VRP functionality with VROOM on the database for pgRouting

1. Contact Details

Name: Ashish Kumar.
Country: India.
Email: ashishkr23438 AT gmail DOT com / ashishkumar DOT cse18 AT iitbhu DOT ac DOT in.
Phone: +91-6205144592
Location: Varanasi, India, +5:30 GMT
Github: https://github.com/krashish8
LinkedIn: https://www.linkedin.com/in/ashishkr23438/
Twitter: https://twitter.com/its_ashishkr

2. Title

VRP functionality with VROOM on the database for pgRouting.

3. Brief Project Description

The project aims at implementing the VRP functionality in the vrprouting repository of pgrouting, using VROOM[1] as a library in C++.

VROOM is an open-source optimization engine, that aims at providing good solutions to well-known Vehicle Routing Problems (VRP)[3], such as:

- TSP (Travelling Salesman Problem)
- CVRP (Capacitated VRP)
- VRPTW (VRP with Time Windows)
- MDHVRPTW (Multi-Depot Heterogeneous VRPTW)
- PDPTW (Pickup-and-Delivery Problem with TW)

I propose to implement this functionality as an extension to PostGIS, the PostgreSQL geospatial database, by porting it to vrprouting, so that this functionality can be used in the PostgreSQL database.

4. State of the Project Before GSoC

The vrprouting[2] repository has been recently created by extracting the code of the pgrouting repository involving the VRP functions. It currently contains the functions to solve the Pickup-and-Delivery problem (`vrp_pgr_pickDeliver` and `vrp_pgr_pickDeliverEuclidean`) and the Single Depot VRP problem (`vrp_oneDepot`). However, the VROOM functionality is not yet implemented in vrprouting, which if implemented, can provide good solutions to these problems in an efficient way.

5. Benefits to the Community

Implementing the VROOM functionality as a postgreSQL extension will be very beneficial to those users who want to use the VROOM functionality in the database. VROOM provides a good solution to well-known vehicle routing problems within small computing times, and it can scale to handle very big instances. It is also customizable and supports user-defined cost matrices. Therefore the VROOM functionality in vrprouting can be of great use to the community, as the users can use it to solve real-world vehicle routing problems efficiently.

6. Deliverables

The deliverables at the end of GSoC period are:

- Implementation of VROOM functionality for vrprouting, as a function in postgreSQL database, where the user will provide the JSON structure expected by VROOM, and get the solution of the problem instance in GeoJSON format.
- Self-documented code in SQL, C, C++ with comments, wherever required.
- User's Documentation for the function.
- Basic pgTAP tests to check the parameter types, parameter names, and connection with VROOM.
- Experimentation on the C++20 Modules can be done in this project.

Only if time allows:

- Additional pgTAP tests can be added, such as the server no crash tests and unit tests.
- More specific functions can be made using VROOM, where the user can give the input in the form of SQL query (instead of JSON structure), and get the result as a set of rows (instead of the GeoJSON).

7. Timeline

Community Bonding Period (May 17th - June 7th)

- Set up the development environment.
- Interact with mentors, introduce myself to the community, and actively get involved in the discussion.
- Get familiar with pgRouting's development style and VROOM. Understand expected coding, documentation, and testing standards set by pgRouting.
- Set up the wiki page to keep track of weekly progress.
- Develop a better understanding of PostgreSQL, PostGIS, Pl/pgSQL, and how they interact with pgRouting.
- Learn to create unit tests using pgTap.

First Coding Period (June 7th - July 12th)

Time Period	Hours	Proposed Work
Week 1 (June 7th - June 14th)	17.5	➢ Developing `vrp_vroom()` starts. ➢ Create a basic skeleton for C, C++, SQL code, users documentation and pgTAP tests.
Week 2 (June 14th - June 21st)	17.5	➢ Read data from PostgreSQL. ➢ Transform data to C++ containers suitable for calling the VROOM instance.
Week 3 (June 21st - June 28th)	17.5	➢ Create the scripts for building the OSRM instance and running the OSRM server, on which VROOM runs. ➢ Create the necessary class wrappers for making connections to VROOM.

Week 4 (June 28th - July 5th)	17.5	➢ Make connections to the VROOM instance, and pass the parameters. ➢ Transform the solution computed by VROOM to C containers suitable for passing to PostgreSQL.
Week 5 (July 5th - July 12th)	17.5	➢ Basic testing of the function. ➢ Preparation of the report for the First Coding Period.
Total	**87.5**	

Phase 1 Evaluation (July 12th - July 16th)

- Submit the `pgr_vroom()` function, albeit without pgTAP tests or users documentation.
- Mentors evaluate me and vice-versa.

Second Coding Period (July 12th - Aug 16th)

Time Period	**Hours**	**Proposed Work**
Week 6 (July 12th - July 19th)	17.5	➢ Work on the feedback provided from the Phase 1 Evaluation. ➢ Create the test script and the environment for running the OSRM server before executing the pgTAP tests of the function, and the GitHub Actions script for the same. ➢ Create pgTAP tests to check the connection with the server.
Week 7 (July 19th - July 26th)	17.5	➢ Create pgTAP tests for the function `vrp_vroom()` to check the parameter types, and parameter names. ➢ Fix the bugs, if any, to ensure that the pgTAP tests pass.
Week 8 (July 26th - Aug 2nd)	17.5	➢ Prepare user's documentation. ➢ Create suitable queries using the examples of the VROOM API documentation.
Week 9 (Aug 2nd - Aug 9th)	17.5	➢ Integration to the `develop` branch of the vrpRouting repository (the branch is not yet created)
Week 10 (Aug 9th - Aug 16th)	17.5	➢ Preparation of the final report.
Total	**87.5**	

Total time spent in the entire Coding Period = **175 hours.**

Final Evaluation (August 16th - August 23rd)

- Submit the complete project with all the required functionality, documentation, and unit tests.
- Submit final report and evaluation of mentors.

8. Do you understand this is a serious commitment, equivalent to a full-time paid summer internship or summer job?

Yes, I completely understand that GSoC is a serious commitment, and the expectations from me would be very similar to a full-time paid summer internship or summer job. Therefore, I will try to put my best efforts to make worthy contributions to the pgRouting community during GSoC, and even after that, if time permits. Also, similar to a job, I'd be expected to respond frequently to the daily updates and reports on a weekly basis. I am really excited to work on this project.

9. Do you have any known time conflicts during the official coding period?

I may be involved with academics or the college work such as the Summer Training. However, considering the changes imposed in the GSoC '21 program, these are not 'actual' time conflicts. The GSoC program demands ~17.5 hours per week, which I can easily devote without any time conflicts, therefore I feel myself suitable for it.

10. Studies

10.1 What is your school and degree?

School: Indian Institute of Technology, Banaras Hindu University (BHU), Varanasi
(a.k.a. IIT (BHU) Varanasi).
Degree: Bachelor of Technology in Computer Science and Engineering.

10.2 Would your application contribute to your ongoing studies/degree? If so, how?

Yes, my application will contribute to my ongoing studies and Bachelor's Degree.

I enjoy coding and exploring various algorithms and have previously participated in various algorithmic competitions, so this project will help me get a practical experience of algorithmic coding. It would be an added asset to my ongoing degree and would help me gain a deeper understanding of GIS. Through this, I would be able to explore the VRP functionality of the VROOM, and thus gain a deeper understanding of the VRP algorithms.

11. Programming and GIS

11.1 Computing experience

Operating System: Ubuntu 20.04.1 LTS (Focal) - Used on a Daily Basis, Windows 10.
Programming Languages: C++, C, Python3, Javascript, Java, PHP, Bash.
Databases: PostgreSQL, MySQL, SQLite.
Tools: Git, Jupyter Notebook.
Frameworks: Django, Django Rest Framework, Angular, Vue.js
Relevant Courses Completed: Data Structures and Algorithms, Graph Theory.

11.2 GIS experience as a user

I have used GIS libraries like pgRouting and have used GIS before on a database management project.

11.3 GIS programming and other software programming

- Worked on a database management application written in Django, SQL, HTML/JS/CSS, and another application written in Spring Boot Framework.
- Worked on a Slack Bot app and a backend web app written in Django Rest Framework for a hackathon.
- Worked on a web app written in SQL, PHP, HTML/JS/CSS, which was extensively used in our college for the smooth organization of our college's Sports Fest.
- Worked on a Django REST API project written in Django REST Framework for making the Backend of our college's technical fest, which was used by ~8000 people.
- Public C / C++ related programming / projects:
    - I have a sound knowledge of Data Structures & Algorithms, and have made around 1500+ code submissions on various online competitive programming platforms such as Codeforces, Codechef, HackerRank, Spoj. Submissions also include the graph problems and the usage of the graph algorithms in solving those problems.
    - I maintain GitHub repositories containing my codes submitted for various academic courses at our college.

11.4 Briefly mention and link to former open-source contributions

- GSoC '20: Addition of pgr_depthFirstSearch() to pgRouting [#1348 and #1599]
- GSoC '20: Addition of pgr_sequentialVertexColoring() to pgRouting [#1362 and #1599]
- pgr_withPoints fails when points_sql is empty [#1640, #1641 and #1742]
- pgr_bdAstar fails when source or target vertex does not exist in the graph [#1733 and #1734]
- Addition of Combinations SQL signatures to various functions in pgRouting [#1648]
    - aStar - Family of functions [#1732]
    - Bidirectional A* - Family of functions [#1774]
    - Bidirectional Dijkstra - Family of functions [#1775]
    - Flow - Family of functions [#1777]
    - withPoints - Family of functions [#1649]
    - pgr_bellmanFord(Combinations) [#1779]
    - pgr_binaryBreadthFirstSearch(Combinations) [#1783]
    - pgr_dagShortestPath(Combinations) [#1793]

    - pgr_edwardMoore(Combinations) [#1794]
- Cleaned pgr_pickDeliver code [#1816, #1822 and #1832]
- Add GitHub Actions script for releases [#1855, #1856 and #1857]
- Add GitHub actions script for publishing documentation [#1877, #1878]
- Add GitHub script for releases, and publishing users documentation [#1 on vrprouting]
- Other small PRs and issues created on the pgrouting repository.
- Other open-source contributions from my GitHub Profile.

12. GSoC Participation

12.1 Have you participated in GSoC before?

Yes, I have participated in GSoC before.

12.2 How many times, which year, which project?

Participated once in GSoC ‘20 for pgRouting project, with OSGeo organization.

12.3 Have you submitted / will you submit another proposal for this year's GSoC to a different org?

No.

13. pgRouting Application Requirements

The requirements for applying to pgRouting (under OSGeo) are mentioned here.
The links to the respective issues are:

- Task 1: Intent of Application.
- Task 2: Experience with GitHub & Git. (Link to the Pull Request)
- Task 3: Build locally pgRouting.
- Task 4: Get familiar with C++.
- Task 5: Get familiar with pgRouting.

14. Detailed Proposal

14.1 Proposed Signature

The proposed signature of the function resembles the VROOM API[4].

```
vrp_vroom(VRP JSON [, osrm_host] [, osrm_port] [, plan] [, geometry])

RETURNS GeoJSON with keys:
   code, error, summary, unassigned, routes.
```

14.2 Parameters

14.2.1 Compulsory Parameters

Parameter	Type	Description
VRP JSON	`JSON`	JSON object described in the Inner Query

14.2.1 Optional Parameters

Parameter	Type	Default	Description
osrm_host	`TEXT`	`‘car:0.0.0.0’`	OSRM routing server host in the form of PROFILE:HOST.
osrm_port	`TEXT`	`‘car:5000’`	OSRM routing server port in the form of PROFILE:HOST.
plan	`BOOLEAN`	`FALSE`	Plan mode. Choose ETA for custom routes and report violations. Requires GLPK as a dependency. • All constraints in input implicitly become soft constraints. • The output is a set of routes matching the expected description while minimizing timing violations and reporting all constraint violations.
geometry	`BOOLEAN`	`FALSE`	Add detailed route geometry and indicators, such as `distance`, `routing` and `geometry`.

14.3 Inner Query

14.3.1 VRP JSON:

The JSON object contains the keys as described below:

Key	Type	Description
jobs	`ARRAY[JSON]`	Array of job objects describing the places to visit
shipments	`ARRAY[JSON]`	Array of shipment objects describing pickup and delivery tasks
vehicles	`ARRAY[JSON]`	Array of vehicle objects describing the available vehicles

Note:
- At least one of the key `jobs` or `shipments` shall be present in the JSON object.
- The vehicles are modelled as the resources, which pick and/or deliver the jobs and shipments.
- The jobs are the single-location pickup and/or delivery tasks.
- The shipments are the pickup-and-delivery tasks that should happen within the same route.

Optional Key	Type	Description
matrix	`ARRAY[ARRAY[INTEGER]]`	Two-dimensional array describing a custom square matrix

1. job

A job object has the following properties:

Key	Type	Description
id	`BIGINT`	Non-negative unique identifier of the job
location	`ARRAY[DOUBLE PRECISION]`	Coordinates array, expected as `[longitude, latitude]` ● `array_length(location, 1) ≥ 2` ● Only `location[1]` and `location[2]` are considered, rest ignored
location_index	`SMALLINT`	Non-negative index of relevant row and column in custom matrix ● Ranges from `[0, SIZE[matrix]-1]`

If a custom matrix is provided:
- `location_index` is mandatory
- `location` is optional but can be set to retrieve coordinates in the response

If no custom matrix is provided:

- `location` is mandatory
- `location_index` is irrelevant

Optional Key	Type	Default	Description
description	`TEXT`		Description of the job
service	`INTEGER`	0	Job service duration, in seconds
delivery	`ARRAY[BIGINT]`		Array of non-negative integers describing multidimensional quantities for delivery such as number of items, weight, volume etc. ● All jobs must have the same value of `array_length(delivery, 1)`
pickup	`ARRAY[BIGINT]`		Array of non-negative integers describing multidimensional quantities for pickup such as number of items, weight, volume etc. ● All jobs must have the same value of `array_length(pickup, 1)`
skills	`ARRAY[INTEGER]`		Array of non-negative integers defining mandatory skills ● A job can only be served by a vehicle that has all its required skills, i.e. `job j` is eligible to `vehicle v` iff `j.skills` is included in `v.skills`.
priority	`INTEGER`	0	Priority level of the job ● Ranges from `[0, 100]` ● Useful in situations where not all tasks can be performed, to gain some control on which tasks are unassigned.
time_windows	`ARRAY[ARRAY[INTEGER]]`		Array of `time_window` objects describing valid slots for job service start. Every `time_window` object satisfies this: ● `array_length(time_window, 1) ≥ 2`

		● Only `time_window[1]` and `time_window[2]` are considered, rest ignored ● `time_window[1] ≤ time_window[2]` Internally, it gets sorted in increasing order of `time_window` start time, then end time. All timings are in seconds.

2. shipment

A shipment object has the following properties:

Key	Type	Description
pickup	`JSON`	A [`shipment_step`] object describing pickup
delivery	`JSON`	A [`shipment_step`] object describing delivery

Optional Key	Type	Default	Description
amount	`ARRAY[BIGINT]`		Array of non-negative integers describing multidimensional quantities such as number of items, weight, volume etc. ● All shipments must have the same value of `array_length(amount, 1)`
skills	`ARRAY[INTEGER]`		Array of non-negative integers defining mandatory skills
priority	`INTEGER`	0	Priority level of the shipment ● Ranges from `[0, 100]`

shipment_step

A shipment_step object has the following properties:

Key	Type	Description
id	`BIGINT`	Non-negative unique identifier of the shipment (unique for pickup, and unique for delivery)
location	`ARRAY[DOUBLE PRECISION]`	Coordinates array, expected as `[longitude, latitude]` ● `array_length(location, 1) ≥ 2` ● Only `location[1]` and `location[2]` are considered, rest ignored
location_index	`SMALLINT`	Non-negative index of relevant row and column in custom matrix ● Ranges from `[0, SIZE[matrix]-1]` ● Used exclusively when the **location** key is not specified.

If a custom matrix is provided:

- `location_index` is mandatory
- `location` is optional but can be set to retrieve coordinates in the response

If no custom matrix is provided:

- `location` is mandatory
- `location_index` is irrelevant

Optional Key	Type	Default	Description
description	`TEXT`		Description of the shipment step
service	`INTEGER`	0	Task service duration, in seconds
time_windows	`ARRAY[ARRAY[INTEGER]]`		Array of `time_window` objects describing valid slots for task service start. Every `time_window` object satisfies this: ● `array_length(time_window, 1) ≥ 2` ● Only `time_window[1]` and `time_window[2]` are considered, rest ignored ● `time_window[1] ≤ time_window[2]` Internally, it gets sorted in increasing order of `time_window` start time, then end time. All timings are in seconds.

A `shipment_step` is similar to a job object, except for the shared keys already present in `shipment`.

3. vehicle

A vehicle object has the following properties:

Key	Type	Description
id	`BIGINT`	Non-negative identifier of the vehicle.
start	`ARRAY[DOUBLE PRECISION]`	Start coordinates array, expected as `[longitude, latitude]` ● `array_length(start, 1) ≥ 2` ● Only `start[1]` and `start[2]` are considered, rest ignored
start_index	`SMALLINT`	Non-negative index of relevant row and column in custom matrix ● Ranges from `[0, SIZE[matrix]-1]`
end	`ARRAY[DOUBLE PRECISION]`	End coordinates array, expected as `[longitude, latitude]` ● `array_length(end, 1) ≥ 2` ● Only `end[1]` and `end[2]` are considered, rest ignored
end_index	`SMALLINT`	Non-negative index of relevant row and column in custom matrix ● Ranges from `[0, SIZE[matrix]-1]`

Note:

- Key `start` and `end` are optional for a vehicle, as long as at least one of them is present.
- If `end` is omitted, the resulting route will stop at the last visited task, whose choice is determined by the optimization process
- If `start` is omitted, the resulting route will start at the first visited task, whose choice is determined by the optimization process
- To request a round trip, specify both `start` and `end` with the same coordinates

Depending on if a custom matrix is provided, the required fields follow this logic:

If a custom matrix is provided:

- `start_index` or `end_index` are mandatory
- `start` or `end` are optional but can be set to retrieve coordinates in the response

If no custom matrix is provided:

- `start` or `end` are mandatory
- `start_index` or `end_index` are irrelevant

Optional Key	Type	Default	Description
capacity	`ARRAY[BIGINT]`		Array of non-negative integers describing multidimensional quantities such as number of items, weight, volume etc. ● Models custom restrictions for several metrics. ● All vehicles must have the same value of `array_length(capacity, 1)`
profile	`TEXT`	`‘car’`	Routing profile of the vehicle. ● Every vehicle must have the same profile.
description	`TEXT`		Description of the vehicle
skills	`ARRAY[INTEGER]`		Array of non-negative integers defining skills
time_window	`ARRAY[INTEGER]`		Array of non-negative integers describing working hours, in seconds. ● `array_length(time_window, 1) ≥ 2` ● Only `time_window[1]` and `time_window[2]` are considered, rest ignored ● `time_window[1] ≤ time_window[2]`
breaks	`ARRAY[JSON]`		Array of `break` objects, described below. Internally, it gets sorted in increasing order of `time_windows`
steps	`ARRAY[INTEGER]`		Array of `vehicle_step` objects describing a custom route for this vehicle, described below. ● Makes sense only when the `plan` flag is `TRUE`.

These arrays are used to model capacity restrictions:

- `capacity` for vehicles.
- `delivery` and `pickup` for jobs.
- `amount` for shipments.

A vehicle is only allowed to serve a set of tasks if the resulting load at each route step is lower than the matching value in capacity for each metric. When using multiple components for amounts, it is recommended to put the most important/limiting metrics first.

- It is assumed that all delivery-related amounts for jobs are loaded at vehicle start, while all pickup-related amounts for jobs are brought back at vehicle end.

(i) break

A break object has the following properties:

Key	Type	Description
id	`BIGINT`	Non-negative identifier of the break • For the same vehicle, the `id` of the break is unique.
time_windows	`ARRAY[ARRAY[INTEGER]]`	Array of `time_window` objects describing valid slots for break start. Every `time_window` object satisfies this: • `array_length(time_window, 1) ≥ 2` • Only `time_window[1]` and `time_window[2]` are considered, rest ignored • `time_window[1] ≤ time_window[2]` Internally, it gets sorted in increasing order of `time_window` start time, then end time. All timings are in seconds.

Optional Key	Type	Default	Description
service	`INTEGER`	0	Break duration
description	`TEXT`		Description of the break

(ii) vehicle_step

A vehicle_step object has the following properties:

Key	Type	Description
type	`TEXT`	Kind of step the vehicle is at: • `start`: Starting location • `job`: Job location • `pickup`: Pickup location • `delivery`: Delivery location • `break`: Break • `end`: Ending location
id	`BIGINT`	Non-negative identifier of the task to be performed at this step, if type value is `job`, `pickup`, `delivery` or `break`

Optional Key	Type	Description
service_at	`INTEGER`	Hard constraint on service time, in seconds
service_after	`INTEGER`	Hard constraint on service time lower bound, in seconds
service_before	`INTEGER`	Hard constraint on service time upper bound, in seconds

4. matrix

The custom travel-time matrix can be used as an alternative to the travel-time matrix computed by the OSRM routing engine. If a custom matrix is provided:

- The `location`, `start` and `end` coordinates array become optional.
- The `location_index`, `start_index`, and `end_index` are used as row and column indicators during optimization.

The matrix must be a square matrix (same number of rows and columns) represented by a two-dimensional array.

14.4 Result Columns

RETURNS GeoJSON with keys `code, error, summary, unassigned, routes`

Key	Type	Description
code	`INTEGER`	Value of the status code ● `0`: No error raised ● `1`: Internal Error ● `2`: Input Error ● `3`: Routing Error
error	`TEXT`	Error message ● Key is present only if status code is not 0.
summary	`JSON`	A `summary` object summarizing solution indicators
unassigned	`ARRAY[JSON]`	Array of objects, describing unassigned tasks with their id and location (if provided)
routes	`ARRAY[JSON]`	Array of `route` objects, describing the route of the vehicle

14.4.1 summary

A summary object has the following properties:

Key	Type	Description
cost	`INTEGER`	Total cost for all routes
unassigned	`INTEGER`	Number of tasks that could not be served
`service`	`INTEGER`	Total service time for all routes, in seconds
duration	`INTEGER`	Total travel time for all routes, in seconds
waiting_time	`INTEGER`	Total waiting time for all routes, in seconds
priority	`INTEGER`	Total priority sum for all assigned tasks
violations	`ARRAY[JSON]`	Array of violation objects for all routes
delivery	`ARRAY[BIGINT]`	Total delivery for all routes
pickup	`ARRAY[BIGINT]`	Total pickup for all routes
distance	`INTEGER`	Total distance for all routes, in metres ● Only when `geometry` flag is `TRUE`
computing_times	`JSON`	Array with keys loading, solving, routing denoting the time in milliseconds required for the specified key.

14.4.2 route

A route object has the following properties:

Key	Type	Description
vehicle	`BIGINT`	Identifier of the vehicle assigned to this route
steps	`ARRAY[JSON]`	Array of step objects
`cost`	`INTEGER`	Cost for this route
service	`INTEGER`	Total service time for this route, in seconds

duration	`INTEGER`	Total travel time for this route, in seconds
waiting_time	`INTEGER`	Total waiting time for this route, in seconds
priority	`INTEGER`	Total priority sum for tasks in this route
violations	`ARRAY[JSON]`	Array of violation objects for this route
delivery	`BIGINT`	Total delivery for tasks in this route
pickup	`BIGINT`	Total pickup for tasks in this route
description	`TEXT`	Vehicle description ● Only if provided in the input
geometry	`TEXT`	Polyline encoded route geometry ● Only when `geometry` flag is `TRUE`
distance	`INTEGER`	Total route distance, in metres ● Only when `geometry` flag is `TRUE`

1. step

A step object has the following properties:

Key	**Type**	**Description**
type	`TEXT`	Kind of step the vehicle is at: ● `start`: Starting location ● `job`: Job location ● `pickup`: Pickup location ● `delivery`: Delivery location ● `break`: Break ● `end`: Ending location
arrival	`INTEGER`	Estimated time of arrival at this step, in seconds
duration	`INTEGER`	Cumulated travel time upon arrival at this step, in seconds
service	`INTEGER`	Service time at this step, in seconds

waiting_time	`INTEGER`	Waiting time upon arrival at this step, in seconds
violations	`ARRAY[JSON]`	Array of violation objects for this step ● Empty array if `plan` flag is `FALSE` ● Reported only when the `plan` flag is `TRUE` to choose ETA for custom routes using the steps keys for a vehicle
description	`TEXT`	Step description ● Only if provided in input
location	`ARRAY[DOUBLE PRECISION]`	Coordinates array for this step, as `[longitude, latitude]` ● Only if provided in the input
id	`BIGINT`	Identifier of the task performed at this step ● Only if `type` value is `job`, `pickup`, `delivery` or `break`
load	`BIGINT`	Vehicle load after step completion (with capacity constraints)
distance	`INTEGER`	Traveled distance upon arrival at this step, in metres ● Present only when `geometry` flag is `TRUE`

2. violation

A violation object has the following properties:

Key	Type	Description
cause	`TEXT`	Cause of the violation ● `delay`: if actual service is late on a time window end ● `lead_time`: if actual service is early on a time window start ● `load`: if vehicle load goes over its capacity ● `skills`: if vehicle does not hold all required skills for a task ● `precedence`: if shipment precedence constraint is not met ○ Pickup without matching delivery ○ Delivery before matching pickup ○ Delivery without matching pickup ● `missing_break`: if a vehicle break has been omitted in its custom route
duration	`INTEGER`	Duration of the violation, in seconds

- Earliness if cause is `lead_time`
- Lateness if cause is `delay`

14.5 Usage

The sample queries are taken from the documentation of VROOM[5], and are executed on the OpenStreetMap data of Ile-de-France[6], downloaded from Geofabrik[7]. An OSRM server[8] needs to be running for executing these queries.

Query 1: Without custom matrix

```
SELECT * FROM vrp_vroom(
'{"vehicles":[{"id":1,"start":[2.35044,48.71764],"end":[2.35044,48.71764],"capacity
":[4],"skills":[1,14],"time_window":[1600416000,1600430400]},{"id":2,"start":[2.350
44,48.71764],"end":[2.35044,48.71764],"capacity":[4],"skills":[2,14],"time_window":
[1600416000,1600430400],"breaks":[{"id":2,"service":300,"time_windows":[[1600423200
,1600425000]]}]}],"jobs":[{"id":1,"service":300,"delivery":[1],"location":[1.98935,
48.701],"skills":[1],"time_windows":[[1600419600,1600423200]]},{"id":2,"service":30
0,"pickup":[1],"location":[2.03655,48.61128],"skills":[1]},{"id":5,"service":300,"d
elivery":[1],"location":[2.28325,48.5958],"skills":[14]},{"id":6,"service":300,"del
ivery":[1],"location":[2.89357,48.90736],"skills":[14]}],"shipments":[{"amount":[1]
,"skills":[2],"pickup":{"id":4,"service":300,"location":[2.41808,49.22619]},"delive
ry":{"id":3,"service":300,"location":[2.39719,49.07611]}}]}'::json
);
```

Result:

```
'{"code":0,"summary":{"cost":0,"unassigned":0,"delivery":[4],"amount":[4],"pic
kup":[2],"service":2100,"duration":0,"waiting_time":0,"priority":0,"violations
":[],"computing_times":{"loading":14,"solving":3}},"unassigned":[],"routes":[{
"vehicle":1,"cost":0,"delivery":[3],"amount":[3],"pickup":[1],"service":1200,"
duration":0,"waiting_time":0,"priority":0,"steps":[{"type":"start","location":
[2.35044,48.71764],"service":0,"waiting_time":0,"load":[3],"arrival":160041870
0,"duration":0,"violations":[]},{"type":"job","location":[2.89357,48.90736],"i
d":6,"service":300,"waiting_time":0,"job":6,"load":[2],"arrival":1600418700,"d
uration":0,"violations":[]},{"type":"job","location":[2.28325,48.5958],"id":5,
"service":300,"waiting_time":0,"job":5,"load":[1],"arrival":1600419000,"durati
on":0,"violations":[]},{"type":"job","location":[2.03655,48.61128],"id":2,"ser
vice":300,"waiting_time":0,"job":2,"load":[2],"arrival":1600419300,"duration":
0,"violations":[]},{"type":"job","location":[1.98935,48.701],"id":1,"service":
300,"waiting_time":0,"job":1,"load":[1],"arrival":1600419600,"duration":0,"vio
lations":[]},{"type":"end","location":[2.35044,48.71764],"service":0,"waiting_
time":0,"load":[1],"arrival":1600419900,"duration":0,"violations":[]}],"violat
ions":[]},{"vehicle":2,"cost":0,"delivery":[1],"amount":[1],"pickup":[1],"serv
ice":900,"duration":0,"waiting_time":0,"priority":0,"steps":[{"type":"start","
location":[2.35044,48.71764],"service":0,"waiting_time":0,"load":[0],"arrival"
:1600422600,"duration":0,"violations":[]},{"type":"pickup","location":[2.41808
,49.22619],"id":4,"service":300,"waiting_time":0,"job":4,"load":[1],"arrival":
1600422600,"duration":0,"violations":[]},{"type":"delivery","location":[2.3971
9,49.07611],"id":3,"service":300,"waiting_time":0,"job":3,"load":[0],"arrival"
:1600422900,"duration":0,"violations":[]},{"type":"break","id":2,"service":300
,"waiting_time":0,"load":[0],"arrival":1600423200,"duration":0,"violations":[]
},{"type":"end","location":[2.35044,48.71764],"service":0,"waiting_time":0,"lo
ad":[0],"arrival":1600423500,"duration":0,"violations":[]}],"violations":[]}]}
'
```

Query 2: With custom matrix

```
SELECT * FROM vrp_vroom(
'{"vehicles":[{"id":0,"start_index":0,"end_index":3}],"jobs":[{"id":1414,"location_
index":1},{"id":1515,"location_index":2}],"matrix":[[0,2104,197,1299],[2103,0,2255,
3152],[197,2256,0,1102],[1299,3153,1102,0]]}'::json
);
```

Result:

```
'{"code":0,"summary":{"cost":5461,"unassigned":0,"service":0,"duration":5461,"
waiting_time":0,"priority":0,"violations":[],"computing_times":{"loading":0,"s
olving":3}},"unassigned":[],"routes":[{"vehicle":0,"cost":5461,"service":0,"du
ration":5461,"waiting_time":0,"priority":0,"steps":[{"type":"start","service":
0,"waiting_time":0,"arrival":0,"duration":0,"violations":[]},{"type":"job","id
":1414,"service":0,"waiting_time":0,"job":1414,"arrival":2104,"duration":2104,
"violations":[]},{"type":"job","id":1515,"service":0,"waiting_time":0,"job":15
15,"arrival":4359,"duration":4359,"violations":[]},{"type":"end","service":0,"
waiting_time":0,"arrival":5461,"duration":5461,"violations":[]}],"violations":
[]}]}'
```

14.6 Vehicle Routing Problems

14.6.1 Introduction

Vehicle Routing Problems (VRP) are a class of combinatorial optimization problems, which are used to design an optimal route for a fleet of vehicles, to service a set of customers, satisfying certain constraints. They model the distribution process from one or more depots to a number of geographically dispersed cities or customers.

The problem is defined by:

- a depot,
- a set of geographically dispersed customers with demands,
- a set of vehicles with capacity.

In addition, the customers are visited exactly once and the total customer demand of a route must not exceed the capacity of the vehicle. The VRP problems are classified as an NP-hard problem, meaning that the required solution time increases exponentially with size.

14.6.2 Mathematical Formulation

VRP can be represented in the form of standard terminologies of graph theory.

Given a graph G = (V, A) where

- V = {1, 2, …, n} is a set of vertices, representing the cities, having one or more depots located at the locations.
- A = set of arcs, connecting these cities.

Every arc (i, j) i ≠ j, is associated with a non-negative distance matrix C = (c_{ij}) denoting the travel time or the travel cost. Also, there are some m vehicles available at a depot.

The VRP consists of designing a set of least-cost vehicle routes, such that

- Each route starts and ends at the depot.
- Each city/customer is visited exactly once by exactly one vehicle.
- The total demand of each route does not exceed the vehicle capacity.
- The total routing cost is minimised.
- Some other side constraints are satisfied.

14.6.3 Side constraints associated with VRP

There are various constraints associated with a VRP problem:

- **Capacity restrictions**: A non-negative demand is associated with each city/customer, excluding the depot. The sum of demands or weights for any vehicle route shall not exceed the vehicle capacity. VRPs with these restrictions are referred to as CVRP.

- **Number of customers/cities visited**: In any route of the vehicle, the number of cities shall not exceed a given number 'n'.

- **Total time/distance restrictions:** The length of any route shall not exceed a distance 'd' or a time interval 't', which includes the travel time between the cities and the service time at each city. The VRPs with these restrictions are referred to as DVRP.

- **Time Windows**: Each city/customer must be visited within the time interval [ti, tj], with an allowed waiting time.

- **Precedence relation between pairs of cities**: City i shall be visited before city j.

14.6.4 Variants of the VRP

A lot of variants of the VRP have been proposed in the literature. For the scope of this proposal, I'm describing those variants which are supported by VROOM.

- **TSP (travelling salesman problem)**: When there is only one vehicle, it starts from a single depot, visits all the customers once, and then returns back to the same depot.

- **CVRP (capacitated VRP)**: The VRP where each vehicle has some capacity, in terms of total weight or total volume of the goods, or both. Each route of the vehicle must satisfy the capacity constraints. The capacity of each vehicle can be considered in 2D or 3D.

- **VRPTW (VRP with time windows)**: Each customer is associated with a time interval and can only be served within this interval. A set of time windows for each customer can also be considered, known as VRP with multiple time windows. These time windows can be flexible, associated with some extra costs, known as VRP with Soft time windows.
- **MDHVRPTW (multi-depot heterogeneous vehicle VRPTW)**: It is a combination of Heterogeneous fleet VRP (HVRP) and Multiple Depots VRP (MDVRP), where the company uses different kinds of vehicles, each with different capacity, and the company has several depots from which it can serve its customers. Therefore, the routes can have different starting or ending points.
- **PDPTW (pickup-and-delivery problem with TW)**: It is a combination of Pickup-and-delivery VRP (PDVRP) and the VRPTW. Each customer is associated with two quantities - representing pickup which needs to be returned to the depot and the demand which needs to be delivered to the customer from the depot. The total pickup and delivery quantity of a vehicle shall not exceed its capacity at any point of the route. There are several variants to this too, such as the pickup demand is delivered to another customer instead of the depot, the pickup and delivery of items are done to the same customer in one visit, and many more.

VROOM can also solve any mix of the above problem types.

14.7 VRP in VROOM

14.7.1 Standard Terminologies of VROOM

- **Vehicles**: the resources that pick and/or deliver the jobs and shipments.
- **Jobs**: the single-location pickup and/or delivery tasks.
- **Shipments**: the pickup-and-delivery tasks that should happen within the same route.
- **Tasks**: Either jobs or shipments are referred to as tasks.
- **Skills**: Every task and vehicle may have some set of skills. A task can be served by only that vehicle which has all the skills of the task.
- **Priority**: Tasks may have some priority assigned, which is useful when all tasks cannot be performed due to constraints, so the tasks with low priority are left unassigned.
- **Amount (for shipment), Pickup and delivery (for job)**: They denote the multidimensional quantities such as number of items, weights, volume, etc.
- **Capacity (for vehicle)**: Every vehicle may have some capacity, denoting the multidimensional quantities. A vehicle can serve only those sets of tasks such that the total sum of the quantity does not exceed the vehicle capacity, at any point of the route.
- **Time Window**: An interval of time during which some activity can be performed - working hours of the vehicle, break of the vehicle, or service start time for a task.
- **Break**: Array of time windows, denoting valid slots for the break start of a vehicle.

- **Service time**: The additional time to be spent by a vehicle while serving a task.
- **Travel time**: The total time the vehicle travels during its route.
- **Waiting time**: The total time the vehicle is idle, i.e. it is neither traveling nor servicing any task. It is generally the time spent by a vehicle waiting for a task service to open.

14.7.2 Comparison of the terminologies of VROOM[4] and vrpRouting[9]

Terminology	vrpRouting	VROOM
Vehicles	`vehicles`	`vehicles`
Shipments	`orders`	`shipments`
Jobs	-	jobs
Amount	denoted by `demand` of an `order` - single numerical value	denoted by `amount` (for `shipment`), `pickup` and `delivery` (for `job`) - array of quantities
Capacity	denoted by `capacity`, single numerical value	denoted by `capacity` - array of quantities
Time Windows	Single continuous slot	Array of different slots
Priority	-	`priority`
Skill	-	`skill`
Stop of a vehicle	`stop`: starting, pickup, ending or delivery location.	`step`: start, job, pickup, delivery, break or end.

14.7.3 Time Windows

In VROOM, time windows are used to describe the following events:

- **Working hours of a vehicle**: A single time window, during which the vehicle can serve the tasks.
- **Break of a vehicle**: An array of time windows, denoting the valid slots for a break start.
- **Task service start**: An array of time windows, denoting the valid slots when a task can be serviced.

The users can decide how to represent the time windows. The values can be interpreted as:

- **Absolute values**: representing the real timestamps. All the times reported in the output with the arrival key are then interpreted as real timestamps.
- **Relative values**: The time can be represented relative to some given time.
  E.g.: A 2 hour time window (2 hours = 2 * 60 * 60 minutes = 7200 minutes) can be represented as [0, 7200], if the initial time is taken as the reference.

If the time window is not specified, this means there are no time constraints on the problem.

- A vehicle with no `time_window` key will be able to serve any tasks anytime.
- A task with no `time_window` key might be included at any time in the route, satisfying other constraints.

Unlike VROOM, in vrpRouting, only single time window is allowed.

Time Window with two tasks

Let i and j be two tasks, and `i -> j` denotes that task j can be done after task i.

Let us use the following terminologies:

- Oi: Opening time of the time window for task i.
- Oj: Opening time of the time window for task j.
- Ci: Closing time of the time window for task i.
- Cj: Closing time of the time window for task j.
- Ai: Time of arrival of vehicle at node i.
- Aj: Time of arrival of vehicle at node j.
- AjOi: Time of arrival of vehicle at node j, if it arrived at the opening time of node i.
- AjCi: Time of arrival of vehicle at node j, if it arrived at the closing time of node i.
- Si: Service time at i
- Sj: Service time at j
- Tij: Travel time from i to j
- Wi: Waiting time at i
- Wj: Waiting time at j

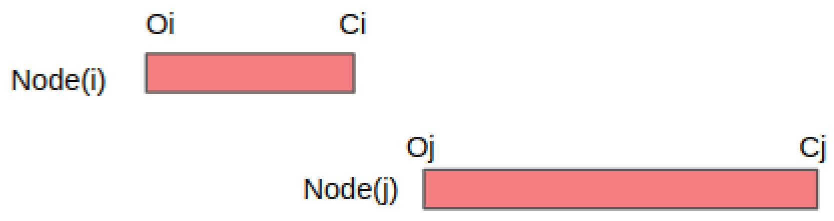
If i and j denote a shipment of a pickup-and-delivery pair, it must follow these two constraints:

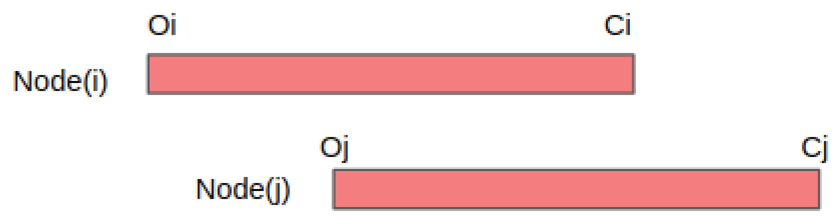
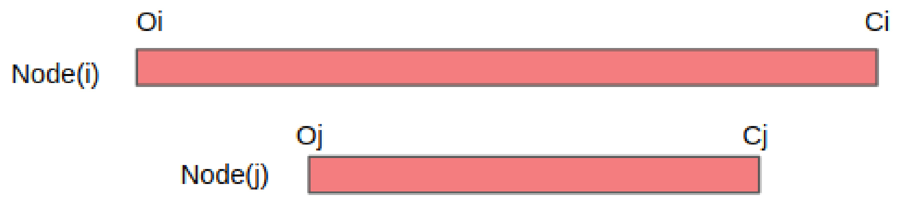
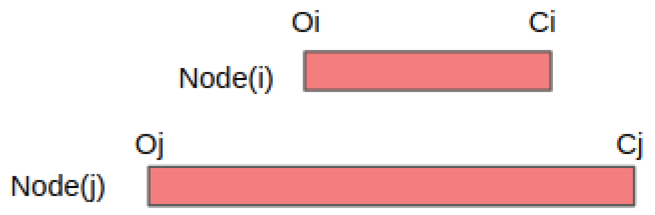
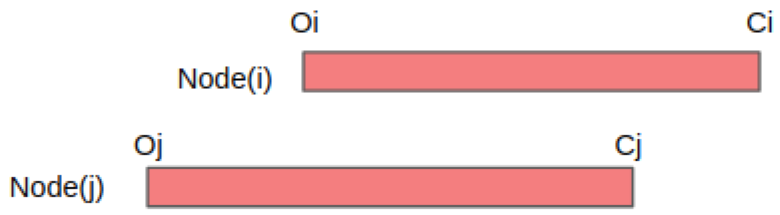
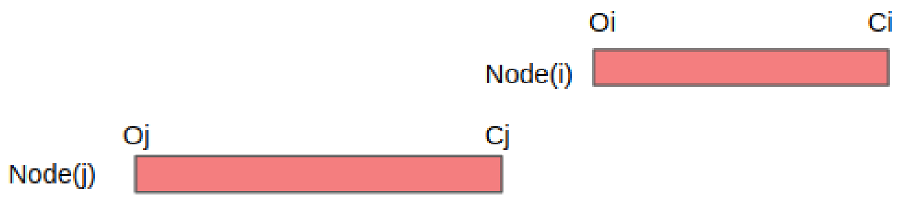
- `i -> j`, i.e. the delivery task can be done after the pickup.
- `! (j -> i)`, i.e. the delivery task cannot be done before the pickup.

Case I: Invalid time windows on the shipments

For any time window to be valid, the opening time must not exceed the closing time, i.e. for a shipment pair: Oi ≤ Oj and Ci ≤ Cj

The following figures denote the valid time windows, opening and closing time for tasks i and j. The horizontal axis denotes the time, and the vertical axis denotes the order in which a vehicle visits the nodes, starting from the top to the bottom node sequentially.

1. Oi ≤ Ci && Ci ≤ Oj && Oj ≤ Cj	

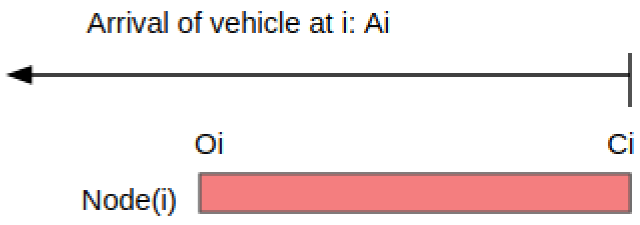
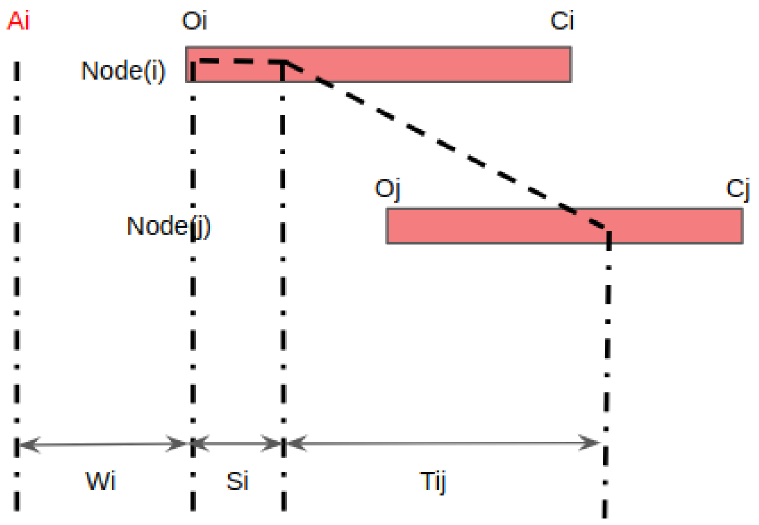
2. Oi ≤ Oj && Oj ≤ Ci && Ci ≤ Cj	
3. Oi ≤ Oj && Oj ≤ Cj && Cj ≤ Ci	
4. Oj ≤ Oi && Oi ≤ Ci && Ci ≤ Cj	
5. Oj ≤ Oi && Oi ≤ Cj && Cj ≤ Ci	
6. Oi ≤ Cj && Cj ≤ Oi && Oi ≤ Ci	

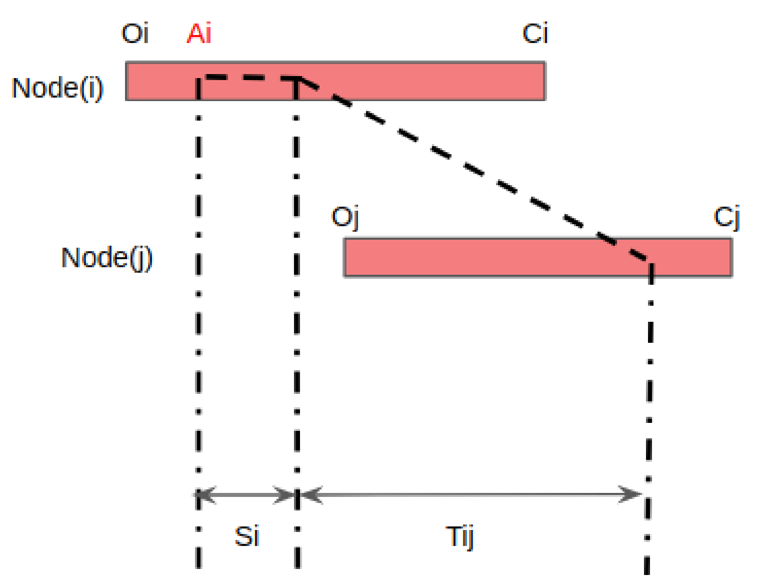
For a pickup-and-delivery pair i and j, the last figure denotes that the shipment cannot be served by a vehicle, under any circumstances.

Case II: Valid time windows on the shipment, but can't be served

Even if a shipment has valid time windows, it may not be served. We use these terminologies,in addition to the terminologies described above:

The following two conditions must be fulfilled for a shipment pair i-j to be served.

1. Ai ≤ Ci, i.e. the vehicle must arrive before the closing time at node i	
2. max(Ai, Oi) + Si + Tij ≤ Cj, i.e. the vehicle must arrive before the closing time at node j. Arrival time at j = Time when service at i starts + Service time + Travel time from i to j which must be less than the closing time of node j.	**Case I: Ai ≤ Oi** The node has to wait for a time Wi until the service at i opens (Oi). Therefore, Aj = Oi + Si + Tij  Ai Oi Ci Node(i) Oj Cj Node(j) Wi Si Tij **Case II: Ai > Oi** The service can start directly when the vehicle arrives at i. Aj = Ai + Si + Tij

	 Oi Ai Ci Node(i) Oj Cj Node(j) Si Tij

Time Window Compatibility (TWC) for two nodes[10]

The introduction of the TWC concept assists in identifying and eliminating the obvious infeasible nodes. This results in a more effective and robust route construction heuristic.

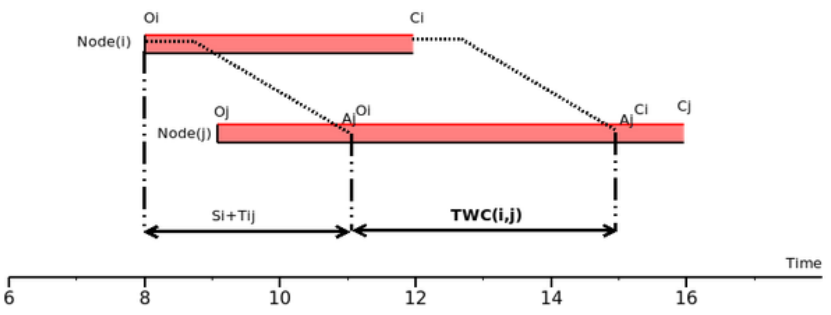
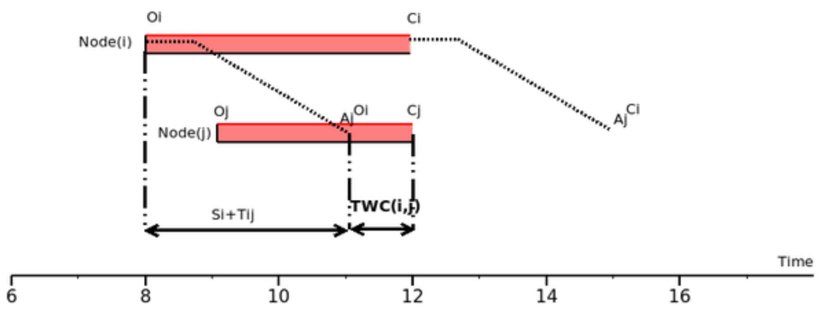
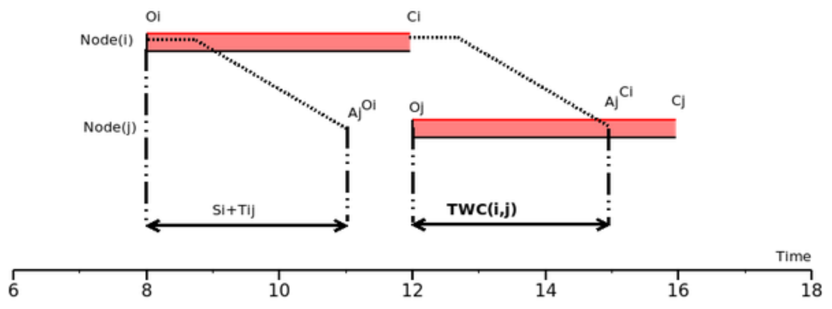
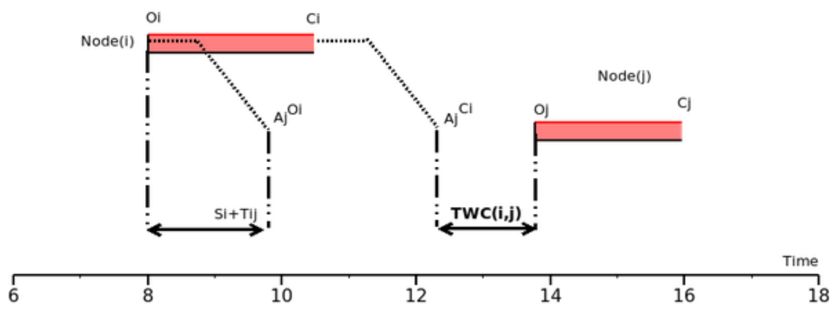
The purpose of TWC is to determine the time overlap of all node combinations i and j, where i, j ∈ {0, 1, 2, ..., n}. During the route construction phase, TWC may be tested, and nodes that are obviously infeasible may be eliminated from the set of considered nodes.

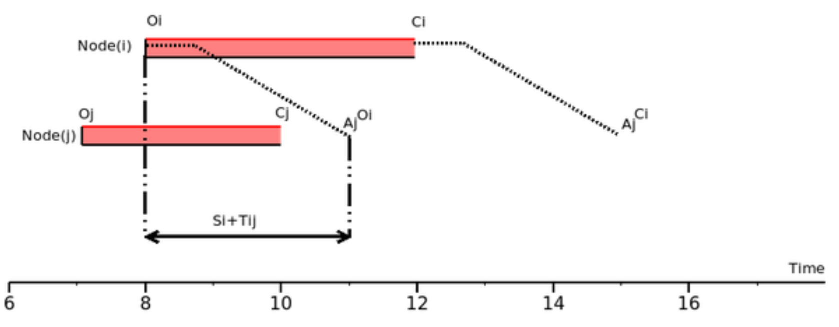
TWCij denotes the TWC when node i is directly followed by node j. In generalized form, the expression for TWCij is given as:

TWC ij = min(AjCi, Cj) - max(AjOi, Oj), if Cj - AjCi > 0
−∞, otherwise

The higher the value of TWC ij, the better the compatibility of the two time windows considered. Therefore an incompatible time window is defined to have a compatibility of negative infinity.

Based on the level and direction of overlap between the time windows of two consecutive nodes, the following five scenarios exist:

1. Fully compatible I +⟶ J AjOi > Oj && AjCi < Cj	 Oi Ci Node(i) Oj AjOi AjCi Cj Node(j) Si+Tij TWC(i,j) Time 6 8 10 12 14 16 $O_j < A_j^{O_i} < A_j^{C_i} \le C_j$
2. Partially compatible I +⟶ J AjOi > Oj && AjCi > Cj	 Oi Ci Node(i) Oj AjOi Cj AjCi Node(j) TWC(i,j) Si+Tij Time 6 8 10 12 14 16 $O_j < A_j^{O_i} < C_j < A_j^{C_i}$
3. Fully compatible with waiting time I +⟶ J AjOi < Oj && AjCi < Cj	 Oi Ci Node(i) AjOi Oj AjCi Cj Node(j) Si+Tij TWC(i,j) Time 6 8 10 12 14 16 18 $A_j^{O_i} < O_j < A_j^{C_i} \le C_j$
4. Negative compatible with waiting time I −⟶ J AjOi < Oj && AjCi < Oj	 Oi Ci Node(i) Node(j) AjOi AjCi Oj Cj Si+Tij TWC(i,j) Time 6 8 10 12 14 16 18 $A_j^{O_i} < A_j^{C_i} < O_j < C_j$

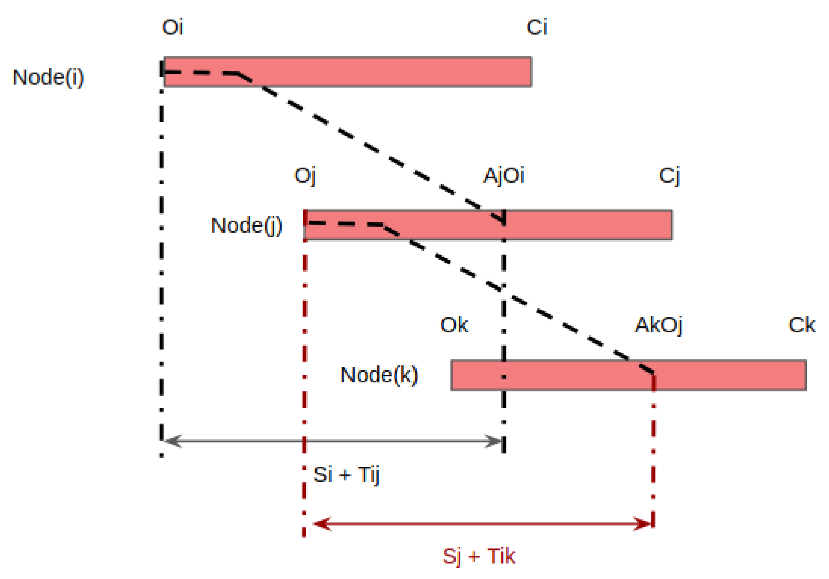
5. No compatibility I ⊖ J AjOi > Cj && AjCi > Cj	

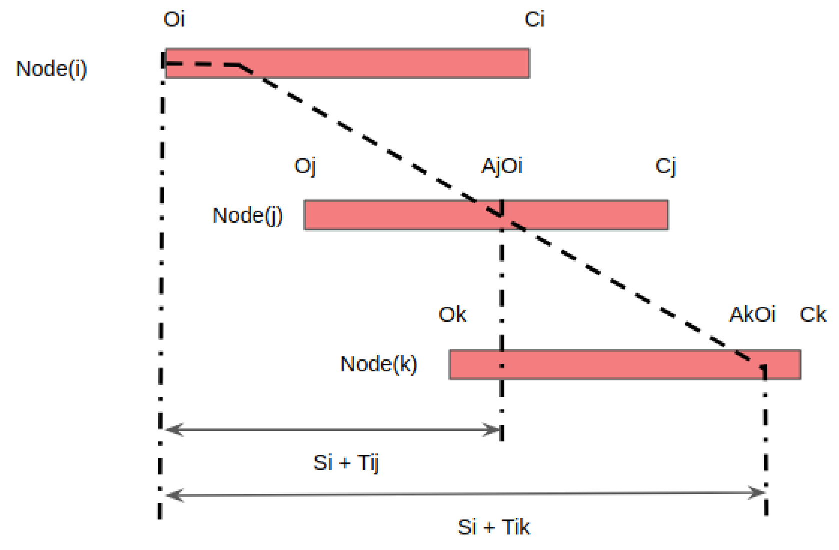
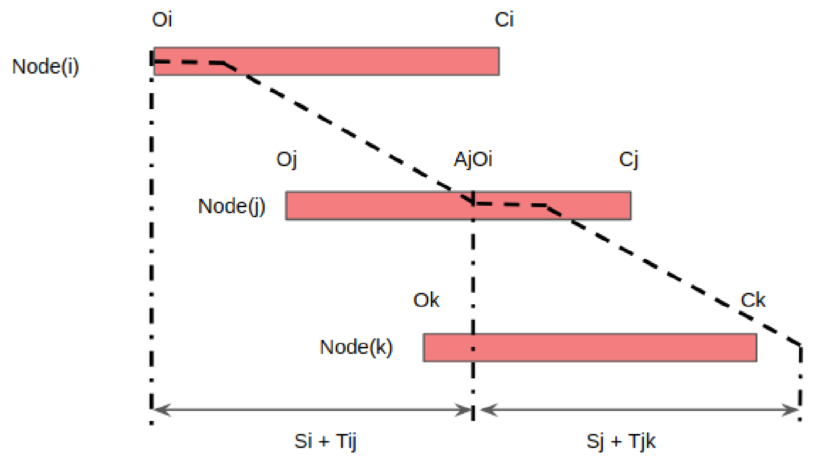
Time Window Compatibility for three consecutive nodes

Given three nodes i, j, k. such that (i -> j && j -> k && i -> k). This does not necessarily imply i -> j -> k. That is:

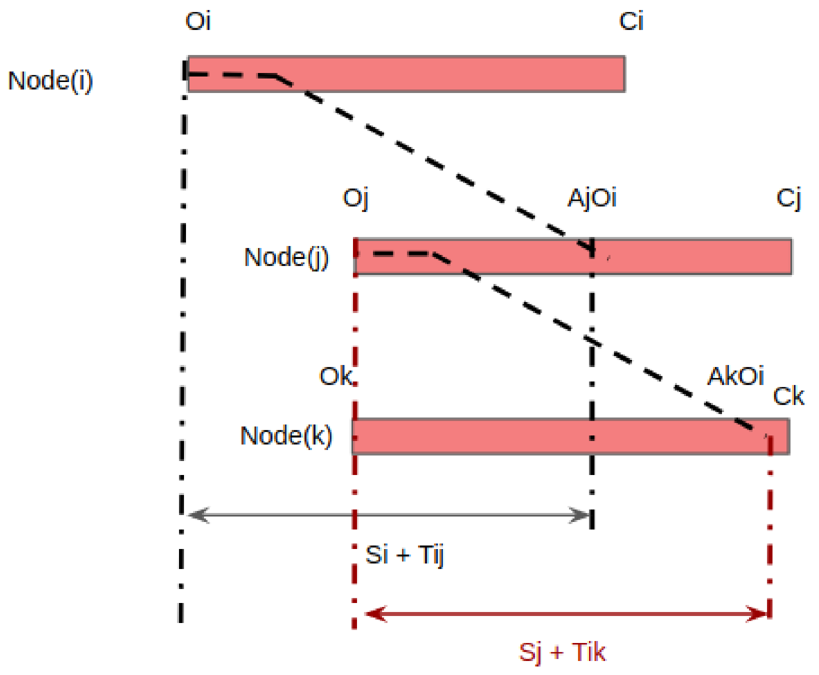
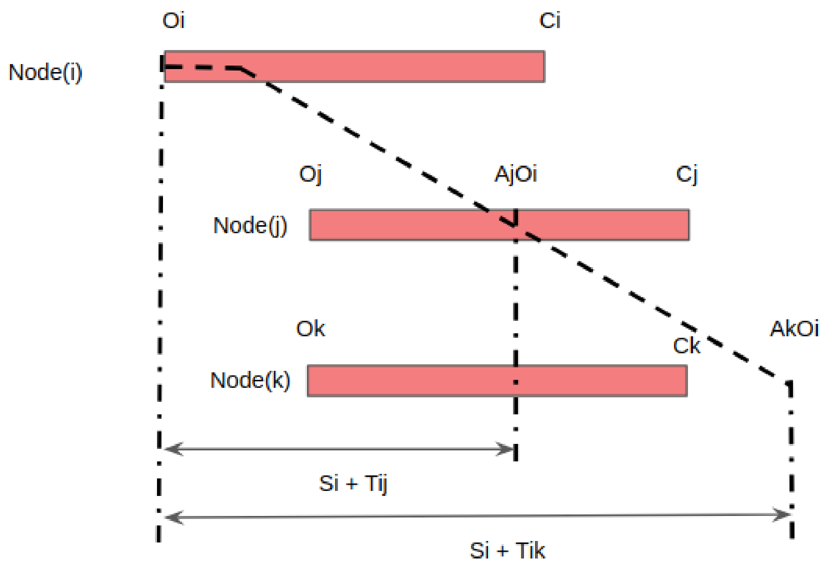
- if i and j are compatible
- if j and k are compatible
- if i and k are compatible
- The nodes i, j, k may not be simultaneously compatible.

This is evident from the following figures:

1. i -> j && j -> k	

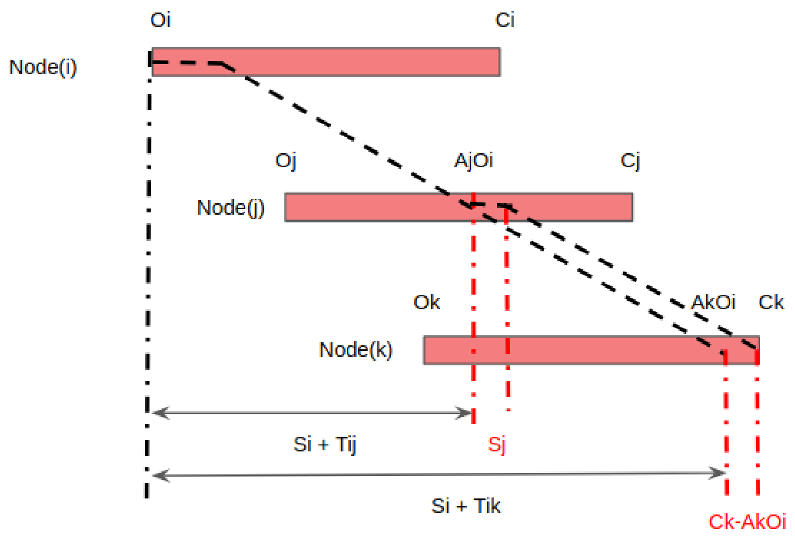
2. i -> k	Oi Ci Node(i) Oj AjOi Cj Node(j) Ok AkOi Ck Node(k) Si + Tij Si + Tik
3. i -> j /-> k	Oi Ci Node(i) Oj AjOi Cj Node(j) Ok Ck Node(k) Si + Tij Sj + Tjk

Similarly, if i -> j && j -> k, this does not necessarily imply i -> k, as evident from following figures

1. i -> j && j -> k	 Oi Ci Node(i) Oj AjOi Cj Node(j) Ok AkOi Ck Node(k) Si + Tij Sj + Tik
2. i /-> k	 Oi Ci Node(i) Oj AjOi Cj Node(j) Ok AkOi Ck Node(k) Si + Tij Si + Tik

Therefore, the condition for compatibility of three nodes i, j, k is

1. i -> j && j -> k
2. Ck - AkOi > Sj



14.7.4 VROOM Matrix

A matrix is represented as a two-dimensional array, with the same number of rows and columns in VROOM. E.g.: Consider the below matrix:

```
"matrix": [
    [0, 2, 3, 7],
    [1, 0, 1, 2],
    [2, 1, 0, 1],
    [3, 2, 1, 0]
]
```

The matrix is equivalently represented in pgrouting as a `SET OF (start_vid, end_vid, agg_cost)`. The equivalent matrix in pgrouting will be:

```
SELECT * FROM pgr_bdDijkstraCostMatrix(
    'SELECT id, source, target, cost, reverse_cost FROM edge_table',
    (SELECT array_agg(id) FROM edge_table_vertices_pgr WHERE id IN (1,2,3,7),
    false
);
 start_vid | end_vid | agg_cost
-----------+---------+----------
         1 |       2 |        1
         1 |       3 |        2
         1 |       7 |        3
         2 |       1 |        1
         2 |       3 |        1
         2 |       7 |        2
         3 |       1 |        2
         3 |       2 |        1
         3 |       7 |        1
         7 |        1 |         3
         7 |        2 |         2
         7 |        3 |         1
```

Unlike VROOM, in pgrouting, indexes can be given to the values in the matrix. Thus, every cell of VROOM's matrix corresponds to a row in pgrouting with three values. Since, for a cell, a single value is required in VROOM corresponding to 3 values in pgrouting, therefore the VROOM's matrix consumes less amount of memory.

14.8 Implementation Details

14.8.1 File Structure

(i) Implementation Files, with self-documented code

Directory	Files
`include/vroom/`	`vrp_vroom.hpp`
`include/drivers/vroom/`	`vroom_driver.h`

`sql/vroom/`	`CMakeLists.txt`
	`_vrp_vroom.sql`
	`vrp_vroom.sql`

`src/vroom/`	`CMakeLists.txt`
	`vroom.c`
	`vroom_driver.cpp`

(ii) Files for pgTAP tests

Directory	Files

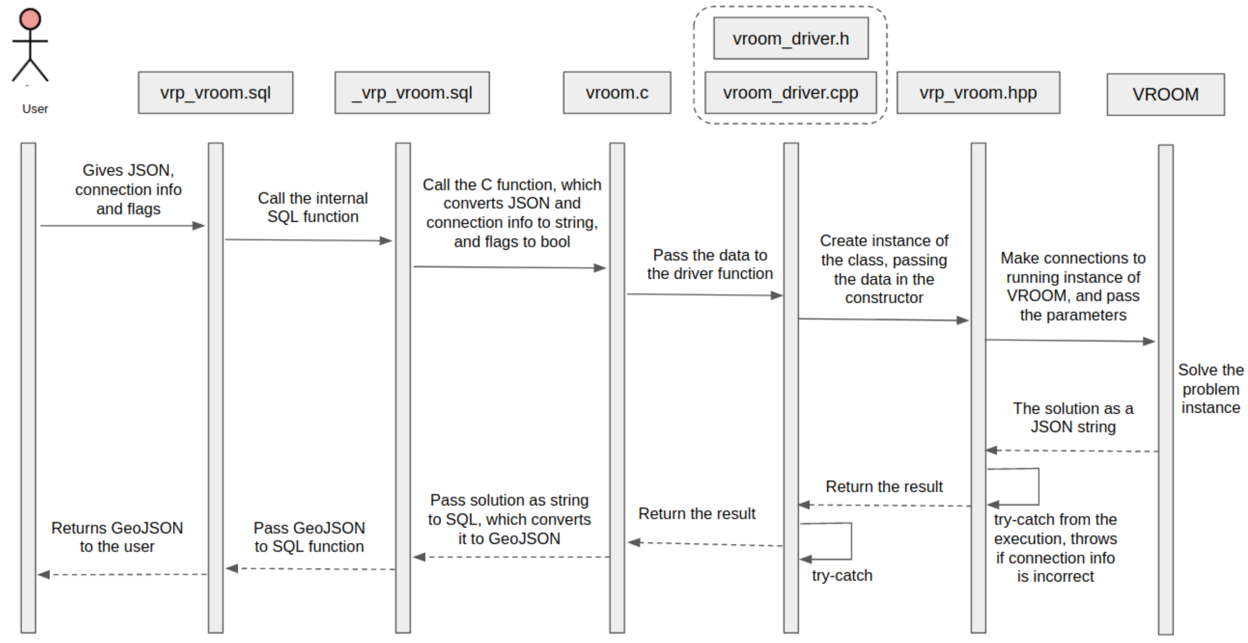
`pgtap/vroom/`	`vroom-innerQuery.sql`
	`vroom-types-check.sql`
	`vroom-connection-check.sql`

(iii) User's Documentation

Directory	Files
`doc/vroom/`	`CMakeLists.txt` `vrp_vroom.rst`

`docqueries/vroom/`	`CMakeLists.txt`
	`doc-vrp_vroom.result`
	`doc-vrp_vroom.test.sql`
	`test.conf`

14.8.2 Flow of Execution


User
vrp_vroom.sql
_vrp_vroom.sql
vroom.c
vroom_driver.h
vroom_driver.cpp
vrp_vroom.hpp
VROOM
Gives JSON, connection info and flags
Call the internal SQL function
Call the C function, which converts JSON and connection info to string, and flags to bool
Pass the data to the driver function
Create instance of the class, passing the data in the constructor
Make connections to running instance of VROOM, and pass the parameters
Solve the problem instance
The solution as a JSON string
try-catch from the execution, throws if connection info is incorrect
Return the result
try-catch
Return the result
Pass solution as string to SQL, which converts it to GeoJSON
Pass GeoJSON to SQL function
Returns GeoJSON to the user

15. Future Work

If time allows:

- Additional pgTAP tests can be added, such as the server no crash tests and unit tests.
- More specific functions can be made using VROOM, where the user can give the input in the form of SQL query (instead of JSON structure), and get the result as a set of rows (instead of the GeoJSON).
- An example of such function can be the school bus problem, in which, in the morning the jobs are pickups, and the delivery is at school, and in the afternoon the jobs are deliveries, and the pickup is at school. A function such as `vrp_schoolBusProblem(sql queries do not need shipment)`, can be created for it.

16. References

[1].Vehicle Routing Open-source Optimization Machine: VROOM-Project/vroom
[2]. vrpRouting - Vehicle Routing problems on PostgreSQL: pgRouting/vrprouting
[3]. Vehicle Routing Problems - Wikipedia
[4]. VROOM API Documentation
[5]. VROOM Documentation - Examples
[6]. OpenStreetMap data for Ile-de-France: ile-de-france
[7]. OpenStreetMap data extracts - Geofabrik
[8]. Open Source Routing Machine (OSRM) - Backend: Project-OSRM/osrm-backend
[9]. Vehicle Routing Functions - Category (Experimental) - vrpRouting documentation
[10]. A sequential insertion heuristic for the initial solution to a constrained vehicle routing problem