

CVRP solver in Python

Benchmark

- Uchoa et al. (2014)
- Download: <http://vrp.atd-lab.inf.puc-rio.br/index.php/en/>
- We will test these data and find biggest data these functions can solve in 5 minutes
- Lastly, we will use data with different client number to compare these librarys performance (time and accuracy)

librarys

- PyVRP
- RoutingBlocks
- VRPSolverEasy

PyVRP

- Wouda, N.A., L. Lan, and W. Kool (2024). PyVRP: a high-performance VRP solver package. INFORMS Journal on Computing, forthcoming.
<https://doi.org/10.1287/ijoc.2023.0055>
- Installation
 - `pip install pyvrp`
 - To get the latest version
 - `pip install 'pyvrp @ git+https://github.com/PyVRP/PyVRP'`

How to use

- create model
 - from pyvrp import Model
 - m = model()
- add vehicle: m.add_vehicle_type()

```
add_vehicle_type(  
    num_available: int = 1,  
    capacity: int = 0,  
    depot: Client | None = None,  
    fixed_cost: int = 0,  
    tw_early: int = 0,  
    tw_late: int = 2147483647,  
    max_duration: int = 2147483647,  
    name: str = ''  
) → VehicleType
```

How to use

- add depot: m.add_depot()

```
add_depot(  
    x: int,  
    y: int,  
    tw_early: int = 0,  
    tw_late: int = 2147483647,  
    name: str = ''  
) → Client
```

- add clients: m.add_client()

```
add_client(  
    x: int,  
    y: int,  
    demand: int = 0,  
    service_duration: int = 0,  
    tw_early: int = 0,  
    tw_late: int = 2147483647,  
    release_time: int = 0,  
    prize: int = 0,  
    required: bool = True,  
    name: str = ''  
) → Client
```


How to use

- add edge: `m.add_edge()`

```
add_edge(  
    frm: Client,  
    to: Client,  
    distance: int,  
    duration: int = 0  
) → Edge ¶
```

How to use

- More easy way to load VRPLIB format data and create model
 - from pyvrp import read
 - INSTANCE = read("data/X-n439-k37.vrp", round_func="round")
 - m = Model.from_data(INSTANCE)

```
read(  
    where: str | Path,  
    instance_format: str = 'vrplib',  
    round_func: str | Callable[[ndarray], ndarray] = 'none'  
) → ProblemData  \[source\]
```


How to use

`instance_format`: File format of the instance to read, one of 'vrplib' (default) or 'solomon'

`round_func`: Optional rounding function. Will be applied to round data if the data is not already integer. This can either be a function or a string:

1. `round` rounds the values to the nearest integer;
2. `trunc` truncates the values to be integral;
3. `trunc1` or `dimacs` scale and truncate to the nearest decimal;
4. `none` does no rounding. This is the default.

How to use

- Solve
 - `solution = m.solve()`

```
solve(  
    stop: StoppingCriterion,  
    seed: int = 0,  
    display: bool = True  
) → Result
```

1. StoppingCriterion:

1. `import(use maxRunTime as instance): from pyvrp.stop import MaxRuntime`
2. `MaxIterations(int)`: stops after a maximum number of iterations
3. `MaxRuntime(float)`: stops after a maximum number of iterations
4. `MultipleCriteria(list[StoppingCriterion])`: Simple aggregate class that manages multiple stopping criteria at once
5. `NoImprovement(int)`: stops if the best solution has not been improved for a fixed number of iterations.

How to use

- Solution structure

```
class Result(  
    best: Solution,  
    stats: Statistics,  
    num_iterations: int,  
    runtime: float  
)
```

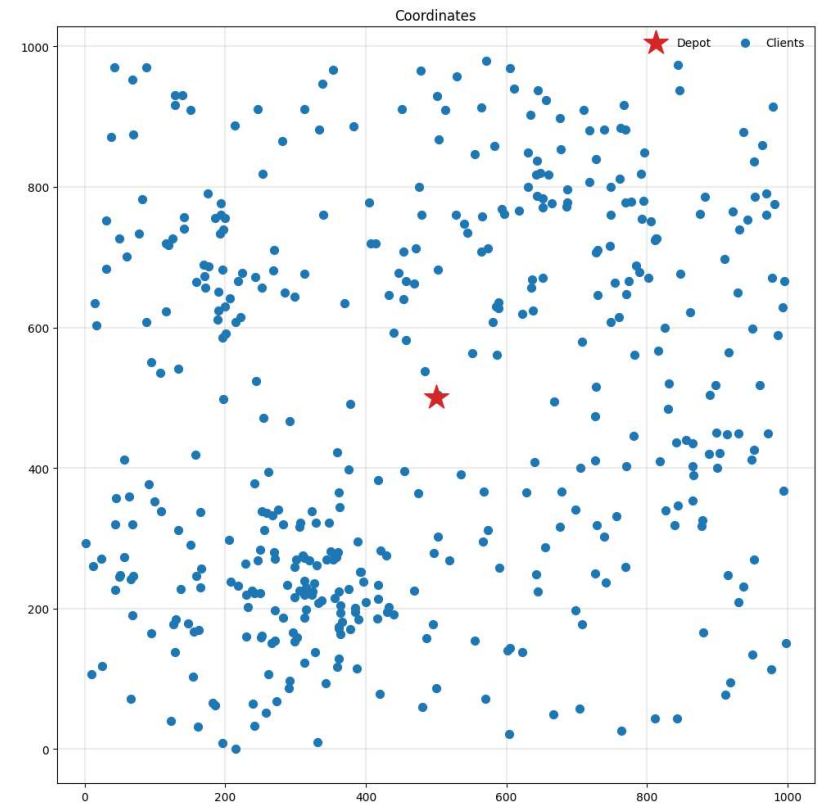
- total cost: solution.cost()
- feasible solution ? : solution.is_feasible()

Performance

- Get solution in 5 minutes: n10001-k43
- Found a solution with cost: 74919.
- This is 3.5% worse than the best known solution, which is 72355.
- Time: 260 seconds (4m 20s)

Plot

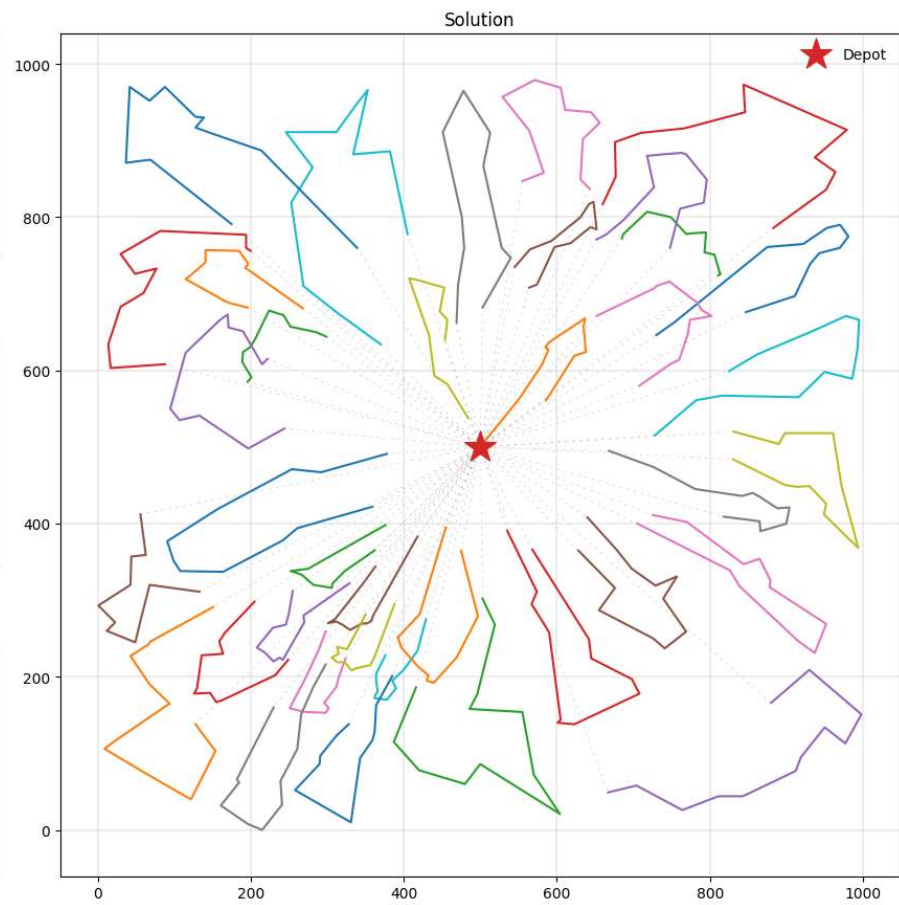
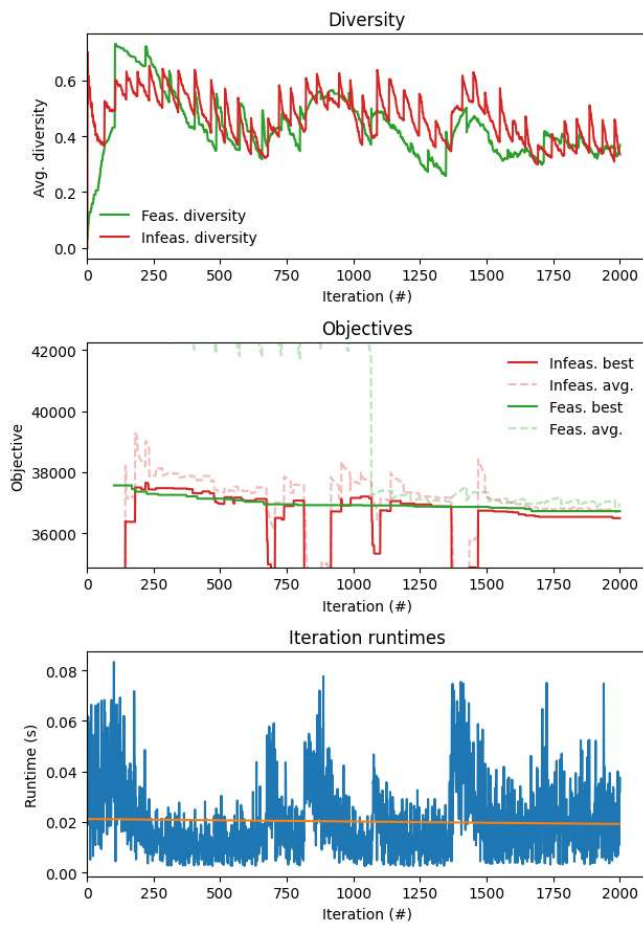
- plot coordinates:
 - import matplotlib.pyplot as plt
 - from pyvrp.plotting import plot_coordinates
 - _, ax = plt.subplots(figsize=(10, 10))
 - plot_coordinates(INSTANCE, ax=ax)



Plot

- plot solution:
 - `from pyvrp.plotting import plot_result`
 - `fig = plt.figure(figsize=(15, 9))`
 - `plot_result(solution, INSTANCE, fig)`
 - `fig.tight_layout()`

Plot



RoutingBlocks

- Patrick S. Klein, Maximilian Schiffer (2024) RoutingBlocks: An Open-Source Python Package for Vehicle Routing Problems with Intermediate Stops. INFORMS Journal on Computing 0(0).<https://doi.org/10.1287/ijoc.2023.0104>
- This library is more like a framework. It provides us an interface to let us create plugin by ourselves. Thus, if you want to solve new question (such as CVRP but has some new constraints), RB is a good choice

RoutingBlocks

- What is meaning of plug-in?
 - RB allow us to create evaluation methods by ourselves
 - The evaluation class implements problem-specific cost and move evaluation functions.
- How to create
 - <https://github.com/tumBAIS/routingblocks-native-extension-example.git>
 - download this base plugin
 - modify 'src/CVRPEvaluation.cpp'
 - Install this plugin by pip

RoutingBlocks

- Install library
 - ``pip install routingblocks``
 - latest version: ``pip install git+https://github.com/tumBAIS/RoutingBlocks``
- Install CVRP plug-in (provided by official)
 - `git clone https://github.com/tumBAIS/routingblocks-native-extension-example.git`
 - go to ``<directory>/pyproject.toml``, modify the ``name`` in [project] to ``name = "routingblocks_cvrp"``
 - `pip install `<directory>``

RoutingBlocks

```
[project]
name = "RoutingBlocks-CVRP"
version = "0.1.2"
description = "Native CVRP evaluation for the routingblocks package"
readme = "README.md"
authors = [
    { name = "Patrick Sean Klein", email = "patrick.sean.klein@tum.de" },
]
dependencies = [
    "routingblocks"
]
```

How to use

- Complex. RB doesn't provide us function like pyVRP. I only introduce some important function.
- read data: `vrplib.download_instance(file_path)`
- create model: depends on your `evaluation.cpp`

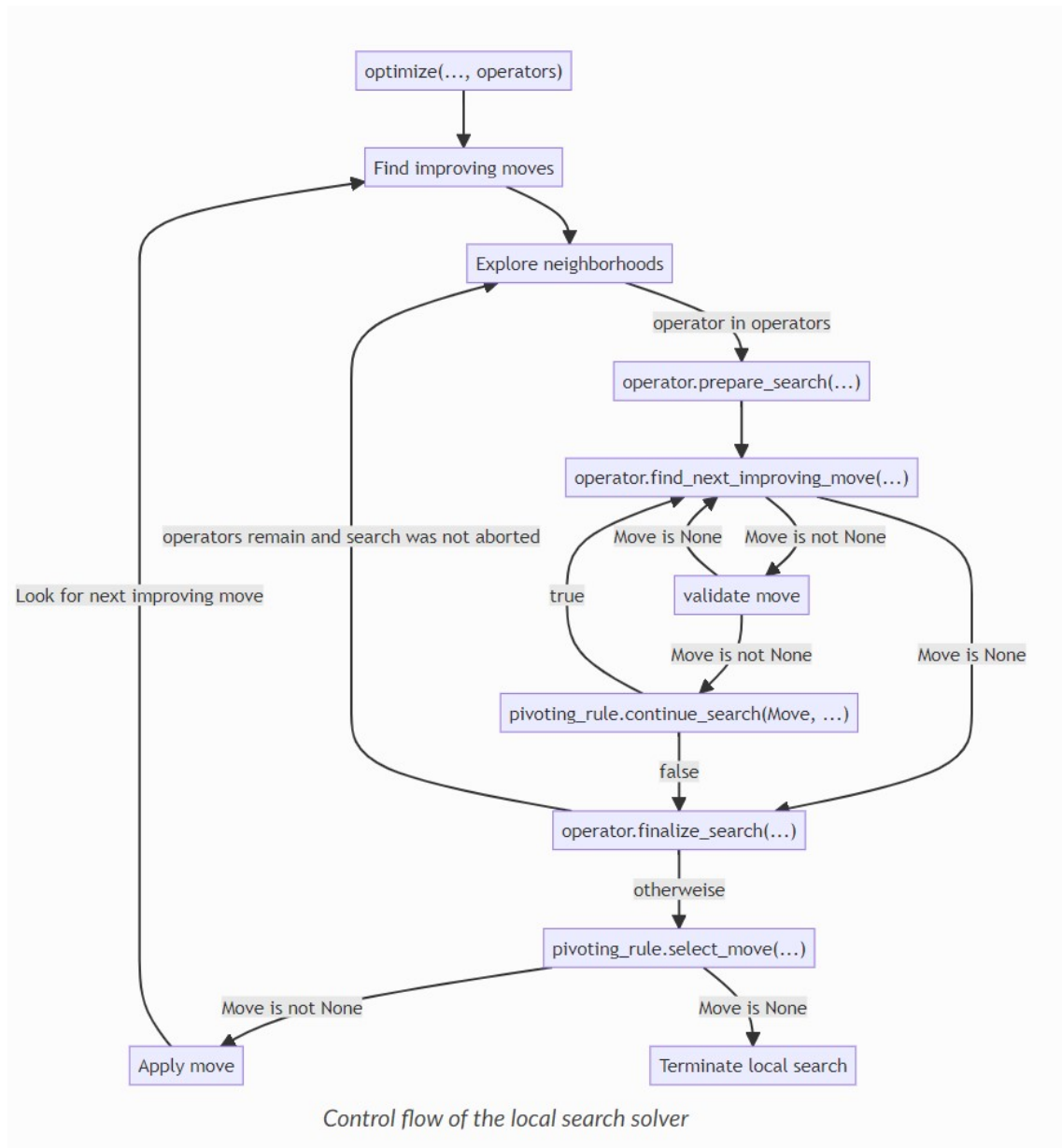
```
def create_cvrp_instance(instance):  
    # Create CVRP vertices  
    n = len(instance['demand'])  
    vertices = [  
        cvrp.create_cvrp_vertex(i, str(i), False, i == instance['depot'][0], cvrp.CVRPVertexData(instance['demand'][i]))  
        for i in range(n)]  
    # Create CVRP arcs  
    arcs = [  
        [cvrp.create_cvrp_arc(cvrp.CVRPArcData(instance['edge_weight'][i][j])) for j in range(n)] for i in range(n)]  
    ]  
  
    return routingblocks.Instance(vertices, arcs, len(vertices))
```

How to use

- create solver: `routingblocks.LocalSearch(instance, evaluation, exact_evaluation, pivoting_rule)`
- solve: `solver.optimize(solution, operators)`

```
def optimize_solution(evaluation: routingblocks.Evaluation, instance: routingblocks.Instance,
                      solution: routingblocks.Solution):
    # Create a local search solver
    solver = routingblocks.LocalSearch(instance, evaluation, evaluation, routingblocks.BestImprovementPivotingRule())
    # Create some operators
    # Create the arc set - by default all arcs are included
    full_arc_set = routingblocks.ArcSet(len(instance))
    operators = [
        routingblocks.operators.SwapOperator_0_1(instance, full_arc_set),
        routingblocks.operators.SwapOperator_1_1(instance, full_arc_set),
        routingblocks.operators.InterRouteTwoOptOperator(instance, full_arc_set)
    ]
    # Optimize the solution (inplace)
    solver.optimize(solution, operators)
    return solution
```

How to use



Performance

- Get solution in 5 minutes: n491-k59
- Found a solution with cost: 70853.5390625.
- This is 2.4% worse than the best known solution, which is 69226.
- Time: 259 seconds (4m 19s)

VRPSolverEasy

- <https://github.com/inria-UFF/VRPSolverEasy.git>
- Installation
 - `python -m pip install VRPSolverEasy`
 - Request bapcod: <https://bapcod.math.u-bordeaux.fr/>
 - go to the VRPSolverEasy folder and copy the system folder corresponding to your computer and copy it into the lib folder of the VRPSolverEasy python package.

How to use

- create model
 - `model = VRPSolverEasy.Model()`
- set parameters: `model.set_parameters()`
 - `time_limit`: easysolver is solver so we have to set a high value when faced with large data (client >100)

Parameter	Explanation	Type	Default
<code>time_limit</code>	Time limit for the solver (in seconds)	<code>float > 0</code>	300
<code>upper_bound</code>	Cutoff value (i.e., only solutions with smaller value will be searched for)	<code>float</code>	10^6
<code>heuristic_used</code>	Switch on/off built-in heuristic	<code>bool</code>	<code>False</code>
<code>time_limit_heuristic</code>	Time limit for a run of the built-in heuristic (in seconds)	<code>float</code>	20
<code>solver_name</code>	Which underlying LP/MIP solver is used	<code>str</code>	"CLP"
<code>print_level</code>	Verbosity of the solver (-2: no output; -1: reduced output; 0: normal output)	<code>int ∈ {-2, -1, 0}</code>	-1

How to use

- set parameters: `model.set_parameters()`
 - `upper_bound`:
 - The solver is focused on improving lower bounds and proving optimality of a known feasible solution.
 - Performance of the solver may be greatly improved by setting the upper bound parameter to the value of a known feasible solution
 - we advice to run a (good) external heuristic before launching the solver.
 - for example, run pyVRP first, get a solution with some error and then use this solution cost as `upper_bound` of EasyVRPSolver

```
result_es = use_EasySolver("X-n106-k14", known_feasible_solution = result_pyvrp.cost())
```

✓ 5m 0.6s

```
result_es2 = use_EasySolver("X-n106-k14")
```

✓ 5m 0.7s

Found a solution with cost: 26373.047446769713.
This is 0.0% worse than the best known solution, which is 26362.

How to use

- More details in my week1 folder note.md
- add depot:
 - ``model.add_depot(id,name="",service_time=0.0,tw_begin=0.0,tw_end=0.0)``
- add clients:
 - ``model.add_customer(id,id_customer=,name="",demand=0,penalty=0.0,service_time=0.0,tw_begin=0.0,tw_end=0.0,incompatible_vehicles=[])``
- add edge:
 - ``add_link(start_point_id=0, end_point_id=0, name="", is_directed=False, distance=0.0, time=0.0, fixed_cost=0.0)``
- add vehicles:
 - ``model.add_vehicle_type(id,start_point_id=-1,end_point_id=-1,name="",capacity=0,fixed_cost=0.0,var_cost_dist=0.0,var_cost_time=0.0,max_number=1,tw_begin=0.0,tw_end=0.0)``

How to use

- solve: `model.solve()`
- solution: `model.solution` and `is_defined`
 - This solver will only output accurate solutions
 - Thus, if it doesn't get optimal solution, the return of `model.solve()` will not contain any information of routes and cost
 - So, check solution at first before you use it:

```
model.solution.is_defined()
```

```
True
```

```
print(model.solution)
```

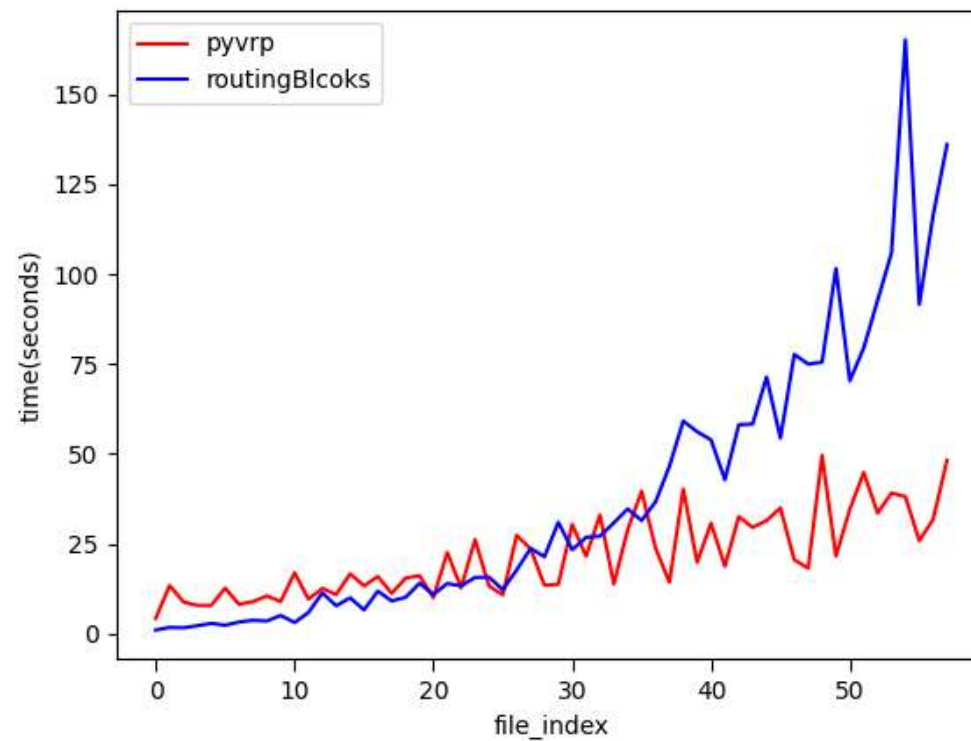
```
Solution cost : 375.2797871556646
```

Performance

- Get solution in 5 minutes: n106-k14
- Found a solution with cost: 26373.047446769713.
- This is 0% worse than the best known solution, which is 26362.
- Time: 300 seconds (5m)

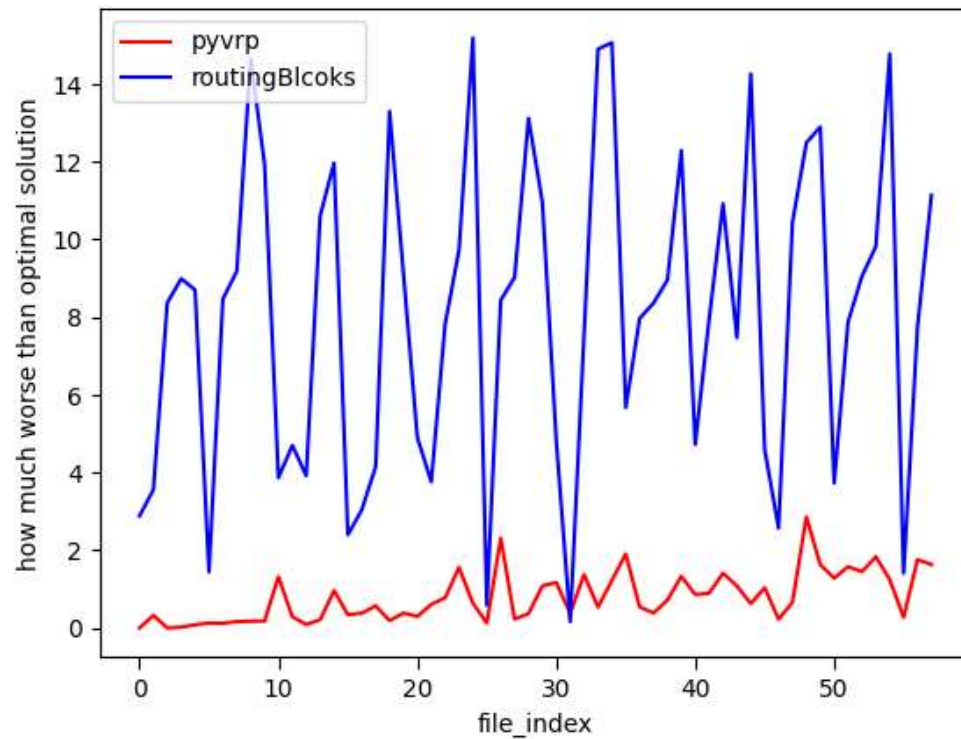
Performance comparison

- Time: start from index 35 (X-n266-k58) , RB's time is larger than pyVRP

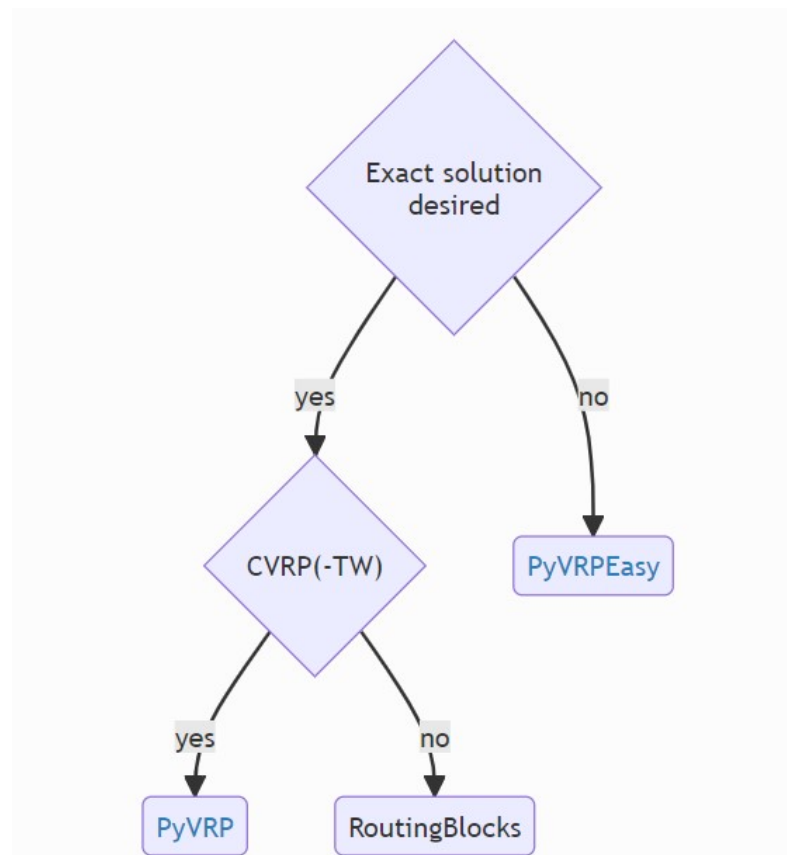


Performance comparison

- Accuracy: pyvrp much better than rb



Guide from RoutingBlocks



- You could find all my files in:
 - <https://github.com/DOHZH/IE597Spring2024>
- Thank you!