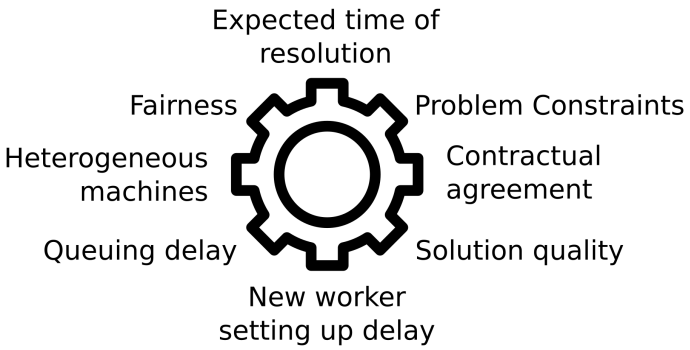

Figure 3: Worker assignment wheel

4.5. Fleet

Another web service, *Fleet*, takes place on the operational side and has for purpose to manage the communication with the route manager, the drivers, and the final customers. The routes organized by the route manager are sent to the drivers. Notifications are sent to final customers with the expected time of arrival (ETA). The driver, on its side, receive the set of missions to perform its tour. *Fleet* may also receive, as the tour progresses, data from the field, both to inform the route manager on eventual issues and the final customers to update the ETA. All these web services could be called independently and may be adapted to a large set of contexts.

4.6. Web Application

The Web Application is available both as a demonstrator of the technical stack and as fully functional Route Management Software (RMS) distributed commercially (see Figure 4). It is a user-friendly application that has been built with the Model-View-Controller (MVC) software design pattern to provide a very low latency but let the user manipulating its data to build its route. The Web application uses a server-side scripting language, which means that the code is executed and interpreted by the server. The client, on its side, only displays generated views with behaviors which trigger controllers on the server-side. The views and the controllers are both exposed on the front end. This has for advantage to allow developers to integrate directly parts the web application, or to re-implement some views to fulfill particular needs.

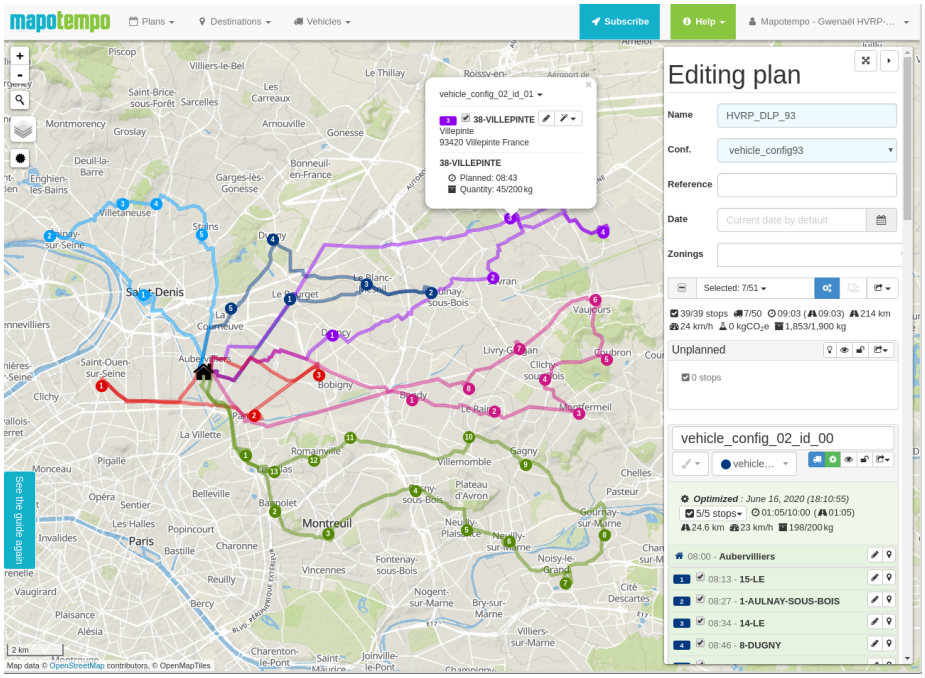

Figure 4: Example of the user friendly interface proposed by the Web Application

This service is designed to fit within a large set of last-mile activities. This generalization implies some choices in the representation of the data, their han-

dling and the operations to apply. Any data representation or behavior implies making the technical choice to balance between reactivity and accuracy. This may also require performing various calls to other services.

The final customers are represented both as complex objects with various properties, especially their address. It is projected on the map with its coordinates. Address and coordinates are then supposed to be highly related. But whenever the point which represents this customer is moved on the map, its coordinates are edited. Does the application have to propose a new address? Or keep the current one, even if the point has moved to a completely different location? And inversely, if the address is edited, does the point have to be geocoded again? The current choice is to geocode the address on the creation of the customer, the following edits suggest the user to edit the according data with a non-blocking button. On a plan, which displays all the routes of a single day, every path of the route is displayed and gives information on the travel time and the associated distance. On any change, automatic (customers insertions) or manual (shift of a customer within a route) shall we recompute the entire path? In the current application, every path is calculated independently, switching two customers only requires to calculate the new detour. But this choice is outdated if we consider traffic data as the whole route is impacted. Edits of the route are in this case less responsive as the route size increase. Such drawbacks illustrate the need of third-party editors to have access to the unitary operation which lets them build their own interface dedicated to the activity of their own customers.

Any plan calculated is dependent on the current state of the road network, the current data relative to the customers, and the vehicle fleet in use. This means that a plan is by essence ephemeral. A solution provided by the Optimizer Web Service is also subject to this temporality. Whenever a user imports again a plan performed months ago within the application, the current road network will be applied to it. Eventually, if the plan becomes infeasible regarding the current constraints, alerts will be raised and a decision from the route manager is expected to arbitrate with its own field experience if the routes are feasible or

not and adjust them consequently. Eventually, his decision could be to request a new resolution.

Furthermore, the supply chain is exposed to inertia. A provisional plan once built may not be changed entirely. Indeed, as some appointments with customers are booked accordingly to the plan along the day, it may be hard to unplan them. The goods are as well prepared and put in the right warehouse bay or vehicle. Such plans are more often subject to the insertion of new flow than to fundamental changes. The level of change is dependent on the organizational choices of each company and should be transcribed as constraints in the Vehicle Routing Problem to solve.

5. Optimizer API

In the sequel, we will focus on the choices performed within the Optimizer service. Indeed, this project is from our point of view a good case study showing the interactions between operational research algorithms, decision-making process and computer science engineering applied to the supply chain.

Choosing a standard to define an API has to be done wisely depending on the context of its usage (Jin et al., 2018; Jacobson et al., 2012). The Event-Driven architectures (Dunkel et al., 2011) expect to have a stable communication all along with the exchange of data and a relatively continuous feed of lightweight data to transmit (Jia et al., 2018). In the context of a Web-Service dedicated to vehicle routing, it might be expected that the API is used on the field with the variable network quality. Moreover, as the processing of a problem may require a long amount of time the necessity of a continuous communication is not mandatory. These reasons have excluded protocols such as Webhooks, WebSockets or HTTPStreaming.

This bears the choice on Request-Response APIs, which expects the clients to send a request and get the expected result directly or provide a key to retrieve the result subsequently, in the case of an asynchronous operation. The communication between the client and the server has to be as lightweight as

possible as we have to deal with the network issues and as the data could be related to really big problems. As SOAP relies on XML which is highly verbose, it can be excluded from the current consideration Tihomirovs & Grabis (2017).

Representational State Transfer (REST) is an established standard (Massé, 2011). Furthermore, we can mention that, as it is close to other web technologies, it is relatively easy for developers to learn how to handle such web service. Note that REST is a software architecture style based on web standards. It means that any of the principles which will be defined below are a convention and may not be applied by every web service (Neumann et al., 2018). The term Restful is applied to services, which comply with every REST principle.

REST exposes resources as part of URLs. A resource is an entity that is exposed from outside of the system and allows to access a part of the Model, or on which an action can be triggered. REST is based on the use of the standard HTTP methods Create, Read, Update and Delete (CRUD) in order to perform transactions on the resources (see Table 2). Any resource implements one or more of the CRUD verbs. Non-CRUD actions that may be applied are not

Table 2: CRUD operations and Rest verbs

CRUD	HTTP verb	Description
Create	POST	A new resource is added to the database
Read	GET	Give a read-only access to resources
Update	PUT/PATCH	Edit or replace existing resources
Delete	DELETE	Remove existing resources

easily represented using such structure. Still, it can usually be sent as an update of the resource state by giving a parameter via the PUT verb. Another way of manipulating those actions is to create a sub-resource which will represent specific actions. A sub-resource shows the relationship with the above resource.

As mentioned above, REST is a standard design to create Web APIs. The rules behind the architecture focused on performance, scalability, simplicity, modifiability, visibility, portability and reliability. The architecture is ruled by

six constraints:

- **Client-Server**: Responsibilities must be separated between the multiple organizational domains. The server receives the requests and store them into a queue. Workers check out this queue to obtain problem to process.
- **Statelessness**: The requests are self-sufficient and do not require the server to store the state of the session. The client sends a request and receive an id as answer, using this id, any client can verify the optimization status and retrieve its result.
- **Cacheability**: Responses are cacheable. From a single optimization, a result could be retrieved without performing the whole resolution.
- **Layered system**: A server can delegate a request to another server to generate a result. The API is organized into an interface delegating resolution to workers which could be placed on another physical machine. This ease the scalability of the system.
- **Code on demand (optional)**: Servers should be able to transfer executable code. The current project doesn't implement this kind of functionality.
- **Uniform interface**: The internal data representation is conceptually separated from the answer returned to the client. It also gives enough information to delete the resource from the server, here using an id. The answer is self-interpretive and holds itself the type of data returned, such that the client knows dynamically which type to handle.

Having a business-centric or a document-centric service is not restricted to the choice of the standard but is more related to the architecture of the project itself. Within a REST context, both alternatives are practicable, the operations may be unitary, following the business-centric definition, such as the user will have to call the right methods to handle the data and perform the expected actions. It is also possible to POST a single document with all the data and

the API will deduce the internal operations to perform. It is also possible to combine both approaches, having some operations which are business-centric and will directly manipulate the data processing. Some other operations that are independent of user interventions.

This property is particularly interesting in our context, indeed we will receive big and complex routing problems to solve, the goal of the API is to provide a black box-like service which will return organized routes. The user would not have to know which operations have to be performed in order to solve the problem sent. Nevertheless, the user may have to manipulate a higher level of the process. As the solving a task may be time-consuming, it will be performed asynchronously, so the user may want to manipulate the object containing the problem to solve with unitary operations. He will have to interrogate on occasion whether or not the resolution is complete. He may also want the resolution to be stopped. He may also want to obtain the list of its resolutions currently in process. These three last operations are business-centric as they do not expect the API to perform hidden and complex tasks but only perform simple and direct tasks directly relative to the operation called.

5.1. Security

The first level of security which has to be discussed is how the data transmitted to the API are kept confidential. At this purpose, HTTP implements the Transport Layer Security (TLS) through the HTTPS protocol (Figure 5). It encrypts the data between the sending app and the web server. First of all, the server has to be known by a certification authority. For this purpose, the server generates a private and a public key and performs a Certification Signing Request (CSR) to a Certification Authority which will act as proof of identity for the server.

Once the server is registered, HTTPS requests can be performed. At the beginning of a new transaction, the client asks the server to transmit a chain which authenticate a certificate authority server. The client verifies if the certificate authority server belongs to its trusted list. If so, the client asks to this

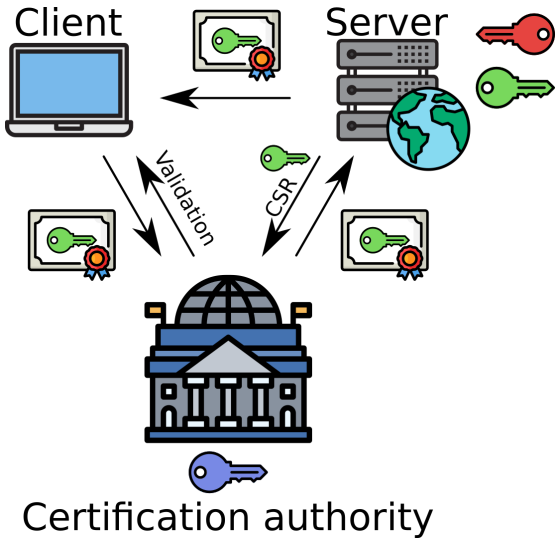

Figure 5: TLS authentication

authority server the public key belonging to the API server, and then send to the server its own public key. Using a trusted third party ensures that both client and server have a unique verified identity. From this point, they agree on a symmetric key within the handshake procedure. As both sides are already authenticated and can exchanged encrypted data, the asymmetric key can be exchanged which speed up the encryption process.

5.2. Authentication

5.2.1. Basic authentication

One of the easiest ways to implement an authentication consists of asking the user a username and a password. It requires no extra libraries or implementation. This solution is simple and widely diffused, but it has also some weaknesses. When an application is compromised, the credentials are exposed which means that private user data may leak through the vulnerability. It becomes even more critical as we know that the same passwords are often used across multiple services. Any access to these credentials maybe gives access to several sources of data even if the applications are provided by an unrelated

service provider. As a consequence, this authentication method is discouraged particularly in the context of business management.

5.2.2. OAuth

Instead of performing the authentication directly within the API, this part may be delegated to another service which is more widely used. In this manner, the user has a centralized access to multiple services with the same credentials, which are only present in a single location. Users authorize applications to access the details of their own account if they successfully login within this centralized location. This way, a user can edit its password without revoking the access of API to its data. And inversely, revoke access of APIs without changing its credentials. This method is particularly interesting as the login is delegated to a trusted party, similar to HTTPS, which decouple the access to personal data of the access to business data. However, it has to be considered wisely, a business is rarely managed by a single user, the access to this set of tokens has been shared by a group of users with multiple right levels. This solution is currently not implemented in the project as it currently used by centralized applications, which only requires a single token, here called api key. But it could with only a few efforts by adding some unitary operations in the input of the API, such as registration or login endpoints.

5.2.3. API Keys

API keys are another way to be authenticate into such service. Theses keys are randomly generated long sequences of characters, making it hard to guess even through a brute-force attack. It has the advantage to be unique, so it is used only by the service generating the key. Another advantage is that the API does not require to delegate the authentication to another service. It is a really good and fast solution to start up a project, until reaching a sufficient user base to consider more advanced authentication methods. The philosophy is similar to OAuth, as the authentication is based on a single key or token. Nevertheless, API keys are less flexible they do not involve a trusted third party.

API keys are the current choice implemented within the project. Indeed, most of the customers interact directly with the Web App and by the same occasion delegates the interactions with the web services. The registration of a new api key is only possible through a contact with the technical team. Nevertheless, this hasn't been blocking as the way to discover the project is to begin with the web application, which has a self-registration system.

5.3. Authorization

Once the client is identified, we have to define which feature it can access. First, we have to define if multiple levels of access are required. This design is dictated by the purpose of the API. That is particularly true if the right access management to data or to resources is required. In the case, that the API only provides a result from self-sufficient data transmitted by the client without interaction from other clients, multiple levels of access are only useful when particular operations access would be restricted. But we may have to restrict for each user the resources allocated simultaneously to a single user.

In the case of interactions between user data, we have to handle privacy and moderation rights. This implies to define individual rights or group management. Individual rights are rarely a good idea as it is harder to handle once the number of users increases. Group management is easier to scale and allows us to define multiple levels of groups. Such a hierarchy allows defining more refined right management.

In the current state of the project, as we have a single API key by company, the data privacy across users of an API key is delegated to the client application. Nevertheless, a single API key only have access to the optimization results sent with this particular key.

5.4. Rate-Limit

API clients may tend to send requests simultaneously with various calls without considering fairness or concurrent requests from other clients. As it is not the client's role to know the charge of the server it has to be managed

on the API side. This is done by a rate-limiting mechanism. Such constraints will make the API protected against denial-of-service (DoS) attacks. It will also prevent the application from abusive usage and spam-sending. A rate-limit could be defined at the server-side regardless of the application and the users. This defines a maximum number of connections to the physical machine simultaneously and will tend to banish misbehaving users. Rate-limiting may also be defined more granularity with a limit defined by source and not only at the scale of the server. This will allow limiting only users with a heavy load. We may also have to consider users which have generally few requests but may send a lot of data in a short amount of time. Shall the API be insensitive to this or may it tolerate such burst calls until a given limit? Does this limitation have to be equal amongst all the clients or defined differently for multiple groups of users who shared resources?

5.4.1. Token-bucket

The token bucket algorithm implementation is based on a maximum number of available tokens, each arriving request consumes a token and new tokens arrive regularly. If there is no token left, the request is rejected.

5.4.2. Fixed-window counter

Within a given time window a maximum number of requests is given. At the end of the period, the counter is reset allowing another bunch set of requests.

5.4.3. Sliding-window counter

Instead of considering a static definition of a minute or an hour, we might consider a floating one. Because you do not want to forget about a potential burst at the end of the last period of time, we may try to smooth the requests.

6. Optimizer Back-End

Once the security layer has been successfully overcome, the optimization layer has two main methods which can be reached from the front-end directly or indirectly.

First, requesting a resolution (Figure 6) requires having a problem, which could be sent entirely from the front-end directly or could be agglomerated through the data layer. So, with a problem, the user still must be authenticated, not from the security but from the contractual agreement point of view. Indeed, users may have to face various policies to process the incoming request. The policies could limit the size of the request, the time to process it, or indicate the eventually dedicated resources allowed to this user. Once the policy rule has been retrieved, the problem is analyzed, and its consistency validated in order to ensure that the data transmitted are relevant both individually and within the context of the global resolution. The policy is then applied depending on the data which have been validated. Note that, some data may have been filtered as they are not relevant but may not prevent problems to be consistent in itself, such data are put aside. If some inconsistent data are part of the core of the problem, the problem cannot be solved, and an error is raised. When a problem passes the validation phase, it is given to the job manager with the policy previously retrieved. The job manager will return in exchange for a job id, which will allow retrieving the solution on a later call.

Secondly, whenever a job has been handled by the job manager and its resolution has been delegated to workers for an asynchronous processing. The client will regularly interrogate the API to know the current status of the resolution (Figure 7). For this purpose, it will request the Get operations on the job resource directly or obtain the result through the Web App which will apply the solution once it is available. In any case, if the solution is not already present in the cache, the operation will interrogate the *Solution Manager* which will return if the id is known, its current status and if possible the partial solution or the final solution whenever it is available.

The exchange of data and status between the resolution layers is displayed in Figure 8. The workers interrogate regularly the job store to know if a job is available for resolution. Note that a worker may not reach certain jobs due to resource policies. If a job is available, the worker takes its own and gets the data associated. Once the job acquired, it will process the tasks described

in Figure 9. At every step of the resolution, the worker may store its current status and eventually the intermediate results. At the end of the resolution, the worker stores the final solution and informs that the job is completed. The worker retrieves at this moment its initial state and starts to fetch again the job store.

The job manager on its side stores problems in the job store when those arrive in exchange of a job id. The job manager also interrogates the store whenever a Get operation is requested to send the content currently stored.

The resolution is divided into various sequential steps which are interdependent preventing then to divide it into multiple tasks which could have been

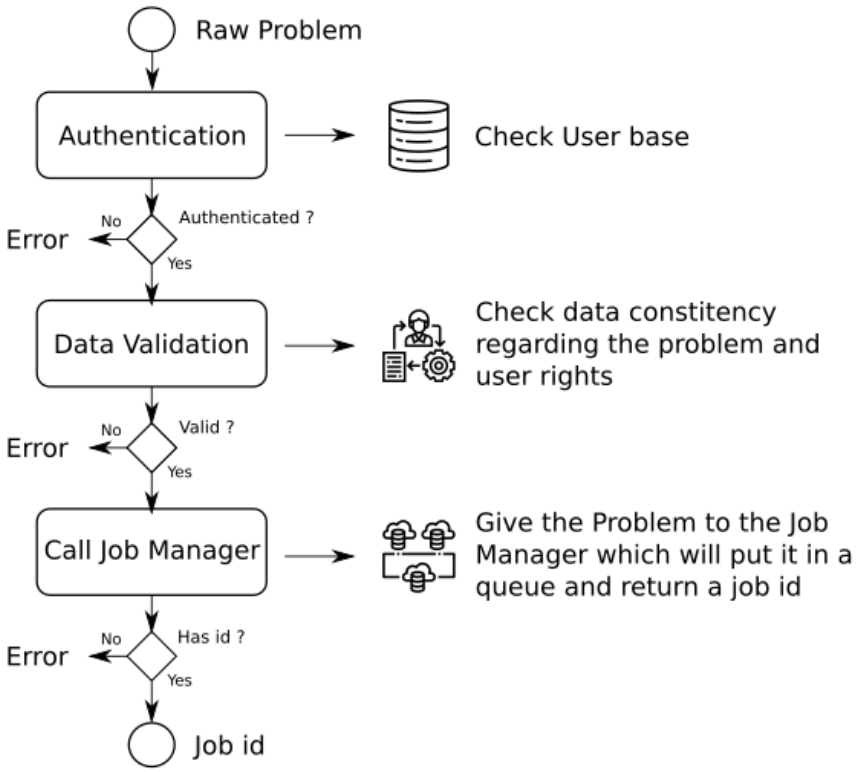

Figure 6: Job manager input

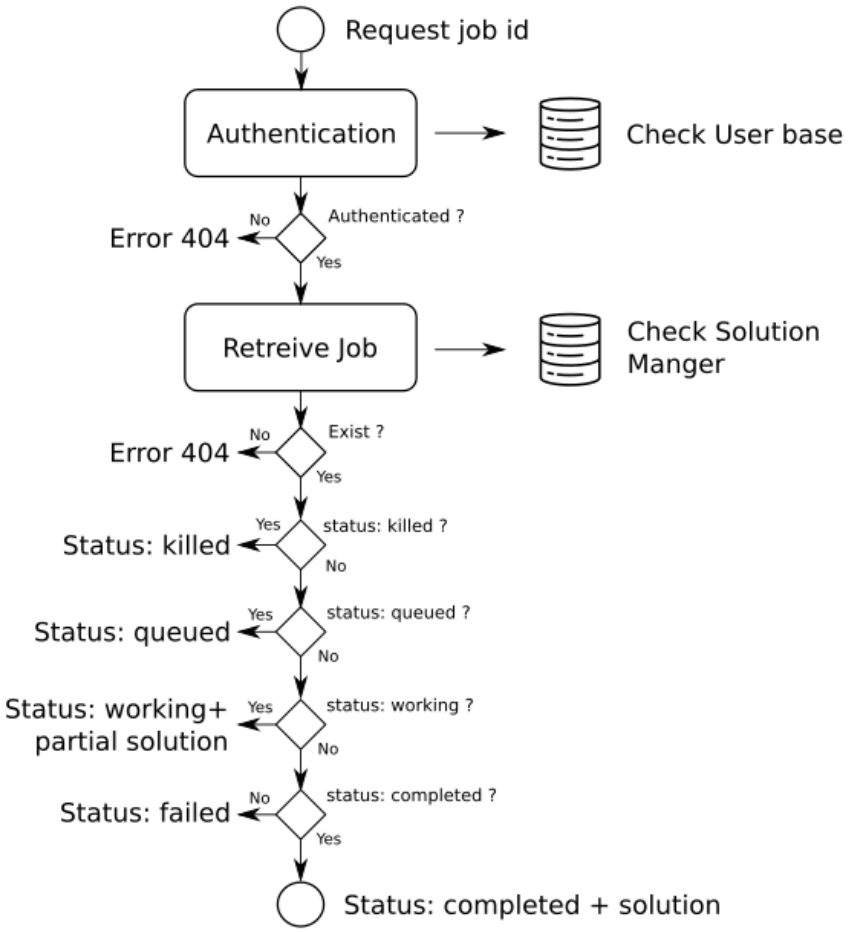

Figure 7: Solution manager

delegated to multiple other workers.

A vehicle routing problem requires one or multiple matrices which is a time-consuming operation. In order to avoid struggling with the computation of insanely heavy matrices the problem may be cut beforehand whenever the estimated time to compute these is too big. Then, the problem is cut in two parts using clustering methods, this step is performed recursively until sub-problems reach an adequate size. Then sub-problems are taken one by one, first, the matrices are computed using the coordinates projected on the road network.

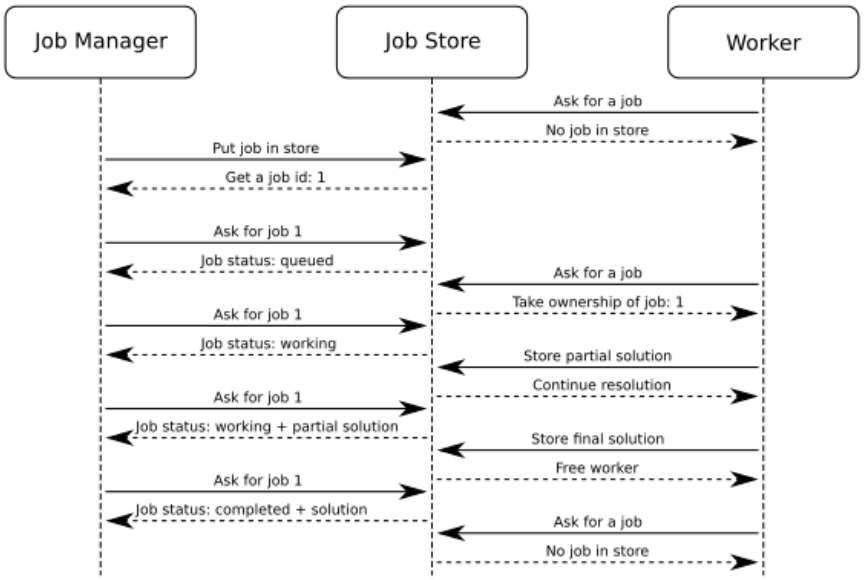

Figure 8: Resolution communication

Once the matrices are ready, the proper resolution may begin. Note that the resolution may produce intermediate solutions, which are stored in order to let the user see if the result is coherent with its expectations. Once a sub-problem is solved the already used vehicles are withdrawn from the pool available for the next sub-problem. Every sub-problem is solved in sequence following this process. Whenever all the sub-problems have been solved, the results are merged and stored, and the status is changed to "completed".

7. Performance Indicators

7.1. Market study

Tables 3 and 4 compare the solution Optimizer-API of Mapotempo against similar solutions on the market as well as the Vehicle Routing Web-based Solver provided by INRIA with the solver developed by Sadykov et al. (2020). Table 3 presents a comparison of the APIs capabilities regardless to the variants it can address. Routific and Tarot Analytics didn't communicate on the solver they

Company/Lab	Solver(s)	API	Sync	Proof	Geocode	Route	Zone	Initial	Relation	Time+ Distance	Distinct Road Networks
Mapotempo	OR-Tools,VROOM	REST				✓	✓	✓	✓	✓	✓
Graphhopper	Jsprit	REST			✓	✓			✓	✓	✓
Verso	VROOM	REST	✓			✓			P&D		
Routific		REST			✓	✓					
Tarot Analytics		REST				✓	✓		P&D		
INRIA	CPLEX,BaPCod	REST		✓							

Table 3: Comparison of API capabilities of various actors of the market

Company/Lab	CVRP	VRPTW	VRPMTW	MDVRP	MDVRPTW	HFVRP	VRPLB	Skills	VRPB	OP	PVRP	VRPSSTW
Mapotempo	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Graphhopper	✓	✓	✓	✓	✓	✓	✓	✓				
Verso	✓	✓	✓	✓	✓		✓	✓				
Routific	✓	✓	✓	✓	✓	✓	✓	✓				
Tarot Analytics	✓	✓	✓	✓	✓	✓	✓	✓				
INRIA	✓	✓		✓		✓						

Table 4: Vehicle Routing Problems variants that can be addressed

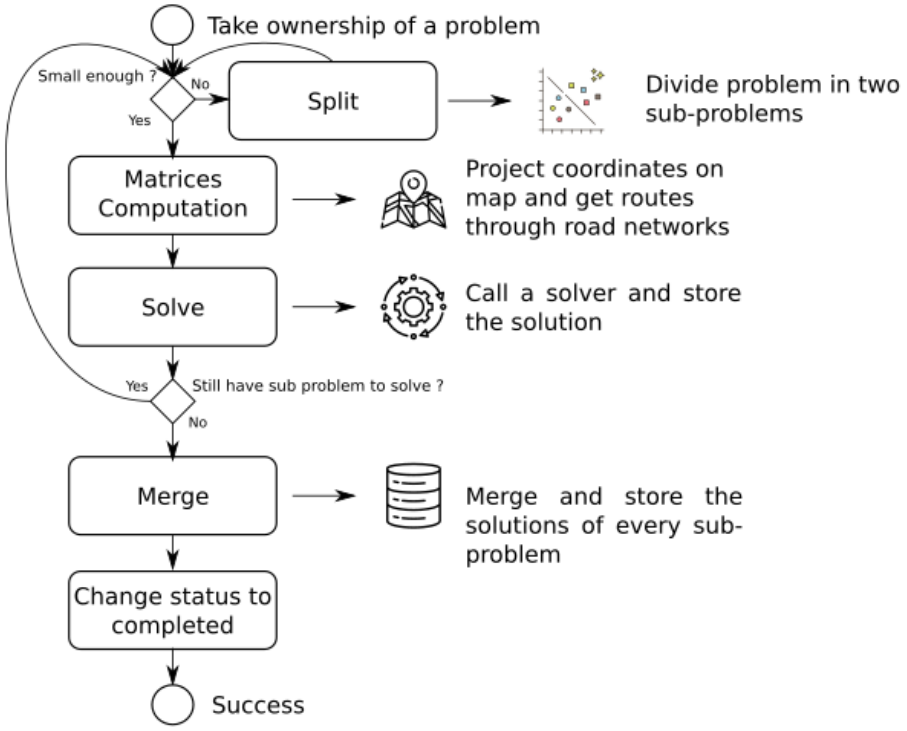

Figure 9: Worker process

use. Verso developp its own solver VROOM which is distributed with an Open Source License as well as the API.

Grapphopper didn't explicitly mention the solver used by their API, nevertheless they develop an Open Source solver called Jsprit. The Open Source version of this solver didn't embed all the features present in the Graphhopper's API. Mapotempo's Optimizer API use a combination of solvers, currently OR-Tools and VROOM, which are both distributed as Open Source solvers. The API is distributed with an Open Source License. The platform developed by

INRIA is not available for commercial purpose and only adapt to academic instances. It use a combination of various methods including the use of the solver CPLEX and BapCod which is developed by the research team R. Sadykov belongs to.

Every project presented here use the REST standard to expose their API. The resolution requests are performed asynchronously except for Verso, which has the particularity to build its solver with a focus to provide good solutions in very short amount of time. Among the list of the APIs presented, the commercial solutions didn't provide proof of optimality. This segment is reserved to the academic API of INRIA.

As mentioned in section 4.4 integrating the geocoding step directly within the resolution segment could be dangerous. Nevertheless, Grapphopper and Routific provide directly this feature with their Vehicle Routing API.

Every commercial solution provide a projection of the problem to solve on the road network to provide to provide route duration. This feature is not present in the INRIA API. Indeed the academic instances use a specified distance metric, which is often an Euclidean or Manhattan distance. Sometimes, the distance matrix is provided with the instance. This is comprehensible as the road network and the legislation applied on may change. In a commercial context, it may be hard to ask the final user to provide an entire matrix.

Note that, the time matrix is often sufficient to solve a VRP. But it many cases, both time and distance matrices are needed. The working time of drivers is regulated by the law, then reducing this value is not an objective in itself but becomes a constraint. The real goal is to reduce the total distance and eventually the greenhouse gas emissions. In these cases, the time and distance matrices at the same time are mandatory. This features is provided by Graphhopper and Mapotempo. In addition if this feature Graphhopper and Mapotempo provide distinct matrices for heterogeneous fleet because trucks, cars or cycles are subject to distinct road laws.

The column Initial presents the capability of taking into account an initial solution or state as a starting point for the resolution. The column Relation

indicate the capability to link nodes together, it could be by putting nodes in the same route, in a given order or a given sequence. "P&D" indicates the possibility to define a link between two nodes, one pickup and one delivery. This refers to the Pickup and Delivery VRP (PDVRP) variant.

Table 4 displays the coverage of the various projects selected on some variants of the VRP. The variants not introduced until now are the Capacitated VRP (CVRP), VRP with Lunch Break (VRPLB), Orienteering Problem (OP), Periodic VRP (PVRP), VRP with Semi-Soft Time Windows (VRPSSTW). The column skills doesn't correspond to a proper variant, as it could be represented as a capacity dimension with a maximum value of zero on certain vehicles and a infinite capacity on others. This attribute displays the ability of given vehicles to visit a set of nodes.

7.2. Numerical results

Table 5 summarizes the results obtained using Optimizer API and a generic model implemented with OR-Tools v7.5 on various instance sets from the literature. Theses variants of the VRP only consider the case where the vehicles starts and finish their routes from and to their depot. The distances are calculated using an euclidean distance. The maximum resolution time of Optimizer-API has been fixed to 10 minutes, except for MDVRPTW, with a time limit set to 1 hour. Note that, Optimizer-API embeds a mechanism to interrupt resolutions which didn't find any improvement within a time equivalent to a multiple of the time spend to find the current best solution.

The results presented here only represent a small subset of the capabilites provided by Optimizer API. Nevertheless, these results are the best way to exhibit the performances of the system. Indeed, these problems have been largely studied. The detailed results are put in appendix 7.

The generic model gives good solution for small instances.Furthermore, the gap increase for bigger instances, but as the time to obtain these solution is strictly limited, the gap remains contained.

The configuration used to solve the instances is set such that above 400

Variant	Source	Instance set	Size	Avg. Time	Avg. Gap
HFVRP	(Duhamel et al., 2011)	DLP	20-95	289,22	2,63 %
HFVRP	(Duhamel et al., 2011)	DLP	102-150	306,48	5,65 %
HFVRP	(Duhamel et al., 2011)	DLP	152-196	369,12	7,22 %
HFVRP	(Duhamel et al., 2011)	DLP	203-256	419,77	7,63 %
MDVRP	(Cordeau et al., 1997)	p & pr	50-100	281,16	0,51 %
MDVRP	(Cordeau et al., 1997)	p & pr	140-360	276,96	5,81 %
MDVRPTW	(Cordeau et al., 2001)	p & pr	48-144	261,74	1,49 %
MDVRPTW	(Cordeau et al., 2001)	p & pr	192-288	1145,53	6,28 %
MDVRPTW	(Vidal et al., 2013a)	pr+	360-960	187,24	14,15 %
VRPTW	(Gehring & Homberger, 1999)	C_1_2	200	243	3,06 %
VRPTW	(Gehring & Homberger, 1999)	C_2_2	200	272	3,52 %
VRPTW	(Gehring & Homberger, 1999)	R1_2	200	340	7,40 %
VRPTW	(Gehring & Homberger, 1999)	R2_2	200	414	6,68 %
VRPTW	(Gehring & Homberger, 1999)	RC1_2	200	262	7,84 %
VRPTW	(Gehring & Homberger, 1999)	RC2_2	200	414	5,75 %
VRPTW	(Gehring & Homberger, 1999)	C1_4	400	205	8,28 %
VRPTW	(Gehring & Homberger, 1999)	C2_4	400	182	11,59 %
VRPTW	(Gehring & Homberger, 1999)	R1_4	400	160	14,26 %
VRPTW	(Gehring & Homberger, 1999)	R2_4	400	168	16,47 %
VRPTW	(Gehring & Homberger, 1999)	RC1_4	400	112	13,57 %
VRPTW	(Gehring & Homberger, 1999)	RC2_4	400	148	13,14 %

Table 5: Gap to the State-of-the-Art

nodes, the problem has to be split in two parts until each sub problem reach a size below this limit. Indeed, this can lead to make the optimal solution unreachable. Nevertheless, it allows in a time constrained situation to obtain a solution of good quality within a strictly limited amount of time.

8. Discussions

The implementation of Web Services dedicated to decision support systems is a great step to democratize operation research algorithms. As such problems are resource and time consuming to solve, delegate such operations are both beneficial for the end-user as its machine will remain entirely free and responsive for others tasks. And for the service provider as the quality of the result will

not rely on the user's device computation performances. Moreover, the service provider is able to associate the most fitted resources to reduce the computation time and the cost required for a single optimization.

Such a solution could be used directly by end users through the Web App. Nevertheless, such a tool is oriented and will only expose a subset of the entire solving capability of the optimization layer. Indeed, providing a generic user interface perfectly fitted to any segment of the supply-chain is particularly difficult as any ergonomic choice is biased and designed for a few use cases. Providing a direct access to the optimization layer makes the optimization independent from the user interface which could be delegated to external entities such as software publishers specialized on a particular field of the last-mile.

Decoupling user interface from a particular module of such application by providing independent software brick makes the particular elements more flexible to integrate new features. By the way, the flexibility is precious in the context of the last-mile as the environment and the constraints evolve quickly. Such architecture divided into multiple services allows to change data providers and technical bricks providers without changing the whole stack. The *Geocoder* is relatively stable in time, as the address conventions are widely diffused, but the data sources uses multiple formats. The *Router* as the transportation networks evolve quickly, that is particularly the case with the progressive release of the public transport network data and the growth of smart cities.

9. Conclusion and Future Work

This article proposes an API able to support as input a large variety of VRP variants with many attributes. The current implementation interprets the input and distinguishes wherever the problem sent is valid or not. Once validated, the problem is prepared and eventually split, this has for consequence to lost the reachability of the optimal solution. But, as the split happens when the problems are large and the resolution time is relatively small, this optimal solution is anyway hard to reach. Optimizer API is designed to orient the problem or

sub-problems to call the most efficient solver depending on the case met, currently OR-Tools and VROOM are used. Other solvers have been integrated, but some didn't met the expected quality of result or simply the current license doesn't allow to use them in a commercial context.

Even if our OR-Tools model is sufficiently flexible to tolerate a large variety of attributes, it shows its limits regarding the State-of-the-Art methods with a tight gap. The future work can be divided in multiple categories.

Currently, some heuristic logic, such the split mechanism, are implemented within the Back-End, their performance is currently under investigation and should be improved in a close future. The split logic is part of it, and is currently extracted to build a new micro-service. This makes the clustering logic developped for the Vehicle Routing context available directly for Mapotempo Web or from any other project without formulating an complete VRP problem.

On the solver side, the OR-Tools model must be edited to improve the performance of the calls to the solver. Some operators allowing to define relations between vehicles are absent and for example makes it hard to solve directly the PVRP efficiently.

The decision support system in its entirety gives both access to bricks of various levels from the basics (Routing, Geocoding) to more sophisticated ones (Optimizer, Fleet, Mapotempo Web) towards the preparation, visualization, resolution and realization of the VRP. Particularly, the system has been designed to tackle daily organization of the last mile delivery, pickup or service. The APIs are used at multiple steps of the decision chain. Among others, it is used as an investigation tool to explore deep changes of the current process for the last mile delivery for various actors of the 3 PL (Third Party-Logistic). It is also used to integrate external flows without increasing the exploitation cost, particularly for the Newspaper delivery. Some companies use the APIs for appointment booking. Some others use the decision system to build their strategic visit planning over several months with regular visits to their customers.

From an OR point of view, the decision support system certainly covers a large set of variants for the static resolution of the VRP. But that's not the only

problem which can be met in the context of the last mile. The resolution of the VRP in a dynamic environment such as the Time Dependent VRP (TDVRP) or the Dial A Ride Problem (DARP). Other problems related, such as the Location Routing Problem (LRP) are not currently covered. Nevertheless, the basic bricks able to provide data to solve these problems are available through the APIs. The Vehicle Routing and Truck Driver Scheduling Problem (VRTDSP) to comply with the regulation laws about the working hours of truck drivers is as well under investigation. But since the main target field is the last mile of the supply chain, the driver work days are subject to few simplification in the model.

The resolution of each user request is currently performed with a single thread. Nevertheless, some resolution methods allow parallel processing. For example, the large problems can be split into multiple subparts which can be processed in parallel.

The OR community has lately developed a big interest to the Machine Learning (ML) tools. Performance improvements may come from the learning of efficient solution patterns. Moreover, the user usually edits manually the solution provided by the system, or the driver chose to switch some parts of the route. Such decisions could be also learned over regular patterns to make the resolution fitting the user preferences make the resolution met user preferences.

At the architecture level, various problems are not yet addressed. For example, the load balancing is currently not taken into account. The system is limited to basic DNS balancing rules which redirect the users to a server depending on their geographic position. Nevertheless, this geographic assignment may cause some problems in term of replication of the data through the multiple nodes of the decision support system. A wise system of balancing is therefore one of the key for large adoption of this technological stack.

References

Almarimi, N., Ouni, A., Bouktif, S., Mkaouer, M. W., Kula, R. G., & Saied, M. A. (2019). Web service API recommendation for automated mashup creation using multi-objective evolutionary search. *Applied Soft Computing Journal*, *82*, 1–13. doi:10.1016/j.asoc.2019.105830.

Arnold, F., Gendreau, M., & Sörensen, K. (2017). Efficiently Solving Very Large Scale Routing Problems, .

Caceres-Cruz, J., Arias, P., Guimarans, D., Riera, D., & Juan, A. A. (2014). Rich vehicle routing problem: Survey. *ACM Computing Surveys*, *47*. doi:10.1145/2666003.

Cardenas, I., Borbon-Galvez, Y., Verlinden, T., Van de Voorde, E., Vanelslander, T., & Dewulf, W. (2017). City logistics, urban goods distribution and last mile delivery and collection. *Competition and Regulation in Network Industries*, *18*, 22–43. doi:10.1177/1783591717736505.

Chen, L., Ayanso, A., & Lertwachara, K. (2018). Performance Impacts of Web-Enabled Retail Services: An Empirical Study. *Journal of Computer Information Systems*, *58*, 301–311. doi:10.1080/08874417.2016.1249536.

Choi, J., Nazareth, D. L., & Ngo-Ye, T. L. (2018). The Effect of Innovation Characteristics on Cloud Computing Diffusion. *Journal of Computer Information Systems*, *58*, 325–333. doi:10.1080/08874417.2016.1261377.

Cohen, M. D., Kelly, C. B., & Medaglia, A. L. (2001). Decision support with Web-enabled software. *Interfaces*, *31*, 109–129. doi:10.1287/inte.31.2.109.10625.

Cordeau, J. F., & Laporte, G. (2001). A tabu search algorithm for the site dependent vehicle routing problem with time windows. *INFOR*, *39*, 292–298. doi:10.1080/03155986.2001.11732443.

Cordeau, J.-F., Laporte, G., & Mercier, A. (2001). *A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows*. Technical Report 8. URL: https://www.jstor.org/stable/822953?seq=1&cid=pdf-.

Cordeau, J.-F. O., Gendreau, M., & Laporte, G. (1997). A Tabu Search Heuristic for Periodic and Multi-Depot Vehicle Routing Problems. *Networks*, *30*, 105–119.

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, *6*, 80–91. doi:https://doi.org/10.1287/mnsc.6.1.80.

Duhamel, C., Lacomme, P., & Prodhon, C. (2011). Efficient frameworks for greedy split and new depth first search split procedures for routing problems. *Computers and Operations Research*, *38*, 723–739. doi:10.1016/j.cor.2010.09.010.

Dunkel, J., Fernández, A., Ortiz, R., & Ossowski, S. (2011). Event-driven architecture for decision support in traffic management systems. *Expert Systems with Applications*, *38*, 6530–6539. doi:10.1016/j.eswa.2010.11.087.

Gehring, H., & Homberger, J. (1999). *A Parallel Hybrid Evolutionary Metaheuristic for the Vehicle Routing Problem with Time Windows*. Technical Report.

Heilig, L., Lalla-Ruiz, E., & Voß, S. (2020). Modeling and solving cloud service purchasing in multi-cloud environments. *Expert Systems with Applications*, *147*. doi:10.1016/j.eswa.2019.113165.

Jacobson, D., Brail, G., & Woods, D. (2012). *APIs: A Strategy Guide*. O'Reilly.

Jia, M. H., Chen, Y. Q., Zhang, G. Y., Jiang, P., Zhang, H., & Wang, J. (2018). A web service framework for astronomical remote observation in Antarctica by using satellite link. *Astronomy and Computing*, *24*, 17–24. doi:10.1016/j.ascom.2018.04.005.

Jin, B., Sahni, S., & Shevat, A. (2018). *Designing Web APIs - Building APIs that developers love*. O'Reilly Media Inc, USA.

Karakoc, E., & Senkul, P. (2009). Composing semantic Web services under constraints. *Expert Systems with Applications*, *36*, 11021–11029. doi:10.1016/j.eswa.2009.02.098.

Luo, J., & Chen, M. R. (2014). Multi-phase modified shuffled frog leaping algorithm with extremal optimization for the MDVRP and the MDVRPTW. *Computers and Industrial Engineering*, *72*, 84–97. doi:10.1016/j.cie.2014.03.004.

Macharis, C., & Melo, S. (2011). *City distribution and urban freight transport : multiple perspectives*. Edward Elgar.

Massé, M. (2011). *REST API Design Rulebook*. O'Reilly.

Neumann, A., Laranjeiro, N., & Bernardino, J. (2018). An Analysis of Public REST Web Service APIs. *IEEE Transactions on Services Computing*, *1-1*, 1–14. doi:10.1109/TSC.2018.2847344.

Papazoglou, M. P., Traverso, P., Dustdar, S., & Leymann, F. (2007). Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, *40*, 38–45. doi:10.1109/MC.2007.400.

Parejo, J. A., Segura, S., Fernandez, P., & Ruiz-Cortés, A. (2014). QoS-aware web services composition using GRASP with Path Relinking. *Expert Systems with Applications*, *41*, 4211–4223. doi:10.1016/j.eswa.2013.12.036.

Penna, P. H. V., Subramanian, A., Ochi, L. S., Vidal, T., & Prins, C. (2019). A hybrid heuristic for a broad class of vehicle routing problems with heterogeneous fleet. *Annals of Operations Research*, *273*, 5–74. doi:10.1007/s10479-017-2642-9.

Ramírez, A., Parejo, J. A., Romero, J. R., Segura, S., & Ruiz-Cortés, A. (2017). Evolutionary composition of QoS-aware web services: A many-objective per-

spective. *Expert Systems with Applications*, *72*, 357–370. doi:10.1016/j.eswa.2016.10.047.

Rasley, J., Karanasos, K., Kandula, S., Fonseca, R., Vojnovic, M., & Rao, S. (2016). Efficient queue management for cluster scheduling. In *Proceedings of the 11th European Conference on Computer Systems, EuroSys 2016*. Association for Computing Machinery, Inc. doi:10.1145/2901318.2901354.

Sadykov, R., Uchoa, E., & Pessoa, A. (2020). A Bucket Graph–Based Labeling Algorithm with Application to Vehicle Routing. *Transportation Science*, . doi:10.1287/trsc.2020.0985.

Solomon, M. M. (1987). Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, *35*, 254–265. URL: http://www.jstor.org/stable/170697http://www.jstor.org/http://www.jstor.org/action/showPublisher?publisherCode=informs.

Terroso-Saenz, F., & Muñoz, A. (2020). Land use discovery based on Volunteer Geographic Information classification. *Expert Systems with Applications*, *140*. doi:10.1016/j.eswa.2019.112892.

Tihomirovs, J., & Grabis, J. (2017). Comparison of SOAP and REST Based Web Services Using Software Evaluation Metrics. *Information Technology and Management Science*, *19*. doi:10.1515/itms-2016-0017.

Toth, P., Vigo, D., & Society for Industrial and Applied Mathematics (2014). *Vehicle routing : problems, methods, and applications*.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013a). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers and Operations Research*, *40*, 475–489. doi:10.1016/j.cor.2012.07.018.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013b). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European*

Journal of Operational Research, *231*, 1–21. doi:10.1016/j.ejor.2013.02.053.

Wang, W., Indulska, M., & Sadiq, S. (2018). Guidelines for Business Rule Modeling Decisions. *Journal of Computer Information Systems*, *58*, 363–373. doi:10.1080/08874417.2017.1285683.

Welch, P. (2017). *Dynamic vehicle routing problems appear in a number of industries*. Technical Report. URL: www.opendoorlogistics.com.

Zhao, P., Di, L., & Yu, G. (2012). Building asynchronous geospatial processing workflows with web services. *Computers and Geosciences*, *39*, 34–41. doi:10.1016/j.cageo.2011.06.006.

	Vidal et al.(2013)	Luo & Chen(2014)	Penna et al.(2019)	Sadykov et al.(2020)	Our proposal
CPU	Opteron 2755 2.2GHz	Pentium 4 2.8 GHz	i7 2.93 GHz	E5-2680 v3 2.50 GHz	i7-7700HQ 2.80GHz
OS			Ubuntu 14.04		Linux Mint 20
Language	C++	C++	C++	C++	C++
Rating	445	502	1394	1810	2069
Speed factor	4,65	4,12	1,48	1,14	1

Table 6: Relative performances of computers

Detailed results

In all the tables, the times are in seconds. n is the number of nodes to serve, m is the number of vehicles and d is the number of depots. Note that in the case of multi depot instances, there are m vehicles per depot and in the case of heterogeneous fleet, m represents the number of vehicle configurations in term of cost and capacity. the results in bold have comes with a proof of optimality.

The VRPTW variant, as shown in Table 13, has been tested both with the generic model and the dedicated method provided by VROOM through Optimizer-API. On every instance the dedicated method has a smaller gap within a smaller amount of time, the generic model has some counterparts, which makes it slower to converge. Nevertheless, the resolution time is strictly limited and this constraint is essential to make the solution usable in practice. This point has to be put in perspective to compare the results provided by Sadykov et al. (2020) where the time limit is set to 60 hours.

				Generic model			Vidal et al.(2013)		BKS
instance	n	m	d	Cost	Time	Gap	Cost	Time	Cost
pr01	48	2	4	1074,12	226,83	0,00 %	1074,12	18,6	1074,12
pr02	96	3	4	1769,48	169	0,41 %	1762,61	69	1762,21
pr03	144	4	4	2397,05	378	0,99 %	2373,65	105	2373,65
pr04	192	5	4	2950,71	273	4,82 %	2815,11	353,4	2815,11
pr05	240	6	4	3181,81	194	7,41 %	2962,25	520,8	2962,25
pr06	288	7	4	3772,82	1 572	5,13 %	3588,78	805,8	3588,78
pr07	72	2	6	1418,22	222	0,00 %	1418,22	30,6	1418,22
pr08	144	3	6	2156,99	186	2,87 %	2096,73	143,4	2096,73
pr09	216	4	6	2803,44	416	3,35 %	2712,56	312	2712,56
pr10	288	5	6	3780,09	585	9,10 %	3464,65	913,2	3464,65
pr11	48	2	4	1018,02	226	1,22 %	1005,73	12,6	1005,73
pr12	96	3	4	1526,22	327	4,21 %	1464,5	100,8	1464,5
pr13	144	4	4	2017,34	321	0,78 %	2001,81	176,4	2001,81
pr14	192	5	4	2391,65	279	8,94 %	2195,33	393	2195,33
pr15	240	6	4	2549,73	1 686	4,79 %	2433,15	753,6	2433,15
pr16	288	7	4	2962,37	2 614	4,43 %	2836,67	958,2	2836,67
pr17	72	2	6	1245,53	209	0,75 %	1236,24	63	1236,24
pr18	144	3	6	1853,23	352	3,64 %	1788,18	198	1788,18
pr19	216	4	6	2299,76	2 309	1,89 %	2257,13	515,4	2257,13
pr20	288	5	6	3325,73	654	11,45 %	2984,01	1330,8	2984,01

Table 7: MDVRPTW

				Generic model			Vidal et al.(2013)		BKS
instance	n	m	d	Cost	Time	Gap	Cost	Time	Cost
pr11a	360	10	4	7782,65	157	15,80 %	6720,71	1008,6	6720,71
pr12a	480	13	4	9184,03	1003	12,28 %	8179,8	1800	8179,8
pr13a	600	16	4	11399,13	121	17,92 %	9667,2	3291	9667,2
pr14a	720	19	4	12994,96	113	16,82 %	11124,01	3939	11124,01
pr15a	840	22	4	15388,74	69	18,25 %	13013,97	7946,4	13013,97
pr16a	960	26	4	16619,25	46	16,22 %	14299,87	8017,8	14299,87
pr17a	360	7	6	7189,93	122	14,05 %	6304,3	1033,8	6304,3
pr18a	520	10	6	9383,69	121	12,94 %	8308,32	2655	8308,32
pr19a	700	13	6	12710,76	66	19,04 %	10677,61	4465,2	10677,61
pr20a	880	16	6	13978,87	55	16,84 %	11963,91	6442,2	11963,91
pr21a	420	4	12	7008,12	178	11,94 %	6260,53	1680	6260,53
pr22a	600	6	12	9194,61	40	15,14 %	7985,37	4563	7985,37
pr23a	780	8	12	11883,46	144	19,58 %	9937,43	8263,2	9937,43
pr24a	960	10	12	13849,1	38	16,15 %	11923,72	11830,2	11923,72
pr11b	360	8	4	5273,83	969	8,98 %	4839,44	1082,4	4839,44
pr12b	480	11	4	6975,09	50	15,04 %	6063,26	1745,4	6063,26
pr13b	600	14	4	8298,58	287	14,40 %	7254,17	4259,4	7254,17
pr14b	720	17	4	9765,87	39	11,84 %	8732,29	5935,2	8732,29
pr15b	840	20	4	11721,77	288	12,29 %	10438,72	7768,8	10438,72
pr16b	960	23	4	12889,59	28	12,25 %	11483,22	10218,6	11483,22
pr17b	360	6	6	5242,77	819	9,09 %	4806,01	946,8	4806,01
pr18b	520	9	6	7213,34	62	10,54 %	6525,72	2367	6525,72
pr19b	700	12	6	9445,98	67	14,81 %	8227,25	4833	8227,25
pr20b	880	15	6	11582,66	16	12,17 %	10325,8	9044,4	10325,8
pr21b	420	4	12	5514,85	109	13,32 %	4866,57	2205	4866,57
pr22b	600	6	12	7354,25	23	13,34 %	6488,5	4395,6	6488,5
pr23b	780	7	12	9663,99	166	13,38 %	8523,41	9839,4	8523,41
pr24b	960	8	12	12154,9	47	11,92 %	10860,08	17910,6	10860,08

Table 8: MDVRPTW

				Generic model			Sadykov et al.(2020)		Luo & Chen(2014)	
instance	n	m	d	Cost	Time	Gap	Cost	Time	Cost	Time
p01	50	4	4	576,87	286	0,00 %			576,87	11,4
p02	50	4	4	473,53	211	0,00 %			473,53	19,2
p03	75	2	5	641,18	228	0,00 %			641,18	29,4
p04	100	2	2	1008,98	560	0,79 %			1001,04	34,2
p05	100	2	2	753,86	156	0,51 %			750,03	51,6
p06	100	3	3	893,55	458	1,94 %			876,5	39,6
p07	100	4	4	904,37	448	2,54 %			881,97	30
p08	249	2	2	4746,75	187	8,55 %	**4372,78**	20580	4372,78	129
p09	249	3	3	4189,98	292	8,59 %	**3858,66**	8700	3860,28	127,2
p10	249	4	4	3931,27	195	8,27 %	**3631,11**	2506	3634,76	125,4
p11	249	5	5	3789,80	240	6,87 %	**3546,06**	2986	3546,06	130,8
p12	80	2	2	1318,95	197	0,00 %			1318,95	36,6
p13	80	2	2	1318,95	200	0,00 %	**1318,95**	1	1318,95	38,4
p14	80	2	2	1360,11	252	0,00 %	**1360,12**	1	1360,12	39
p15	160	4	4	2568,72	169	2,53 %			2505,42	112,2
p16	160	4	4	2572,23	362	0,00 %	**2572,23**	1	2572,23	110,4
p17	160	4	4	2709,09	197	0,00 %	**2709,09**	55	2709,09	111
p18	240	6	6	3890,24	510	5,06 %			3702,85	127,2
p19	240	6	6	3827,06	147	0,00 %	**3827,06**	85	3827,06	129
p20	240	6	6	4147,93	160	2,21 %	**4058,07**	150	4058,07	129,6
p21	360	9	9	6011,64	487	9,80 %			5474,84	186,6
p22	360	9	9	5760,71	152	1,03 %	**5702,16**	148	5702,16	188,4
p23	360	9	9	6224,16	85	2,39 %	**6078,75**	352	6078,75	184,8
pr01	48	4	4	861,32	296	0,00 %	**861,32**	3	861,32	17,4
pr02	96	4	4	1318,77	166	0,87 %	**1307,34**	258	1307,34	36
pr03	144	4	4	1827,41	370	1,31 %	**1803,8**	622	1803,8	58,8
pr04	192	4	4	2131,08	273	3,54 %	**2058,31**	1372	2058,31	86,4
pr05	240	4	4	2590,69	230	11,13 %	**2331,2**	22740	2331,2	144,6
pr06	288	4	4	3074,22	600	14,87 %	**2676,3**	23940	2677,64	151,2
pr07	72	6	6	1089,56	198	0,00 %	**1089,56**	17	1089,56	29,4
pr08	144	6	6	1696,36	363	1,89 %	**1664,85**	522	1664,85	82,8
pr09	216	6	6	2285,21	295	7,13 %	**2133,2**	1386	2133,2	124,8
pr10	288	6	6	3468,46	227	20,97 %	**2867,26**	86760	2868,26	160,8

Table 9: MDVRP

			Generic model			Sadykov et al.(2020)		Penna et al.(2019)	
instance	n	m	Cost	Time	Gap	Cost	Time	Cost	Time
HVRP_DLP_75	20	3	452,85	161	0,00 %	**452,85**	0	452,85	246,66
HVRP_DLP_92	35	3	564,39	141	0,00 %	**564,39**	579	564,39	672,65
HVRP_DLP_93	39	6	1043,81	222	0,66 %	**1036,99**	99	1036,99	20,63
HVRP_DLP_94	46	5	1386,11	222	0,57 %	**1378,25**	17	1378,25	27,39
HVRP_DLP_55	56	3	10351,23	283	1,04 %	**10244,34**	38	10244,34	565,12
HVRP_DLP_52	59	3	4071,67	260	1,10 %	**4027,27**	777	4027,27	315,6
HVRP_DLP_08	69	4	4685,10	187	2,03 %	**4591,75**	2	4591,75	306,09
HVRP_DLP_39	77	5	3050,49	230	4,51 %	**2918,87**	1727	2921,36	421,5
HVRP_DLP_70	78	4	6878,84	561	2,91 %	**6684,56**	833	6684,56	205,32
HVRP_DLP_82	79	3	4787,30	404	1,46 %	**4718,27**	3629	4766,74	83,71
HVRP_DLP_06	84	3	12495,16	384	6,95 %	**11682,98**	157	11688,64	488,63
HVRP_DLP_36	85	6	5867,54	473	3,22 %	**5684,62**	89	5684,62	811,07
HVRP_DLP_43	86	7	9349,40	271	7,01 %	**8737,02**	43	8737,02	966,84
HVRP_DLP_01	92	4	9500,94	168	3,16 %	**9210,14**	1477	9210,14	52,29
HVRP_DLP_09	95	4	7970,43	371	4,88 %	**7599,72**	8160	7603,38	304,85
HVRP_DLP_90	102	4	2397,00	449	5,22 %	**2278,05**	145740	2346,43	245,63
HVRP_DLP_15	105	3	8534,58	425	3,81 %	8221	216000	8260,65	361,72
HVRP_DLP_84	105	4	7494,26	207	3,69 %	**7227,88**	1261	7233,95	332,47
HVRP_DLP_81	106	4	10927,08	150	3,25 %	**10583,6**	20160	10693,7	410,38
HVRP_DLP_29	107	6	10034,02	582	9,88 %	**9132,03**	8280	9142,86	343,06
HVRP_DLP_05	108	4	12133,20	172	11,63 %	**10869,04**	29	10876,48	667,11
HVRP_DLP_87	108	4	3862,86	186	2,90 %	**3753,87**	1791	3753,87	440,02
HVRP_DLP_47	111	5	17147,10	204	6,13 %	**16156,12**	349	16206,88	1475,05
HVRP_DLP_48	111	5	22285,34	153	4,84 %	**21257,38**	1705	21320,3	333,85
HVRP_DLP_61	111	3	7433,66	294	1,94 %	7293	216000	7292,03	444,32
HVRP_DLP_10	112	4	2135,95	159	1,35 %	**2107,55**	1118	2107,55	319,39
HVRP_DLP_30	112	3	6532,79	263	4,05 %	**6278,62**	113340	6313,5	122,02
HVRP_DLP_28	113	6	5865,48	316	6,15 %	5525,9	216000	5530,55	995,85
HVRP_DLP_53	115	3	6669,79	350	3,65 %	**6434,83**	7214	6434,83	39,97
HVRP_DLP_03	116	5	11443,08	268	6,85 %	**10709,66**	73	10709,66	303,14
HVRP_DLP_11	119	5	3499,96	163	3,94 %	**3367,41**	1254	3367,41	24,83
HVRP_DLP_04	121	8	11175,02	218	4,29 %	**10714,84**	49980	10748,17	512,1
HVRP_DLP_2A	124	4	8126,12	199	4,27 %	**7793,16**	102	7793,16	689,81
HVRP_DLP_83	124	4	10226,64	232	2,25 %	**10001,8**	3117	10020,07	144,51
HVRP_DLP_68	125	4	9780,64	279	10,03 %	**8889,03**	10800	8970,63	506,65
HVRP_DLP_74	125	5	11987,94	237	3,46 %	**11586,58**	258	11586,58	598,34

Table 10: HFVRP - with customers up to 125 customers

			Generic model			Sadykov et al.(2020)		Penna et al.(2019)	
instance	n	m	Cost	Time	Gap	Cost	Time	Cost	Time
HVRP_DLP_18	126	3	11538,86	344	19,59 %	9649,05	216000	9655,91	172,82
HVRP_DLP_24	126	5	9536,07	539	4,77 %	9102	216000	9126,55	802,3
HVRP_DLP_88	127	5	13521,94	456	9,17 %	**12385,74**	29	12388,23	104,11
HVRP_DLP_14	129	6	5988,9	178	6,19 %	**5639,98**	216000	5663,91	303,37
HVRP_DLP_51	129	3	7999,62	266	3,60 %	**7721,47**	1757	7721,47	365,46
HVRP_DLP_31	131	8	4256,59	523	4,03 %	**4091,52**	1104	4091,52	201,39
HVRP_DLP_40	132	5	11944,70	161	8,04 %	**11056,13**	10740	11122,32	182,11
HVRP_DLP_89	134	5	7485,47	151	5,62 %	7087	216000	7091,99	632,22
HVRP_DLP_41	135	7	8436,93	318	11,33 %	7598	216000	7578,53	614,92
HVRP_DLP_34	136	6	6051,13	156	5,44 %	**5739,02**	4200	5751,05	606,39
HVRP_DLP_60	137	4	17811,19	362	4,70 %	**17012,42**	2678	17037,23	476,61
HVRP_DLP_73	137	5	10599,05	304	3,96 %	**10195,33**	127	10195,33	458,28
HVRP_DLP_26	141	5	6794,61	498	6,06 %	**6406,16**	36600	6433,23	518,28
HVRP_DLP_23	143	6	8073,52	516	4,28 %	7742	216000	7760,01	835,87
HVRP_DLP_85	146	4	9130,04	586	4,18 %	**8763,9**	9600	8795,31	206,41
HVRP_DLP_79	147	4	7602,15	352	4,74 %	**7257,97**	4380	7262,91	439,7
HVRP_DLP_66	150	4	13463,87	428	5,38 %	12776,24	216000	12783,94	635,64
HVRP_DLP_69	152	4	9736,01	174	6,67 %	**9127,16**	12960	9169,42	269,63
HVRP_DLP_76	152	8	12792,41	440	6,65 %	11994,22	216000	11994,4	0,02
HVRP_DLP_56	153	4	32750,59	159	5,97 %	**30905,95**	211800	31030,19	190,76
HVRP_DLP_86	153	5	9269,94	453	2,76 %	**9020,63**	31500	9038,03	382,89
HVRP_DLP_37	161	5	7329,06	453	7,17 %	6838,72	216000	6850,77	104,39
HVRP_DLP_64	161	3	17549,35	171	2,42 %	17135,16	216000	17135,16	253,1
HVRP_DLP_22	163	4	14012,85	574	7,38 %	13050	216000	13096,26	330,23
HVRP_DLP_57	163	4	47560,66	180	6,21 %	44782	216000	44781,64	339,08
HVRP_DLP_27	164	4	9376,92	593	11,33 %	8423	216000	8424,73	350,71
HVRP_DLP_07	167	5	8835,77	271	9,46 %	**8071,97**	727	8089,46	367,91
HVRP_DLP_35	168	6	10503,5	285	10,30 %	**9522,45**	197	9555,92	405,62
HVRP_DLP_45	170	3	11120,95	544	6,15 %	10477	216000	10476,25	744,39

Table 11: HFVRP - with customers above 125 and up to 170

			Generic model			Sadykov et al.(2020)		Penna et al.(2019)	
instance	n	m	Cost	Time	Gap	Cost	Time	Cost	Time
HVRP_DLP_80	171	3	7136,03	274	4,68 %	**6816,89**	47220	6826,38	473,69
HVRP_DLP_44	172	3	13545,65	503	11,10 %	12192	216000	12208,3	219,91
HVRP_DLP_54	172	4	11047,78	364	6,72 %	10352	216000	10351,97	418,83
HVRP_DLP_67	172	5	11514,79	547	6,89 %	**10772,81**	82800	10884,91	442,89
HVRP_DLP_63	174	5	22103,29	254	11,12 %	**19890,65**	19440	20075,44	828,76
HVRP_DLP_12	176	4	3591,14	264	1,33 %	**3543,99**	5940	3543,99	264,61
HVRP_DLP_42	178	7	12230,99	199	13,19 %	10805,94	216000	10817,9	325,8
HVRP_DLP_02	181	4	12571,18	538	7,91 %	**11649,81**	574	11675,26	52,29
HVRP_DLP_2B	183	4	8900,84	426	5,18 %	8463	216000	8462,56	298,92
HVRP_DLP_95	183	2	6472,84	342	4,90 %	6170,2	216000	6175,62	15,68
HVRP_DLP_71	186	3	11123,87	371	13,53 %	**9798,06**	14880	9892,58	120
HVRP_DLP_72	186	4	6399,40	507	9,01 %	**5870,43**	113460	5917	389,13
HVRP_DLP_50	187	6	12940,14	256	4,60 %	12371	216000	12370,94	990,34
HVRP_DLP_13	188	7	7012,21	326	4,72 %	6696,43	216000	6696,43	71,46
HVRP_DLP_33	189	7	10081,43	419	8,01 %	9333,39	216000	9410,99	1131,44
HVRP_DLP_77	190	3	7794,23	143	12,68 %	6917	216000	6933,15	426,51
HVRP_DLP_78	190	4	7411,51	463	5,34 %	7036	216000	7045,3	278,69
HVRP_DLP_59	193	6	14896,51	447	4,30 %	14283	216000	14304,46	1028,25
HVRP_DLP_91	196	4	6729,05	505	6,00 %	6348,36	216000	6390,24	15,36
HVRP_DLP_21	203	4	5309,10	577	3,29 %	**5139,84**	1505	5139,84	1009,87
HVRP_DLP_38	205	5	11798,73	366	5,40 %	11195	216000	11194,68	571,37
HVRP_DLP_25	220	5	7636,11	505	5,95 %	7207	216000	7209,29	443,1
HVRP_DLP_58	220	6	25367,06	573	8,54 %	23371	216000	23370,42	471,94
HVRP_DLP_65	223	3	14096,86	522	8,08 %	13044	216000	13043,54	70,38
HVRP_DLP_17	224	5	5531,85	205	3,15 %	**5362,83**	210	5362,83	180,91
HVRP_DLP_62	225	5	26565,24	396	15,57 %	22987	216000	23010,35	108,21
HVRP_DLP_19	239	2	13320,69	547	13,98 %	11687	216000	11686,39	1216,1
HVRP_DLP_32	244	8	10288,08	591	9,65 %	9383	216000	9382,6	308,39
HVRP_DLP_49	246	8	17815,52	292	10,10 %	16182	216000	16181,17	371,3
HVRP_DLP_46	250	5	26217,5	280	6,72 %	24567	216000	24566,23	415,02
HVRP_DLP_16	256	5	4204,03	184	1,13 %	4157	216000	4156,97	905,21

Table 12: HFVRP - with customers above 170

	Generic model					Dedicated method - VROOM					Sadykov et al.(2020)
Instance Set	Time	Avg	Med	Min	Max	Time	Avg	Med	Min	Max	Time
C_1_2	243	3,06 %	3,09 %	0,21 %	9,02 %	3,3	0,41 %	0,29 %	0,22 %	0,97 %	117,7
C_2_2	272	3,52 %	3,32 %	1,31 %	6,08 %	3,3	1,03 %	0,63 %	0,38 %	3,36 %	25895,7
C1_4	205	8,28 %	8,20 %	0,16 %	14,09 %	24,8	0,71 %	0,25 %	0,19 %	2,06 %	1671
C2_4	182	11,59 %	9,63 %	5,65 %	24,92 %	22,6	4,28 %	3,44 %	2,01 %	11,40 %	93302,9
R1_2	340	7,40 %	6,37 %	2,14 %	15,36 %	5,3	3,03 %	3,04 %	1,25 %	4,59 %	11461,7
R1_4	160	14,26 %	13,55 %	9,08 %	19,37 %	32,5	4,85 %	5,10 %	2,85 %	6,16 %	194500,8
R2_2	414	6,68 %	6,51 %	3,55 %	12,29 %	6,0	7,16 %	7,38 %	3,72 %	9,90 %	7238
R2_4	168	16,47 %	15,77 %	12,50 %	19,49 %	29,6	11,84 %	12,23 %	7,78 %	14,14 %	153892,2
RC1_2	262	7,84 %	7,42 %	5,30 %	10,27 %	5,8	3,16 %	3,20 %	1,91 %	3,98 %	55485,8
RC1_4	112	13,57 %	13,29 %	11,09 %	17,27 %	30,9	5,05 %	5,35 %	3,75 %	5,98 %	216000
RC2_2	414	5,75 %	4,83 %	2,14 %	11,52 %	5,7	5,01 %	5,10 %	3,41 %	6,62 %	48567,7
RC2_4	148	13,14 %	13,15 %	6,87 %	19,27 %	22,7	7,63 %	7,20 %	4,86 %	10,63 %	166728

Table 13: VRPTW