Getting Started with NVIDIA cuOpt

User Guide

DU-11281-001 | January 2023

Document History

DU-11281-001

Version	Date	Authors	Description of Change
01	2023-01-30	Anandh Anandh and Robert Sohigian	Initial release

Contents

cuOpt Basics

Overview

NVIDIA® cuOpt is a GPU-accelerated combinatorial optimization engine that enables real-time route optimization for problems such as the Traveling Salesperson Problem (TSP), Vehicle Routing Problem (VRP), and Pickup and Delivery (PDP) problems common to logistics and operations research. NVIDIA cuOpt supports many variations on these base problems, by adding constraints on vehicle capacities, delivery time windows, driver, and vehicle breaks during the working day, and so on.

TSP, VRP, and PDP

The VRP and PDP problems are derived from the TSP, which is one of the most studied problems in operations research. TSP asks the following question: "given a list of destinations and a matrix of distances between each pair of destinations, what is the shortest possible route that visits each destination exactly one time and returns to the original location?" For example, the TSP has several applications in planning and logistics, where a good solution can save significant travel time and fuel costs in transportation and delivery of goods. VRP and PDP are derived from TSP.

VRP generalizes the TSP to solve for the optimal set of routes for a fleet of vehicles in order to deliver to a given set of customers. PDP adds the possibility of two different types of services, namely pickup or delivery, whereas in VRP all customers require the same service be performed at a customer location.

What is NP-Hard?

In mathematical terms, the TSP, VRP, and PDP belong to the class of problems called ‘NP-hard’, meaning that the required time to find an optimal solution increases *at least* exponentially with the size of the problem (for example, number of deliveries to make). The number of possible states in the search space for VRP is of the order of n!, where n is the number of nodes (locations the vehicle must reach) in the network. Given the large search space, brute force approaches are practically intractable for large problem sizes (more than a few dozen locations) even on a modern supercomputer.

For instance, a ten-node problem has about 3628800 (3.6*10^6) possible states, but if we double the problem size, the solution space becomes 2432902008176640000 (2.4 * 10^18), which means that the solution space grew by a factor of 670442572800; it is about 6.7 trillion times larger.

The Necessity for Heuristics

Given the time and computational resources required for brute-force enumeration, obtaining the exact optimal solution is not realistic at all. However, there are well-studied heuristics that yields near-optimal solutions for very large networks within a reasonable time, and NVIDIA cuOpt focuses on using these heuristics.

How cuOpt Solves the Problem

cuOpt first generates a feasible initial solution (initial phase) and then iteratively improves the solution quality (improvement phase). The termination criteria is reached when either the solution quality is improving slower than a threshold tolerance (which is adaptively set internally by cuOpt heuristics) or if an execution time-limit has elapsed.

GPUs Unleash Massive Parallel Computing Capabilities

GPUs, with their ability to harness thousands of parallel cores, are an ideal computing platform for accelerating massive parallelizable problems where thousands or millions of separate tasks to be computed in parallel. This enables orders-of-magnitude speedups when running heuristics for this class of problems.

Climbers

cuOpt has innovatively adapted meta-heuristics (tabu search, guided local search, ant colony) to exploit the power of GPU architecture using parallel climbers. Climbers enable parallel exploration of the solution space, and each climber is essentially a separate thread of execution working on a distinct subset of the solution space. Each climber is only a weak solver, executing hill climber heuristics that cannot escape a local optima, but combining local climbers with the meta-heuristics allows a methodical exploration of the solution space and in practice a rapid convergence to a neighborhood of the global optima.

There is no guarantee that the global optima is reached, only that we find a ‘good’ solution in a reasonable amount of time.

Approaches to using cuOpt

1. cuOpt provides developer APIs in python that are intuitive and easy to adopt. It exposes a composable solver for all the implemented variants of the VRP problem and available heuristics.
2. A containerized microserver has been built upon the core functionality of cuOpt that can be easily deployed. Clients can set and update data and get results from the microserver using REST APIs. The microserver leverages `HTTP POST` requests running on `port 5000` to accept input data.
   a. Cloud scripts are also provided to spin up an instance of cuOpt microserver on a CSP (Cloud Service Provider). Deployment on AWS, Azure, and GCP are supported.

Setup

Refer to the prerequisites and setup instructions at this location:

https://docs.nvidia.com/cuopt/setup.html

cuOpt Memory Usage

Before diving into this, it is key to understand the two types of memories—global memory and shared memory. `cudaMalloc` always allocates global memory that resides on the GPU. The contents of global memory are visible to all the threads running in each kernel that is any thread can read and write to any location of the global memory. In contrast, shared memory is memory shared between the threads within a block and is not visible to all threads. For instance, the shared memory of the A100 GPUs capacity per SM is 164 KB, with 108 SMs on the chip. Each SM has an isolated shared memory and can only communicate by copying to global memory. Global memory is limited by the total memory available to the GPU, for instance, an A100 GPU (40 GB) offers 40 GB of device memory. Shared memory is like a local cache shared among the threads of a block, magnitudes faster to access than global memory and limited in capacity.

While in general cuOpt memory usage is problem size dependent, it is equally important to note that cuOpt memory usage is very sensitive to the constraints specified. The global memory usage is determined by the size of the input distance and cost matrices while the longest route in terms of number of nodes on the route determines the peak shared memory usage. As an approximate estimate, a 10,000 locations CVRPTW test case with challenging constraints can execute on a singleA100 GPU (40 GB) without any out-of-memory issue. However, for even larger problem sizes, cuOpt in the latter half of 2023 will offer multi-GPU support by partitioning the inputs seamlessly across GPUs.

cuOpt Troubleshooting

1. Docker sanity check.

If docker is installed and configured correctly, the following will run `nvidia-smi` in a container and display information about the GPU. If this command fails, check the items below.

```
$ docker run --rm --gpus all nvcr.io/nvidia/cuda:12.0.0-base-ubuntu20.04 nvidia-smi
```

2. Docker not installed or the wrong version.

This page shows the supported Linux distributions and container runtime versions.

Check the Docker version.

```
$ docker –version
```

If docker is not installed, or the docker version is not supported, follow this guide to install a supported version. We recommend at least Docker 19.03 because it includes the `-gpus` flag

3. Docker does not start.
   Current user does not have permission to run containers.

   Docker may return an error like this:

   ```
   docker: Got permission denied while trying to connect to the Docker daemon socket…
   ```

   This means that the current user is not part of the `docker` group. To enable the current user to run containers:

   ```
   $ sudo usermod -aG docker $USER
   ```

   Logout and then login again for the changes to take effect.

   Docker service is not running.

   Check the status of the docker service.

   ```
   $ systemctl status docker
   ```

   If the status shows that docker is not running, try restarting it.

   ```
   $ sudo systemctl restart docker
   ```

   If docker still does not start, check the status report for error messages.

NVIDIA container toolkit is not installed.

If the NVIDIA container toolkit is not installed, docker may return an error like to this:

```
docker: Error response from daemon: could not select device driver ""
with capabilities: [[gpu]]
```

Follow this guide to set up the NVIDIA Container Toolkit.

NVIDIA GPU driver is not installed.

If the NVIDIA driver is not installed, docker may return an error similar to this:

```
docker: Error response from daemon: failed to create shim: OCI runtime
create failed: runc create failed: unable to start container process:
error during container init: error running hook #0: error running
hook: exit status 1, stdout: , stderr: Auto-detected mode as 'legacy'
nvidia-container-cli: initialization error: load library failed:
libnvidia-ml.so.1: cannot open shared object file: no such file or
directory: unknown.
```

There are various ways to install the NVIDIA GPU driver. One simple method is to install the CUDA Toolkit. For other methods, search the NVIDIA documentation.

4. cuOpt service not starting.
   Check the logs for the container (see cuOpt service monitoring below).

   Is port 5000 already in use?

   If port 5000 is unavailable, the logs will contain an error like this

   ```
   ERROR: [Errno 98] error while attempting to bind on address
   ('0.0.0.0', 5000): address already in use”
   ```

   Try to locate the process that is using port 5000 and stop it if possible. A tool like `netstat` run as the root user can help identify ports mapped to processes, and `docker ps` will show running containers.

   Alternatively, use port mapping to launch cuOpt on a different port such as 5001 (note the omission of `–network=host` flag):

   ```
   $ docker run -d --rm --gpus all -p 5001:5000
   nvcr.io/nvidia/cuopt/cuopt:22.12
   ```

5. cuOpt service not responding
   cuOpt microservice health check on the cuOpt host

   Perform a health check locally on the host running cuOpt:

   ```
   $ curl -s -o /dev/null -w '%{http_code}\n' localhost:5000/cuopt/health
   200
   ```

   If this command returns 200, cuOpt is running and listening on the specified port.

   If this command returns something other than 200, check the following:

   - Check that a cuOpt container is running with `docker -ps`
   - Examine the cuOpt container log for errors

- Did you include the `–network=host` or a `-p` port-mapping flag to docker when you launched cuOpt? If you used port mapping, did you perform the health check using the correct port?
- Restart cuOpt and see if that corrects the problem

cuOpt microservice health check from a remote host

If you are trying to reach cuOpt from a remote host, run the health check from the remote host and specify the IP address of the cuOpt host, for example:

```
$ curl -s -o /dev/null -w '%{http_code}\n'
34.23.145.121::5000/cuopt/health
200
```

If this command does not return 200, but a health check locally on the cuOpt host does return 200, the problem is a network configuration or firewall issue. The host is not reachable, or the cuOpt port is not open to incoming traffic.

6. cuOpt service monitoring

   checking the cuOpt container log

   Look for the cuOpt container id, for example:

   ```
   $ docker ps --format 'table {{.ID}} {{.Image}}' | grep cuopt
   22a1726778ee nvcr.io/nvidia/cuopt/cuopt:22.12
   ```

   Print the logs using the container id.

   ```
   $ docker logs 22a1726778ee
   INFO:    Started server process [1045]
   INFO:    Waiting for application startup.
   INFO:    Application startup complete.
   INFO:    Uvicorn running on http://0.0.0.0:5000 (Press CTRL+C to quit)
   INFO:    127.0.0.1:36870 - "GET /cuopt/health HTTP/1.1" 200 OK
   INFO:    12.22.141.131:16016 - "GET /cuopt/health HTTP/1.1" 200 OK
   ```

   cuOpt microservice health check

   Perform a health check on the host running cuOpt.

   ```
   $ curl -s -o /dev/null -w '%{http_code}\n' localhost:5000/cuopt/health
   200
   ```

7. How to check if my configuration is valid for a cuOpt variant (constraints, inputs) when using the cuOpt service?

   The cuOpt service will validate the data passed in a set or update operation for the following general data categories:

   - Cost matrix
   - Waypoint graph
   - Fleet data
     - Includes time windows, breaks, capacities, and so on.
   - Task data
     - Includes time windows, service times, priorities, and so on.
   - Solver configuration

If the data passed is not valid, the HTTP status code of the response will be >= 400, and the response will contain a detailed error message. If the data passed is valid, the HTTP status code of the response will be 200. See the Server API documentation for more detail on the HTTP response.

Other configuration errors may be caught by the solver when `get_optimized_routes` is called.

8. Common misconfigurations reported by the cuOpt service, and how to fix them (missing data, incompatible constraints)

   The error messages listed below are up to date for version 22.12 of the cuOpt server.

   Errors reported when setting or updating the fleet data

   Errors reported when setting or updating the solver configuration

   ```
   No solver config to update. The set_solver_config endpoint must be
   used before update is available
   ```

   This means that `update_solver_config` was called without first calling `set_solver_config`.

   ```
   (SolverSettings time limit | Number of iterations | Number of
   climbers) must be greater than 0
   ```

   These values must be greater than zero or left unspecified.

   ```
   Solution scope must be selected from the following FEASIBLE, SOFT_TW
   ```

   The solution scope must be one of the enumerated values or left unspecified. The default is FEASIBLE.

   ```
   Initial Solution Strategy must be selected from the following
   INSERTION, SAVINGS
   ```

   The initial solution strategy must be set to INSERTION (the default) or left unspecified. Although SAVINGS is listed as a possible value, it is not supported.

   ```
   Solution Strategy must be selected from the following HILL_CLIMBING,
   TABU_SEARCH, TABU_GLS, NONE
   ```

   The solution strategy must be one of the enumerated values or left unspecified. The default is TABU_GLS. If NONE is selected, there will be no improvement phase.

   ```
   File path to save configuration should be valid and not empty
   ```

   This means that the value of `config_file` was an empty string.

   Errors reported when setting or updating a cost or travel time matrix

   ```
   Cost matrix must be a square matrix
   ```

   A cost matrix must contain the cost to travel from each location to each location (including traveling from a location to itself). So, a cost matrix for two locations should be a 2x2 matrix (for a total of four separate values).

   ```
   All rows in the cost matrix must be of the same length
   ```

   The length of each row should equal the number of locations. At least one row has missing or extra values.

`All values in cost_matrix must be >= 0`

At least one value in the cost matrix is negative. Negative costs are not allowed.

`If updating a matrix, a matrix must already be set. If adding a travel time matrix, a primary matrix must already be set`

This message can mean that `update_cost_matrix` or `update_travel_time_matrix` was called without first calling `set_cost_matrix` or `set_travel_time_matrix` respectively.

It can also mean that `set_travel_time_matrix` was called without first calling `set_cost_matrix`.

`When updating a cost matrix or setting a travel time matrix, the shape of the input must match the shape of the primary matrix`

This message can mean that `update_cost_matrix` or `update_travel_time_matrix` was called with a matrix having a different number of rows and columns than the matrix passed to `set_cost_matrix` or `set_travel_time_matrix` respectively.

It can also mean that `set_travel_time_matrix` was called with a matrix having a different number of rows and columns than the matrix passed to `set_cost_matrix`.

`Matrices for all vehicle types must be the same shape` (versions 22.12+)

Cost matrices were passed for multiple vehicle types, but they are not all the same shape.

cuOpt FAQ

1. **cuOpt resource estimates (how large a problem can I run with xyz constraints).**

For the standard CVRPTW problem with real-world constraints, cuOpt can easily solve for 10K locations.

2. **How to set up multi-GPU cuOpt on a single server**

Currently cuOpt does not support using multiple GPUs from a single cuOpt instance.

Research is ongoing into which use cases may benefit from multiple GPUs. For the time being, if multiple GPUs are available, scale out is possible by partitioning problems into sub-problems and running multiple cuOpt instances.

3. **How to set up high-availability service using Kubernetes + Helm Charts**

cuOpt does not provide any high-availability failover mechanisms at this time, or the ability to share state between multiple cuOpt services. A problem must be set up and solved on the same cuOpt instance. We understand the need for high availability in production environments and plan to address this in the future.

4. **How do I solve (some realistic problem) -> this is probably a decision tree that points to some variant of CVRP or PDP with constraints that match their needs but do not over-constraint the problem.**

Benchmarks and quality numbers for Homberger and L&L suites (for competitive comparisons) on various GPUs (V100, A100, H100 when available, maybe even some SOCs like NVIDIA Orin).

5. **Do I need a GPU to use cuOpt?**

GPU is required and cuOpt works on any NVIDIA Pascal GPUs or later.

6. **What CUDA versions are supported?**

The cuOpt pre-built containers are currently built using CUDA 11.5. It is also recommended to use CUDA 11.5 when building from source to keep in alignment with underlying dependencies like RAPIDS.

7. **Can I deploy cuOpt to my cloud environment?**

Yes, you can find script to deploy on major clouds in the resource repository.

8. **Is CuOpt compatible with the new NVIDIA Hopper DPX instructions?**

Support for NVIDIA Hopper DPX instructions in cuOpt will come out in later part of 2023.

9. **What is the memory consumption compared to the scale of data/model settings (for example, locations or number of climbers).**

The amount of memory used is dependent on the solution structure. It is proportional to: (N^2 + N*c) * n_climbers where N is the number of nodes and c is a small coefficient (<30). A GPU with 4 GB is sufficient for medium datasets, if they do not have a high number of constraints. A GPU of at least 16 GB is recommended for flexibility.

10. **Is there a way to change the default port of cuOpt server?**

The default port cannot be changed but, it can be overridden with port mapping if launching from docker, and if deploying in Kubernetes it can be mapped to a different port on the host with a NodePort service.

In the docker port-mapping case, it looks like this:

```
docker run --rm --gpus=all -it -p 8001:5000 nvcr.io/nvstaging/cuopt/cuopt:22.12
```

Leave off `--network=host` and supply a `-p` argument, should be able to run multiple cuOpt instances this way.

Installation error - when installing NVIDIA driver on Ubuntu 20.04:

`ERROR: Unable to load the kernel module nvidia.ko`. This happens most frequently when this kernel module was built against the wrong or improperly configured kernel sources, with a version of gcc that differs from the one used to build the target kernel, or if another driver, such as nouveau, is present and prevents the NVIDIA kernel module from obtaining ownership of the NVIDIA device(s), or no NVIDIA device installed in this system is supported by this NVIDIA Linux graphics driver release.

See the log entries `Kernel module load error` and `Kernel messages` at the end of the file `/var/log/nvidia-installer.log` for more information.

> `Kernel module load error: No such device`

> Kernel messages:

```
[ 1716.519323] [UFW BLOCK] IN=eno1 OUT=
MAC=18:66:da:20:86:2f:b8:69:f4:01:56:96:08:00 SRC=159.65.178.29 DST=89.216.51.58
LEN=40 TOS=0x00 PREC=0x00 TTL=242 ID=44827 PROTO=TCP SPT=61953 DPT=6156
WINDOW=1024 RES=0x00 SYN URGP=0
```

There is a resource on NGC that installs the necessary drivers and Docker on a clean Ubuntu 18.04/20.04/22.04 machine. It has been tested on a fresh OS install, and it may also work on an existing machine where other packages/drivers have been installed.

```
https://ngc.nvidia.com/resources/ea-reopt-member-zone:setup_ubuntu_for_cuopt
```

Also, note the new cuOpt-Resources repository on GitHub.

There are scripts there under “cloud-scripts” that will create a VM for you on AWS, Azure, or GCP running cuOpt on top of Cloud Native Core (single node Kubernetes), or the same stack can be installed locally on another system running Ubuntu 22.04.

https://github.com/NVIDIA/cuOpt-Resources

11. **Is cuOpt supported on NVIDIA H100/Hopper architecture?**

cuOpt will be supported on NVIDIA Hopper architecture by end of 2023. Updates will be posted to the forums.

Solver-Modeling Issues/Questions

12. Is there any lower bound as Mixed integer gives a solution and lower bound?

cuOpt does not provide any lower bound. cuOpt is based on heuristics and does not use exact methods so lower bound does not matter as long as the solution is robust, and within constraints.

13. How do we account for dynamic changing constraints while the solver is executing?

Dynamic reoptimization is used when there is a change in the conditions of the operation. A vehicle getting broken, a driver calling in sick, road block, traffic, a high priority order coming in. Currently, a solver instance is started by considering which packages each vehicle has and the last location(cannot be in between nodes) of the vehicle. The solver then can be run with any configuration. Except the ones that are currently in truck and waiting to be delivered.

14. How are order cancellations handled in cuOpt?

We can reduce the capacity of the vehicle accordingly, or the user can provide information wherein the canceled order can be dropped off at a location/depot. In such cases, set large time windows to derive an optimal solution.

15. Is there a C++ SDK to the cuOpt?

We support microservices and python APIs that just sit on top of our C++ code.

16. How many climbers do I need?

The total time depends on the user. The user has the flexibility of setting a higher time-limit for better results. If the user wants the first solution, then around 2-3 seconds for 2000-4000 climbers are enough. cuOpt solver does not interrupt the initial solution. So if the user specifies a shorter time than it takes for the initial solution, the initial solution is returned when it is computed. Increasing the number of climbers will increase the time it takes to compute the initial solution.

17. Can cuOpt be combined with Reinforcement Learning?

cuOpt is an optimization solver but it can accept initial guesses from ML models. This is still in the exploratory phase as it requires extensive customization to a particular dataset.

18. If the time limit is set to X seconds and the solver completes earlier, will it still run for X seconds?

Yes, the solver will continue to run for X seconds looking for more solutions.

In the `set_pickup_delivery_pairs()`, see an error - ValueError: Pickup/delivery indices size is not equal to number of pairs from this function.

In cuOpt, the depot is always node 0 and it should be included in most APIs. The pickup delivery pairs must have a size of (orders+1)/2 (+1 comes from the depot).

19. Do we need to normalize the data when creating a time window matrix?

The units can be whatever the customer wants them to be minutes, seconds, milliseconds, hours, and so on. It is the user's responsibility to normalize the data.

20. How does cuOpt check for collisions in NVIDIA Isaac Sim?

Dynamic obstacle avoidance (humans, other robots, and so on) is handled by local planners on the robots because these events cannot be foreseen when cuOpt gets the data for a problem. For static obstacles (shelves, walls, etc.) the waypoint graph describes areas that are traversable by the robot, therefore by default, the robots should not hit static obstacles as cuOpt will only return locations that are present in the waypoint graph.

21. While setting certain customer nodes as break points, using add_break_dimensions(), there is an error where the result has additional locations.

This is not an error. Breakpoints get appended to the existing location list. You can refer to the locations by the location ids.

22. How can we set the demand of a customer node higher than the capacity of the vehicle?

If this is for one node, then multiple vehicles can deliver a bunch of orders. However, if one order demands more capacity than one vehicle can handle, then we will have to use a workaround by assigning to dummy vehicles.

23. How to ensure that cuOpt does not assign high priority orders to dummy vehicles or technicians (that were added to get feasible solutions)?

We can set `vehicle_order_match(v, o) = 0` for when o is an important task and v is a dummy vehicle.

Note that setting vehicle order match makes it a hard constraint. So if there are many high-priority orders/tasks, it may end up with infeasible solutions. But, if the high-priority orders are only a fraction of the total orders, then these solutions should work.

24. When should I use the microservice vs Python API?

From a functional perspective, the server option is equivalent to the Python APIs. There are some differences in how data is grouped, for example `set_task_data` and `set_fleet_data` tie together a bunch of data that is interdependent whereas in the Python API there is a separate call for each item. The server APIs also take data as JSON instead of cuDF DataFrames and Series.

The major advantage of the server API of course is availability of a single system with a GPU and where problems from multiple non-gpu systems can be submitted.

25. **While calling API `/cuopt/get_optimized_routes`, we get an error {"detail":"Failed to find solution with given constraints"}. Is it possible to know due to which constraint this failed?**

It is not possible to prioritize constraints with R22.12. So even if one constraint cannot be satisfied, the solver returns failure and an empty route. With drop infeasible tasks enabled, the solver does not fulfill all the customer orders and returns a partial route and a list of unfulfilled orders. You can also provide more vehicles in the fleet data to get a feasible solution if it makes sense for your problem.

26. **How can I add multiple capacity information for a specific vehicle—height, widths and length, mileage, carbon footprint, and so on.**

It is possible to add multiple capacities. These are represented as capacity dimensions in the solver. For each capacity constraint, you can call `data_model.add_capacity_dimension` if you are using python API or have multiple columns in task_data["demand"] and `fleet_data["capacities"]` if you are using server API. For each dimension, there should be one vector capacity of length `fleet_size` and demand of length `num_orders/tasks`.

27. **Is there a way to prevent vehicles from traveling along the same path in a waypoint graph, or is there a way to prevent more than one vehicle from visiting a location, or even that a location is only visited one time by a single vehicle?**

Currently we do not have such restrictions, and cuOpt tries to optimize for the fewest number of vehicles as the primary default objective. However, spatiotemporal restrictions continue to be a topic of research.

28. **Can I use my own data instead of Homberger's instance data in cuOpt?**

Yes, you can certainly use your own data. One way to do it would be to structure your data in the same format as the Homberger data files and use the cuOpt utilities to read it.

Another way is to construct your cost matrices, and so on, directly in Python. If you look at the example notebooks in the `ea-cuopt` image, you can see how to do this. For example, the `cost_matrix_creation.ipynb` notebook shows a simple example of constructing a cost matrix from literal values.

29. **`RuntimeError: cuOpt failure at file=/cuopt-build-utilities/cuopt/cpp/src/solution/get_solution.cu line=196: Truck capacity is smaller than volume.`**

This is not accuracy as there is one vehicle with a capacity of 750 can carry a demand of 713 and another can carry a demand of 434 (considering that we have 70 vehicles available with no time restrictions or anything extra, only capacity restriction).

The problem was in the order in which we passed the data frame with the locations to go through, for display purposes we were ordering from highest to lowest according to demand, leaving the largest values in the first position (wrongly representing the deposit), the solution was to always insert the deposit in the first position with zero demand.

30. **The property `num_of_climbers: int`. What number is a typical value? What is the default value?**

By default, the number of climbers is chosen by considering occupancy of a small GPU and experimented runtime vs number of climbers trade-off (that is, the best result in shortest time). Normally 1024 or multiple is a good start.

31. **Travel time deviation—When using the same dataset, the travel time varies by a couple of seconds in different runs, but the distance remains the same. How can travel time deviate in multiple runs on the same data and distance remains constant?**

This is because travel time is not part of the objective so we could have two solutions that are equivalent and when picking the best solution in this case it is not deterministic. You can include travel time, including waiting times as part of the default objective. You can also include as part of a predefined set of objectives as an extra comparator. cuOpt optimizes for fleet size and cost so these solutions are equivalent from the solver perspective. Or you can set time as your cost matrix.

32. **Null values in the cost matrix**

Replace null values with some high value, not max of float type.

33. **I cannot model road conditions like traffic.**

cuOpt accepts arbitrary distance and time matrices as input. This data typically comes from popular map engines (Google Maps, open route service, and so on) and may account for current traffic information natively. The matrix can also get updated based on current conditions and cuOpt can reoptimize the new problem quickly.

34. **Can a break happen anywhere?**

If break locations are provided, breaks can only be taken in those points, by default break can be taken in any location.

35. **Time windows issues**

Time window and any other time-related data are accepted in the form of float, so the time in date/time format or string format must be converted to floating value. (Example: 9:00 am - 6:00 pm, Converting this to minutes in a 24-hour period starting at 12:00 am, 540 - 1080).

All time/cost unit provided to cuOpt should be in the same units.

36. **What is the difference between `set_min_vehicles` and the horizontal loading feature?**

`set_min_vehicles` indicates the solver to use a predefined number of vehicles. This is a hard constraint. One would want to do this to use all the resources (fleet). The solver would still want to optimize for total cost while using these many vehicles. Usually, hard constraints are bad for finding optimal solutions, because you are blocking the solver from exploring wider space.

Horizontal loading tries to minimize the variance between routes). It will penalize any technician or vehicle doing many more tasks than the avg tasks or for doing much fewer tasks than the avg tasks. So if you penalize enough, you should get a route with all techs doing a similar number of jobs.

37. Does cuOpt implement a topo sort for every route map?

This is not currently supported. Future consideration.

38. How to choose various solution strategies?

Available strategies are HILL_CLIMBING, TABU_SEARCH, TABU_GLS, default is TABU_GLS.

39. Floating point vs integers when using the Set Task data server API call.

https://docs.nvidia.com/cuopt/cuopt_server.html?operation/set_task_data_cuopt_set_task_data_post#operation/set_cost_waypoint_graph_cuopt_set_cost_waypoint_graph_post

It says `task_locations` should be integers. But in real world, latitude, longitude coordinates are floating point values.

cuOpt expects that a user provides either:

1. a cost matrix and corresponding location indices.
2. a waypoint graph and locations corresponding to waypoints as integers.

If you have (lat, long) values, then one can generate a cost matrix using a map API. cuOpt does not directly connect to a third-party map engine, but that can be easily done outside cuOpt as shown in the example below:

https://github.com/NVIDIA/cuOpt-Resources/blob/branch-22.10/notebooks/routing/python/cost_matrix_creation.ipynb

cuOpt does not directly connect to a third-party map engine, but that can be easily done outside cuOpt as shown in the example below:

https://github.com/NVIDIA/cuOpt-Resources/blob/branch-22.10/notebooks/routing/python/cost_matrix_creation.ipynb

40. How do you compute the max driving that does not include wait time or service time)?

We have `set_vehicle_max_times`, with which you can control the max driving (does not include wait time, service time). This uses `time_matrix` to compute time so the units pertain to time and not distance.

41. When do I use `set_vehicle_max_costs` vs `set_vehicle_max_times`?

`set_vehicle_max_costs` uses cost matrix; cost can be distance.

`set_vehicle_max_times` uses a time matrix.

42. How do I calculate and adjust the optimal route while avoiding collisions using cuOpt and Issac Sim?

There is a distinction between path planning in cuOpt and local dynamic obstacle avoidance. The latter deals with any local dynamic situation that is not represented in the environment—such as a human, unexpected pallets, and so on—cuOpt handles aspects of the environment, static or dynamic, that can impact global navigation. Highly transit, localized events are handled by the local planning systems on the robot, and they follow the route provided by cuOpt.

In this case, cuOpt would consider the routes as part of the environment and factor them in global planning. If the zones become unrestricted due to any reason and the environment is updated on the cuOpt server, the subsequent optimizations will reconsider the changes. The question related to acceleration or deceleration would fall under dynamic collision avoidance. It is up to the user to reoptimize based on any user-defined triggers such as automated environment monitoring or timer.

43. Is it possible to define constraints such as refrigerated vehicles required for certain orders?

Yes, you can define constraints to match vehicles to order type. Frozen goods are a great example!

44. Are there any plans to allow each vehicle to have its own maximum route distance so that mixed fleets with electric and traditional vehicles can be optimized?

Today, we can set a max distance per vehicle per route—look at the following API to see if this works for your use case.

```
set_max_distance_per_route(*max_distance_per_route*)
```

45. What is the role of `set_order_location()` in pickup delivery combination?

```
pickup_indices   = [0, 2, 4]
```

delivery_indices = [1, 3, 5] # 2 -> 1, 3 -> 4 and 1->4

Here the pickup and delivery pairs are indices to the order locations provided, considering following example from the doc:

- order_locations = [2, 1, 3, 4, 1, 4]
- pickup_indices = [0, 2, 4]
- delivery_indices = [1, 3, 5] # 2 -> 1, 3 -> 4 and 1->4

Each pickup and delivery are considered an order, so we have six orders, three pickups, and three deliveries.

Pickup and Delivery are indices to order locations, 0 in pickup indices correspond to value in 0th index in order locations, so it depicts location - 2, similarly, index 4 represents location -1.

For something like pickup and delivery pair locations, (2 -> 1), (3->4), (1->4), pickup and delivery pair indices into order location would be (0->1), (2->3), (4->5).

46. How to model inputs for following scenarios?

Same truck picks up and delivers some orders on the way.

If we want to tie few orders to particular trucks, we can use vehicle order `add_vehicle_order_match`.

If this is a generic question on whether a truck can pick up multiple orders, then it can.

47. How do we model the following scenario—Pick up from multiple different locations and deliver to a single customer.

Pickup from multiple different locations and delivering to a single customer will be handled by algorithm depending on how far the orders are and whether using multiple trucks would reduce the cost.

If we want to restrict it to a single truck, then use `add_vehicle_order_match`.

It can also be modeled as a pickup-delivery problem. Suppose that there are four pickups from different locations and drop off to one location.

There will be four pickup-delivery pairs (eight orders in total).

The solver can then figure out an optimal way to serve these orders/requests.

If these have to be served by only a specified vehicle, use `add_vehicle_order_match` API.

48. We enabled the following "error_logging": "verbose_mode": true but still did not get any reason why it failed?

The solver does not report the details of the error at the moment when using the server API.

49. I know that the problem has a feasible solution, but cuOpt fails with infeasible. How to avoid this?

The optimizer first tries to find feasible solutions that may be sub-optimal and then try to optimize these feasible solutions. If the problem is tightly constrained, it is possible that the solver may not find initial feasible solutions. If this happens, solver immediately returns infeasible. Another option is to relax the constraints or provide more vehicles to get a feasible solution.

50. Where can I learn more about cuOpt?

Here are links to learn about various aspects of cuOpt:

- docs.nvidia.com
- developer.nvidia.com
- forums.developer.nvidia.com
- catalog.ngc.com
- github

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com