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

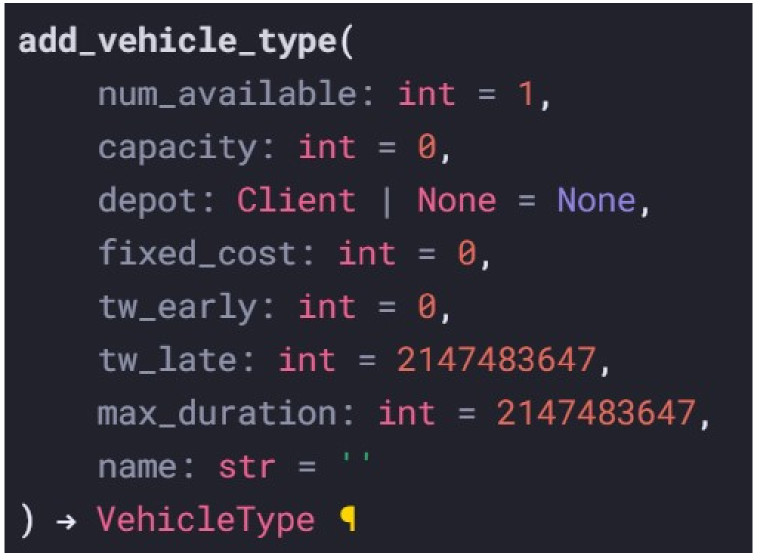
- PyVRP
- RoutingBlocks
- VRPSolverEasy

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 lateset version
    - pip install 'pyvrp @ git+https://github.com/PyVRP/PyVRP'

How to use

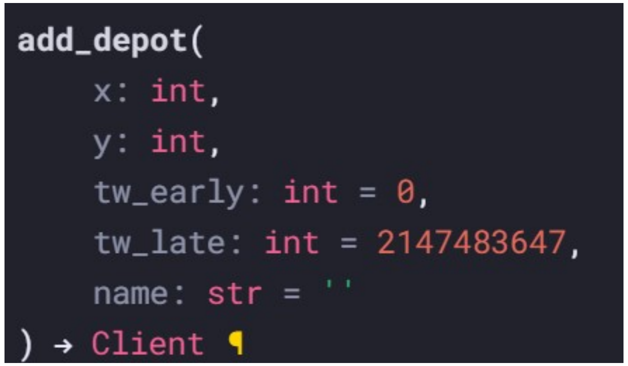
- create model
  - from pyvrp import Model
  - m = model()
- add vehicle: m.add_vehicle_type()



```
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 ¶
```

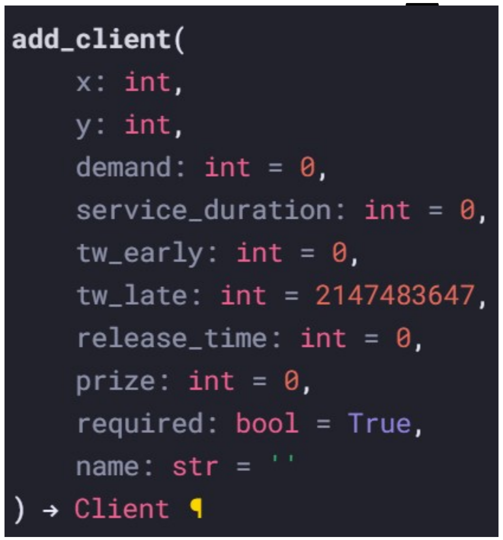
How to use

- add depot: m.add_depot()



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

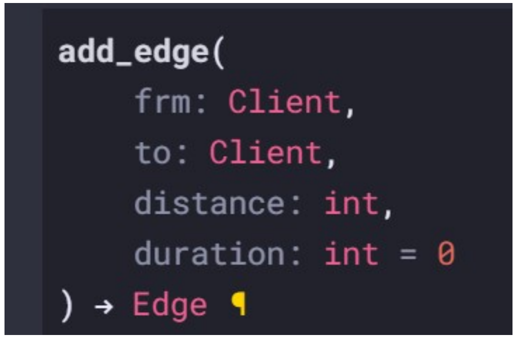
- add clients: m.add_client()



```
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

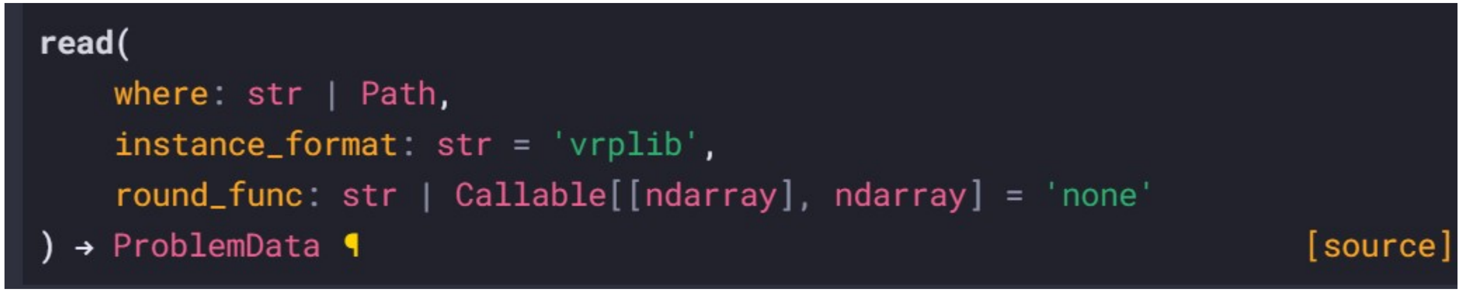
- add edge: m.add_edge()



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

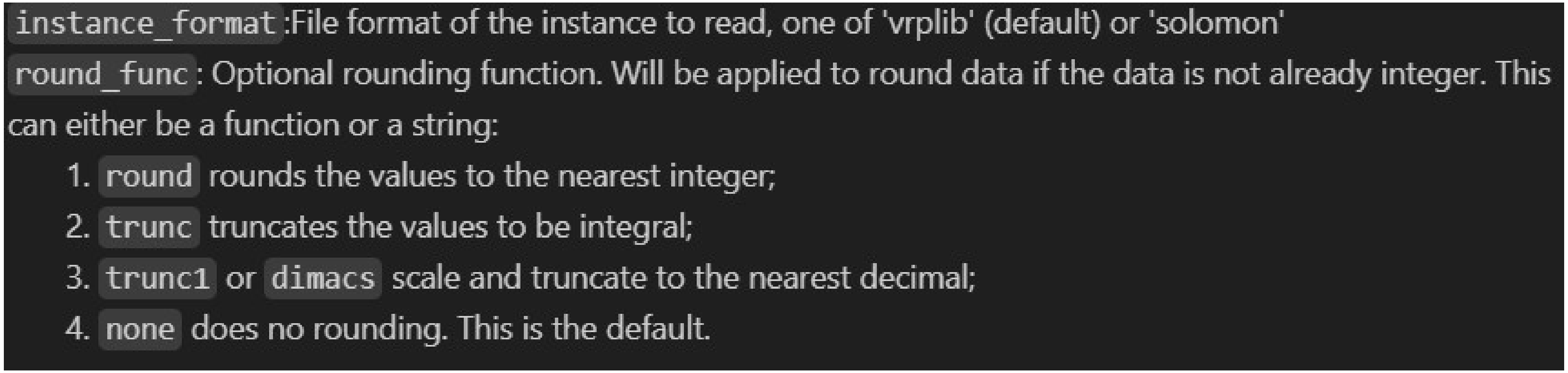
How to use

- 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



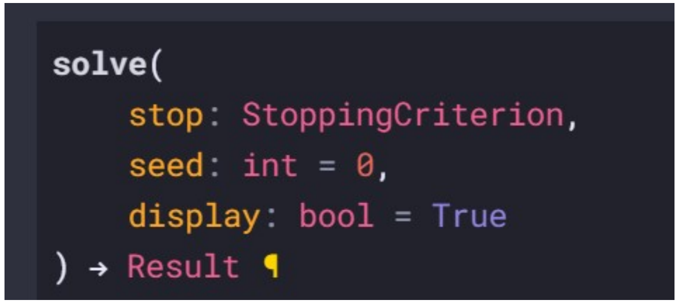
`instance_format`:File format of the instance to read, one of 'vrplib' (default) or 'solomon'

`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:

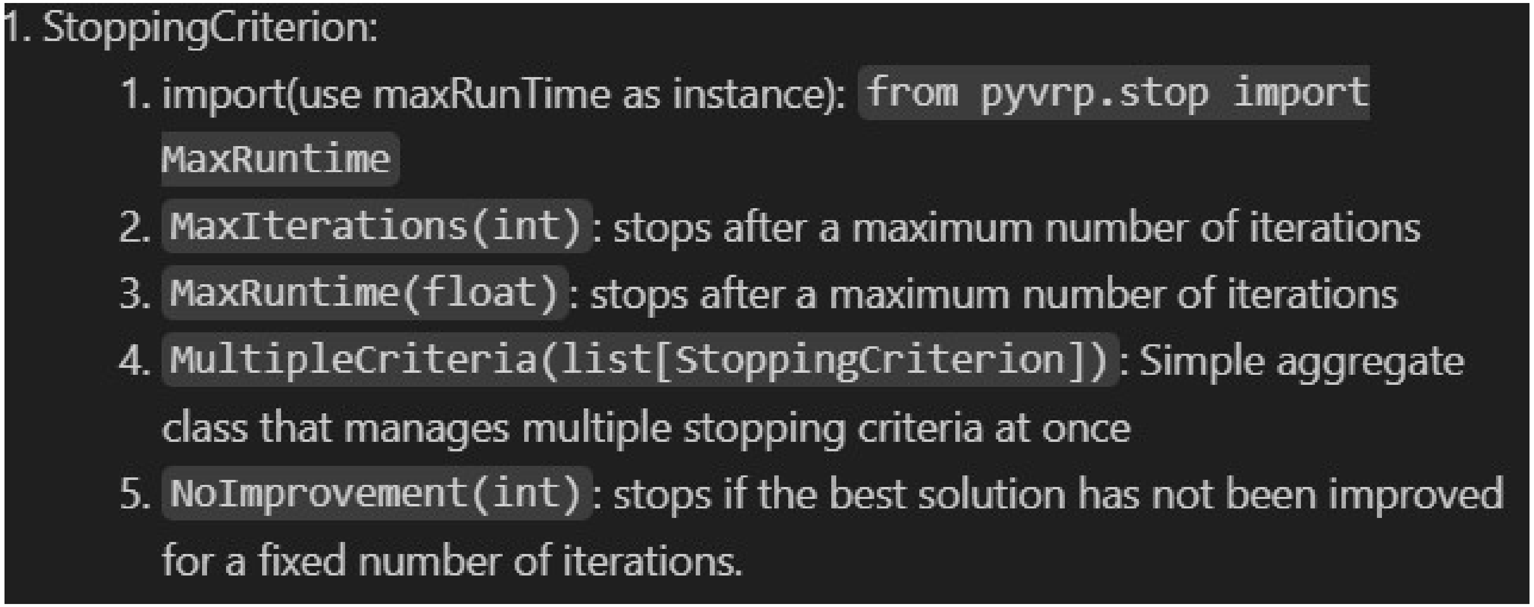
1. `round` rounds the values to the nearest integer;
2. `trunc` truncates the values to be integral;
3. `trunc1` or `dimacs` scale and truncate to the nearest decimal;
4. `none` does no rounding. This is the default.

How to use

- Solve
  - solution = m.solve()



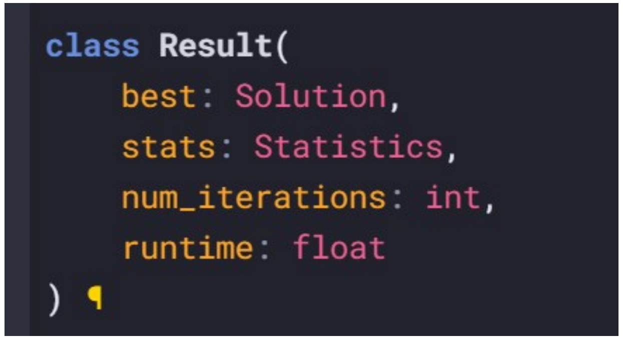
```
solve(
    stop: StoppingCriterion,
    seed: int = 0,
    display: bool = True
) → Result ¶
```



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
) ¶
```

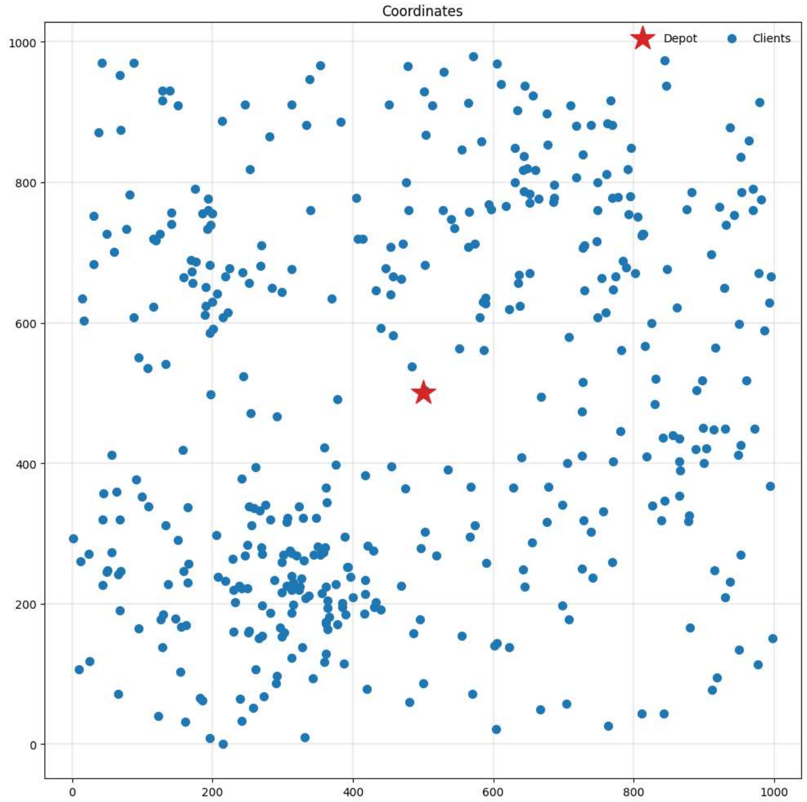
  - total cost: solution.cost()
  - feasible soultion ?: solution.is_feasible()

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

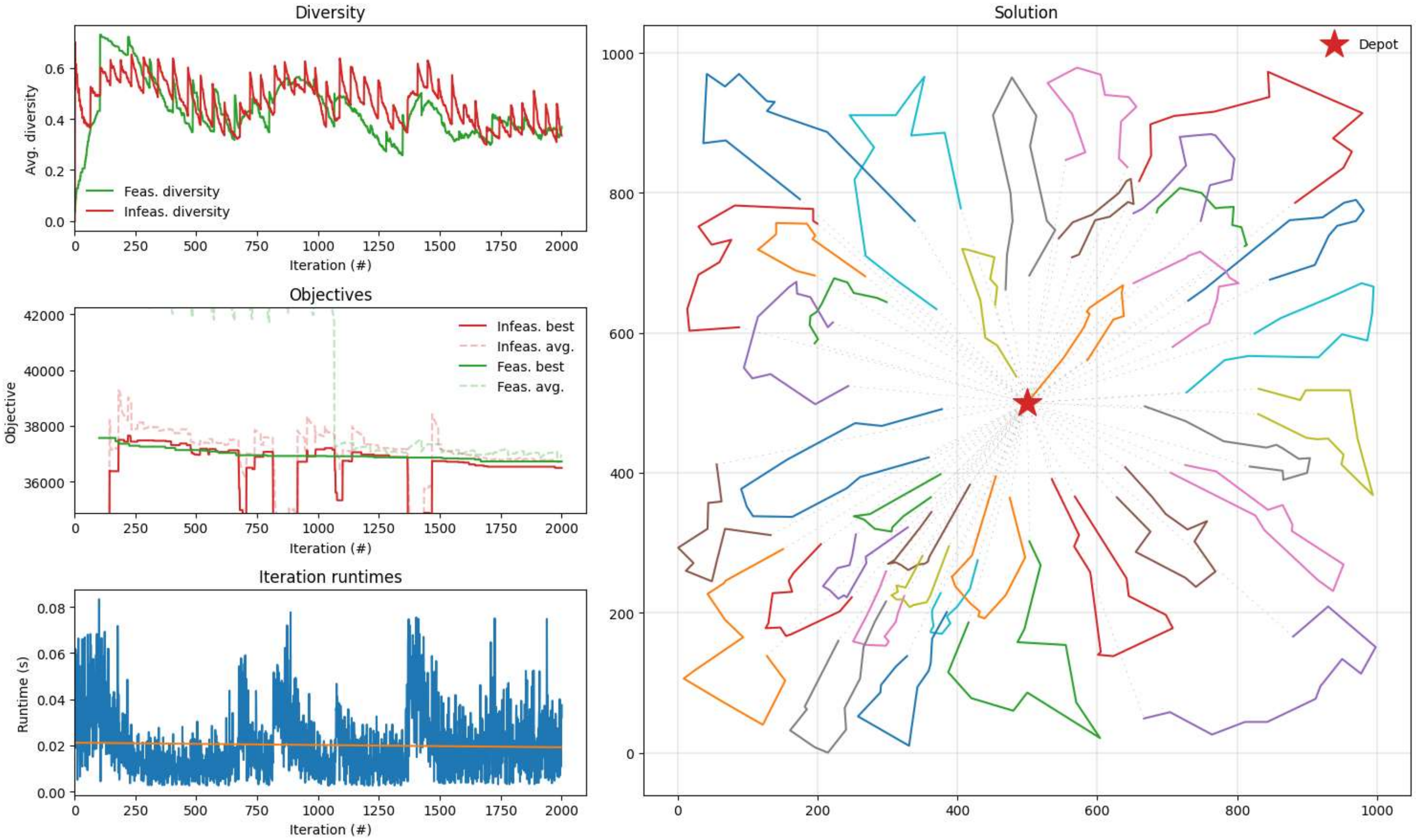
- plot coordinates:
  - import matplotlib.pyplot as plt
  - from pyvrp.plotting import plot_coordinates
  - _, ax = plt.subplots(figsize=(10, 10))
  - plot_coordinates(INSTANCE, ax=ax)



Plot

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

Plot


Diversity
Avg. diversity
0.0
0.2
0.4
0.6
Feas. diversity
Infeas. diversity
0
250
500
750
1000
1250
1500
1750
2000
Iteration (#)
Objectives
Objective
36000
38000
40000
42000
Infeas. best
Infeas. avg.
Feas. best
Feas. avg.
Iteration (#)
Iteration runtimes
Runtime (s)
0.00
0.02
0.04
0.06
0.08
Iteration (#)
Solution
Depot
0
200
400
600
800
1000

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

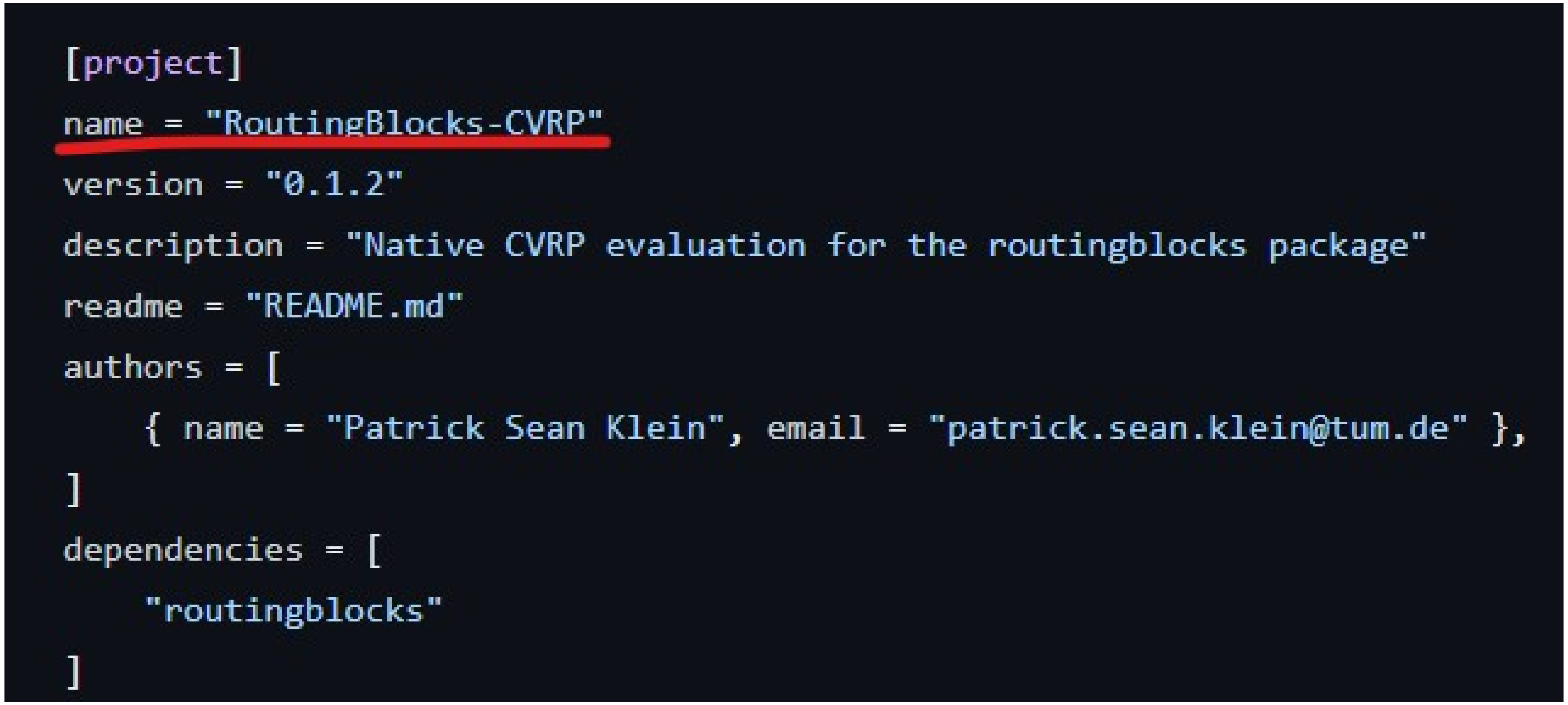
RoutingBlocks

- 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 `<dirctory>/pyproject.toml`, modify the `name` in [project] to `name = "routingblocks_cvrp"`
  - pip install `<dirctory>`

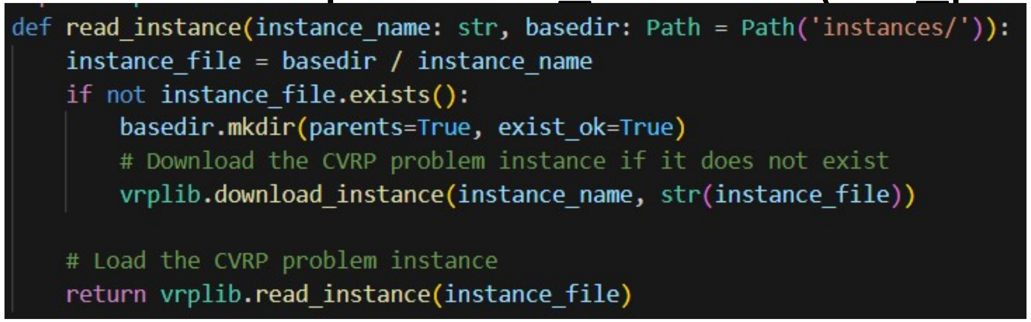
RoutingBlocks



```
[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"
]
```

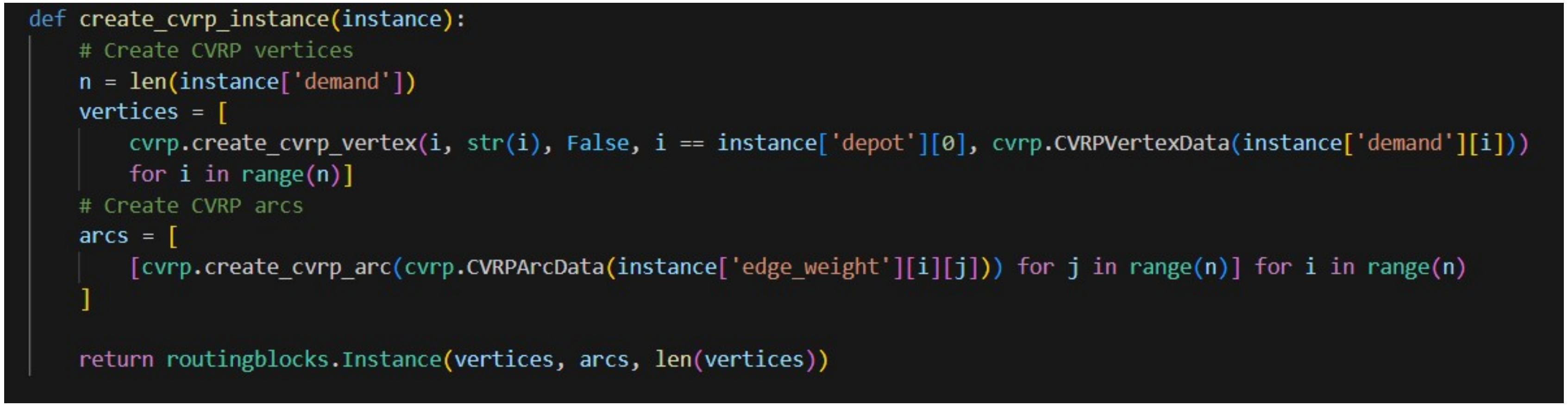
How to use

- read data: vrplib.read_instance(file_path)



```python
def read_instance(instance_name: str, basedir: Path = Path('instances/')):
    instance_file = basedir / instance_name
    if not instance_file.exists():
        basedir.mkdir(parents=True, exist_ok=True)
        # Download the CVRP problem instance if it does not exist
        vrplib.download_instance(instance_name, str(instance_file))

    # Load the CVRP problem instance
    return vrplib.read_instance(instance_file)
```

- create model: depends on your evaluation.cpp



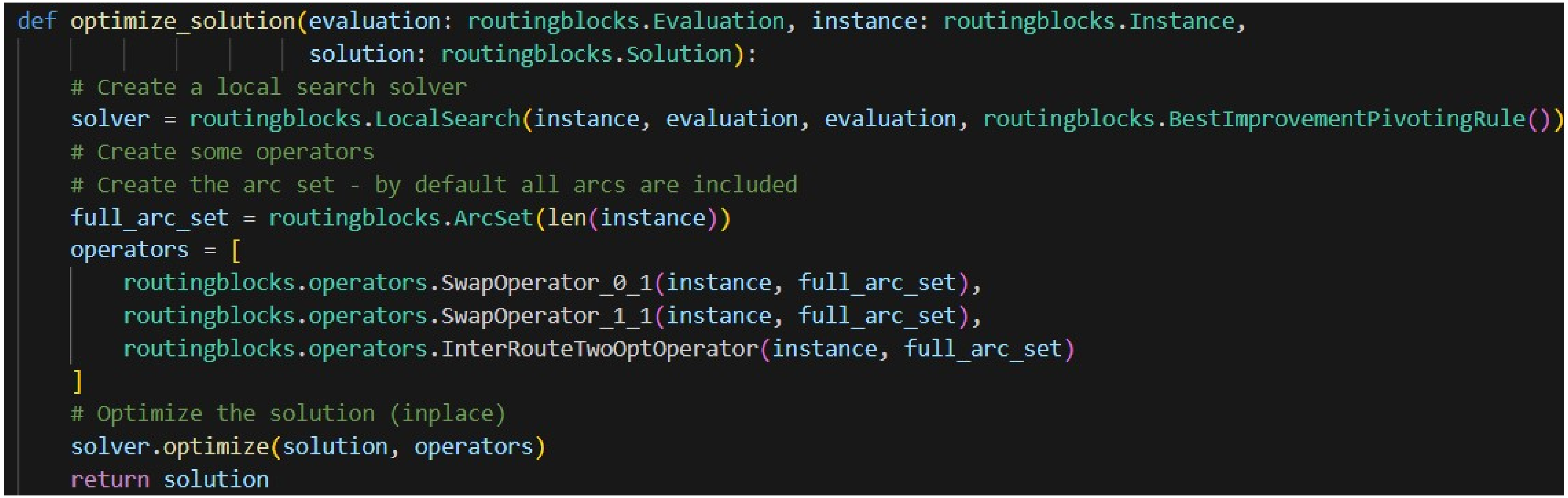
```python
def create_cvrp_instance(instance):
    # Create CVRP vertices
    n = len(instance['demand'])
    vertices = [
        cvrp.create_cvrp_vertex(i, str(i), False, i == instance['depot'][0], cvrp.CVRPVertexData(instance['demand'][i]))
        for i in range(n)]
    # Create CVRP arcs
    arcs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

- evaluation.cpp
  - allow us to define porblem, including: how to compute cost, which properties are vertex and edges included
  - routingblocks::concatenationBasedEvaluation
  - pybind11
    - m: let all class and module to bind the target module
    - CVRPEvaluation bind to routingblocks::Evaluation
    - bindings::helpers
      - provide us an interface to bind vertex and arc methods to routingblocks

How to use

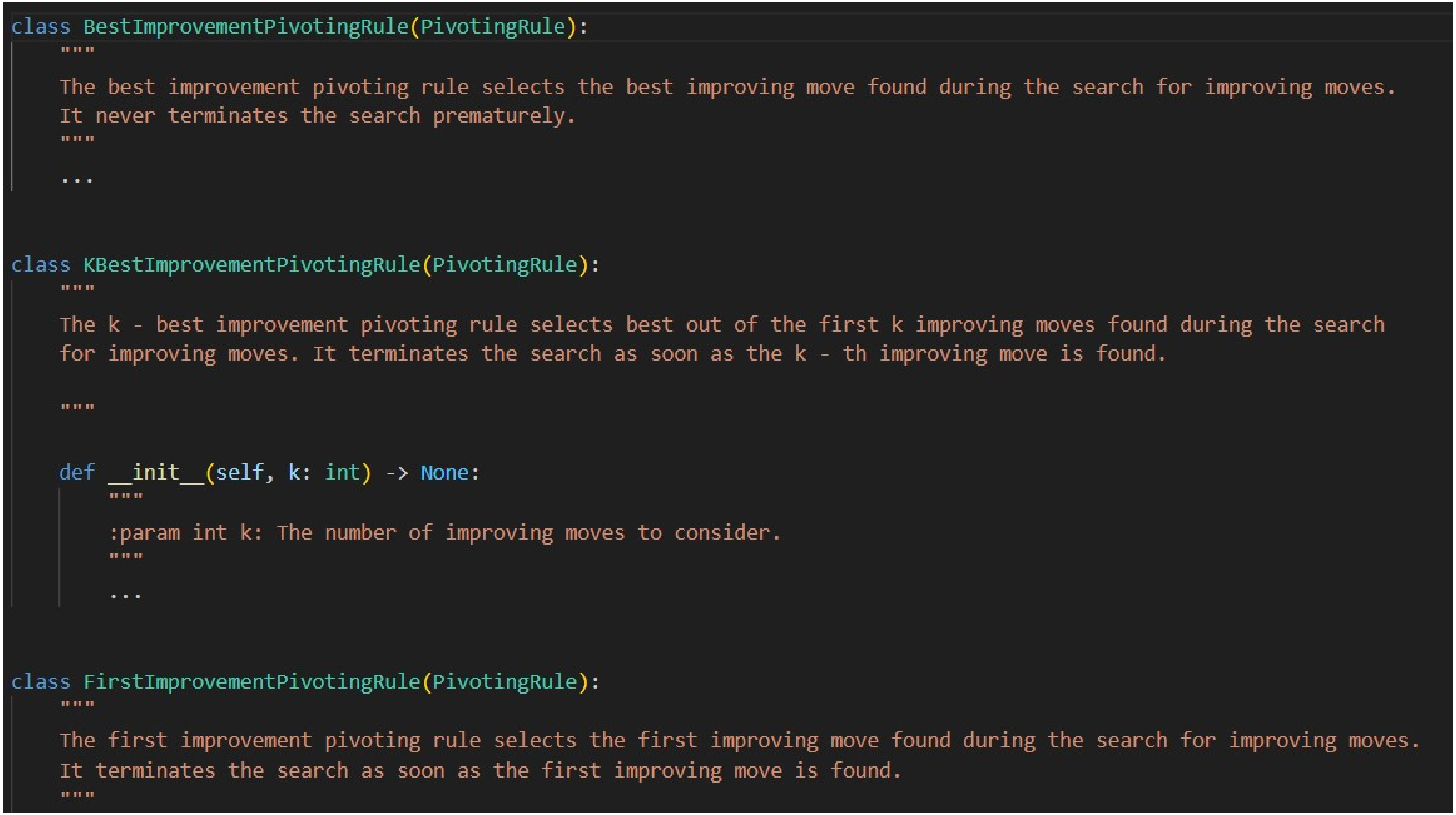
- create solver: routingblocks.LocalSearch(instance, evaluation, exact_evaluation, pivoting_rule)
  - /include/routingblocks/LocalSearch
  - exact_evaluation: for EVRP-TW, CVRP will keep it equal to evaluation
- solve: solver.optimize(solution, operators): search neighborhood



```
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

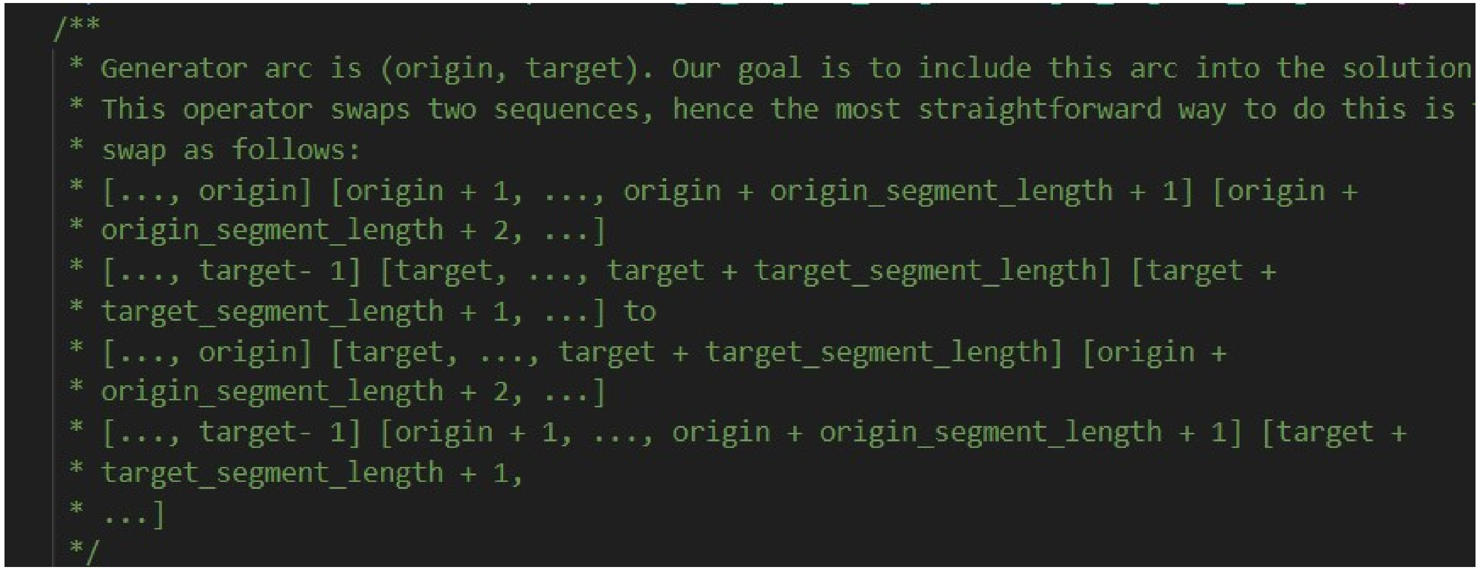
- pivoting_rule
  - /_routingblocks.pyi



```
class BestImprovementPivotingRule(PivotingRule):
    """
    The best improvement pivoting rule selects the best improving move found during the search for improving moves.
    It never terminates the search prematurely.
    """
    ...


class KBestImprovementPivotingRule(PivotingRule):
    """
    The k - best improvement pivoting rule selects best out of the first k improving moves found during the search
    for improving moves. It terminates the search as soon as the k - th improving move is found.

    """

    def __init__(self, k: int) -> None:
        """
        :param int k: The number of improving moves to consider.
        """
        ...


class FirstImprovementPivotingRule(PivotingRule):
    """
    The first improvement pivoting rule selects the first improving move found during the search for improving moves.
    It terminates the search as soon as the first improving move is found.
    """
```

How to use

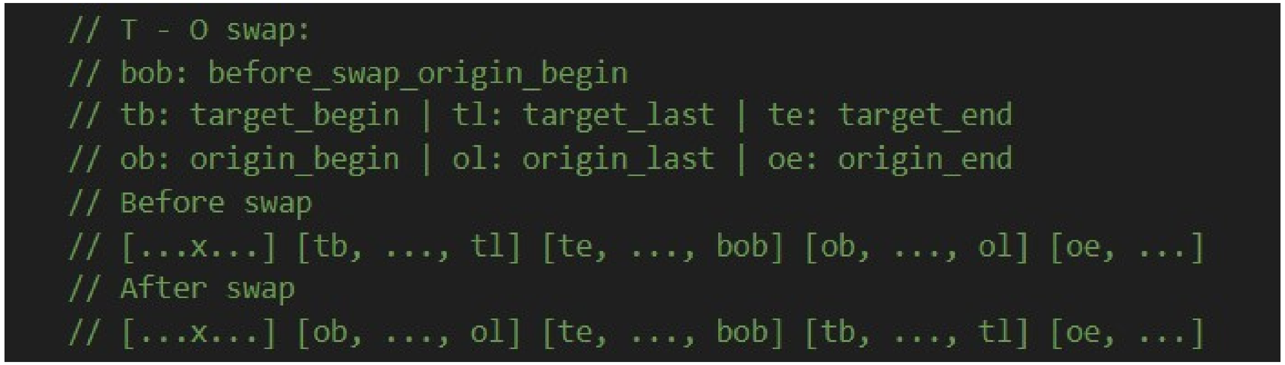
- operator
  - SwapOperator and InterRouteTwoOptOperator can be found in /include/routingblocks/operators
  - operator(instance, arc_set): how to find neighborhood
  - SwapOperator_[origin_segment_length]_[target_segment_length]



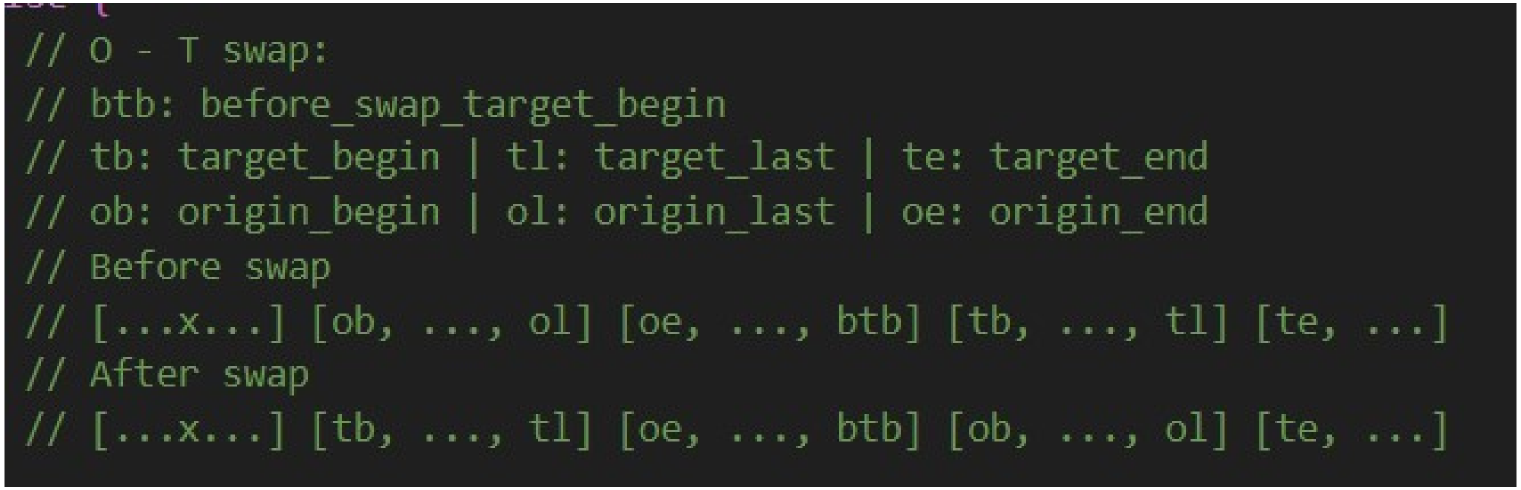
```
/**
 * Generator arc is (origin, target). Our goal is to include this arc into the solution
 * This operator swaps two sequences, hence the most straightforward way to do this is
 * swap as follows:
 * [..., origin] [origin + 1, ..., origin + origin_segment_length + 1] [origin +
 * origin_segment_length + 2, ...]
 * [..., target- 1] [target, ..., target + target_segment_length] [target +
 * target_segment_length + 1, ...] to
 * [..., origin] [target, ..., target + target_segment_length] [origin +
 * origin_segment_length + 2, ...]
 * [..., target- 1] [origin + 1, ..., origin + origin_segment_length + 1] [target +
 * target_segment_length + 1,
 * ...]
 */
```

  - InterRouteTwoOptOperator

How to use



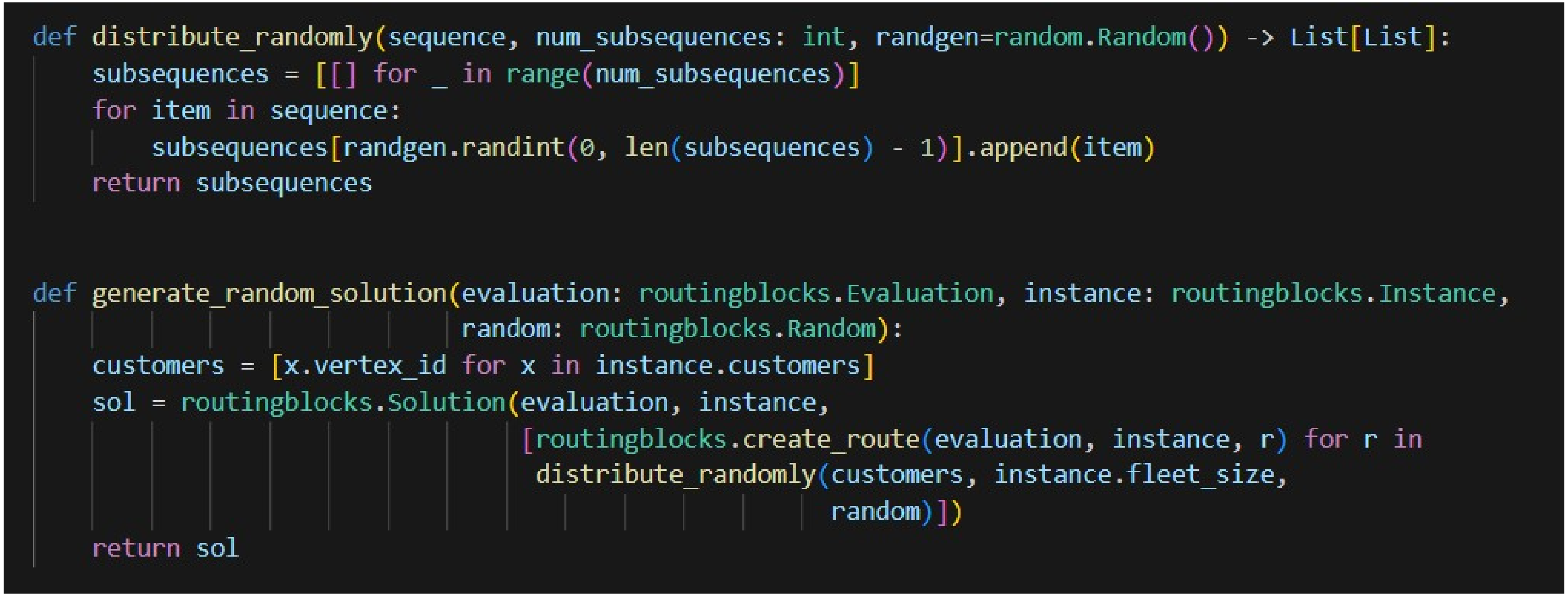
```
// T - O swap:
// bob: before_swap_origin_begin
// tb: target_begin | tl: target_last | te: target_end
// ob: origin_begin | ol: origin_last | oe: origin_end
// Before swap
// [...x...] [tb, ..., tl] [te, ..., bob] [ob, ..., ol] [oe, ...]
// After swap
// [...x...] [ob, ..., ol] [te, ..., bob] [tb, ..., tl] [oe, ...]
```



```
// O - T swap:
// btb: before_swap_target_begin
// tb: target_begin | tl: target_last | te: target_end
// ob: origin_begin | ol: origin_last | oe: origin_end
// Before swap
// [...x...] [ob, ..., ol] [oe, ..., btb] [tb, ..., tl] [te, ...]
// After swap
// [...x...] [tb, ..., tl] [oe, ..., btb] [ob, ..., ol] [te, ...]
```

How to use

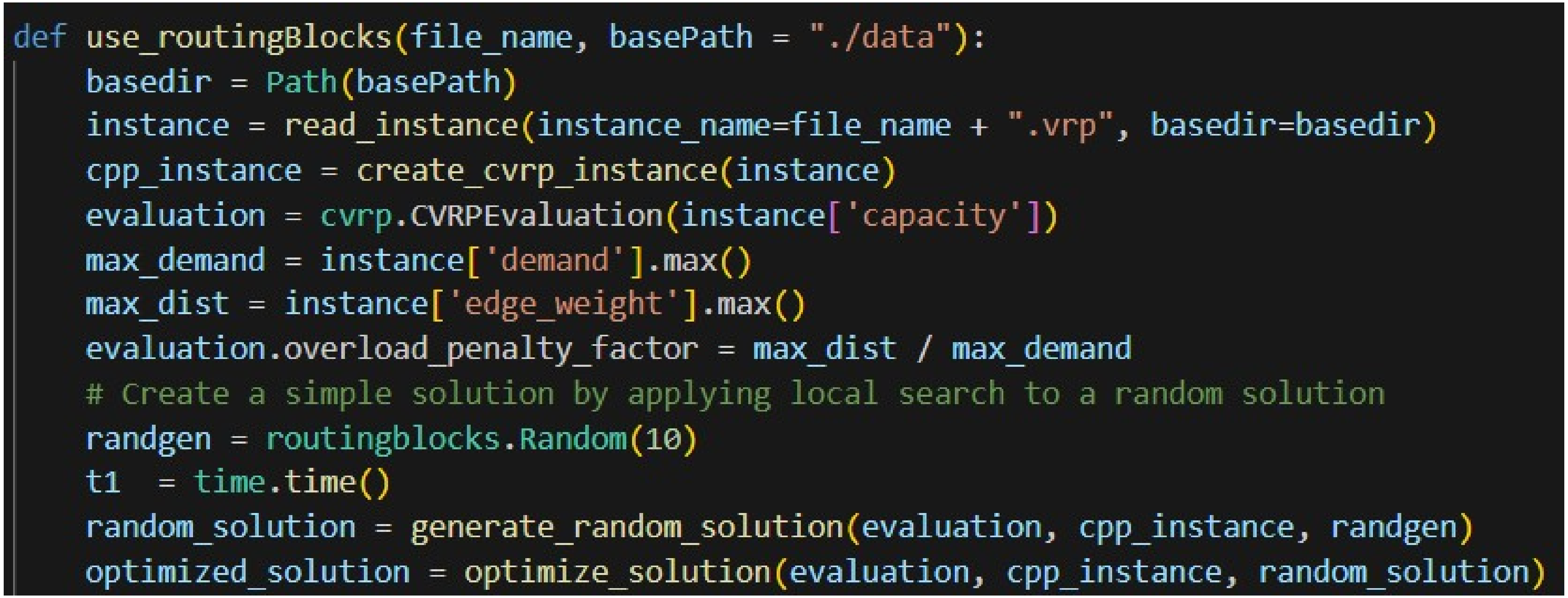
- generate random solution and optimize it



```python
def distribute_randomly(sequence, num_subsequences: int, randgen=random.Random()) -> List[List]:
    subsequences = [[] for _ in range(num_subsequences)]
    for item in sequence:
        subsequences[randgen.randint(0, len(subsequences) - 1)].append(item)
    return subsequences


def generate_random_solution(evaluation: routingblocks.Evaluation, instance: routingblocks.Instance,
                             random: routingblocks.Random):
    customers = [x.vertex_id for x in instance.customers]
    sol = routingblocks.Solution(evaluation, instance,
                                 [routingblocks.create_route(evaluation, instance, r) for r in
                                  distribute_randomly(customers, instance.fleet_size,
                                                      random)])
    return sol
```

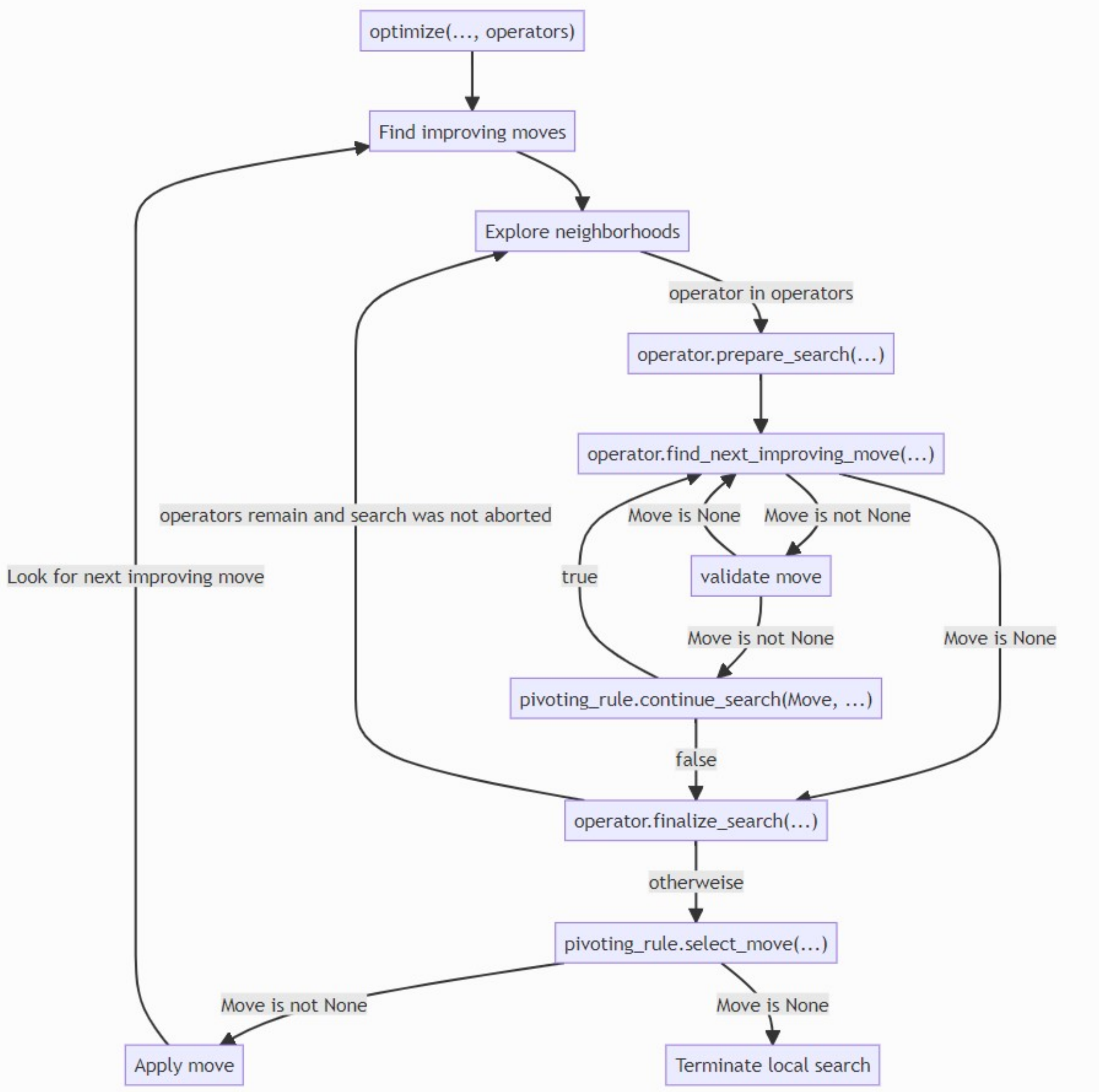
How to use

- generate random solution and optimize it



```python
def use_routingBlocks(file_name, basePath = "./data"):
    basedir = Path(basePath)
    instance = read_instance(instance_name=file_name + ".vrp", basedir=basedir)
    cpp_instance = create_cvrp_instance(instance)
    evaluation = cvrp.CVRPEvaluation(instance['capacity'])
    max_demand = instance['demand'].max()
    max_dist = instance['edge_weight'].max()
    evaluation.overload_penalty_factor = max_dist / max_demand
    # Create a simple solution by applying local search to a random solution
    randgen = routingblocks.Random(10)
    t1  = time.time()
    random_solution = generate_random_solution(evaluation, cpp_instance, randgen)
    optimized_solution = optimize_solution(evaluation, cpp_instance, random_solution)
```

How to use


optimize(..., operators)
Find improving moves
Explore neighborhoods
operator in operators
operator.prepare_search(...)
operator.find_next_improving_move(...)
operators remain and search was not aborted
Move is None
Move is not None
Look for next improving move
true
validate move
Move is not None
Move is None
pivoting_rule.continue_search(Move, ...)
false
operator.finalize_search(...)
otherweise
pivoting_rule.select_move(...)
Move is not None
Move is None
Apply move
Terminate local search

Control flow of the local search solver

Performance

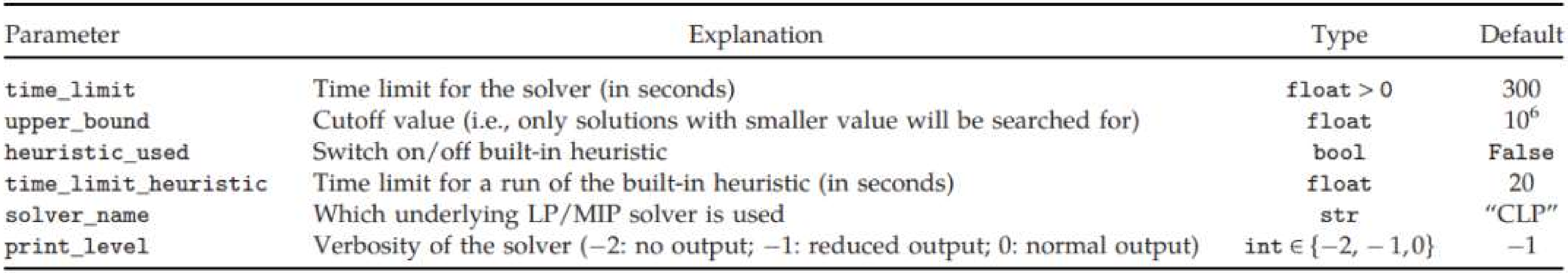
- Get solution in 5 minutes: n491-k59
- Found a solution with cost: 70853.5390625.
- This is 2.4% worse than the best known solution, which is 69226.
- Time: 259 seconds (4m 19s)

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

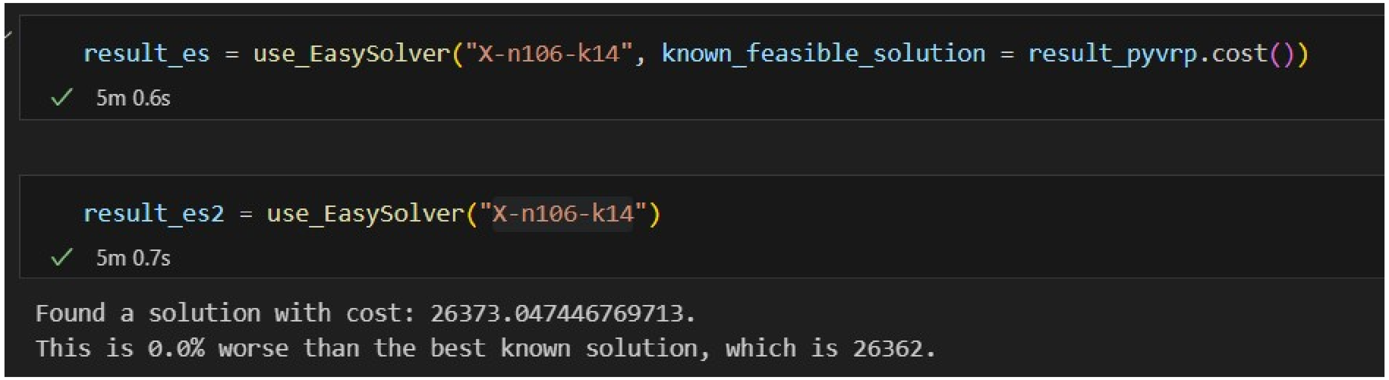
- create model
  - model = VRPSolverEasy.Model()
- set parameters: model.set_parameters()
  - time_limit: easysolver is solwer so we have to set a high value when faced with large data (client >100)



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

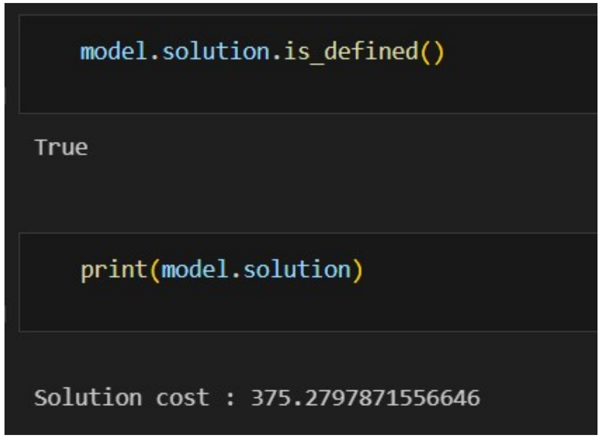
How to use

- 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 vechicles:
  - `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)
```

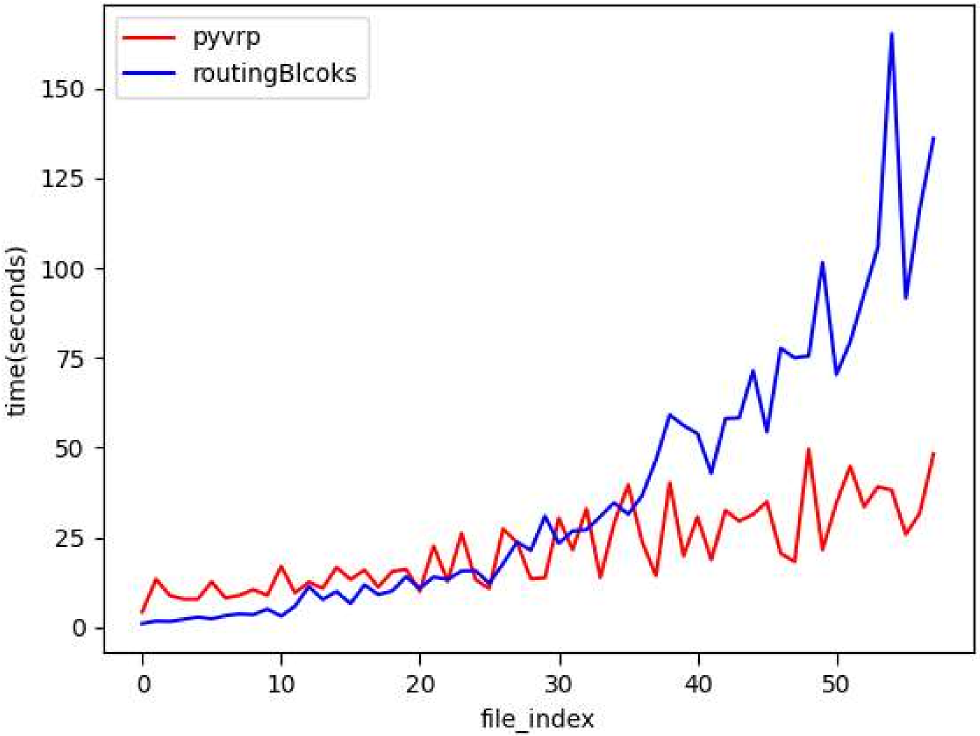
```
Solution cost : 375.2797871556646
```

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)

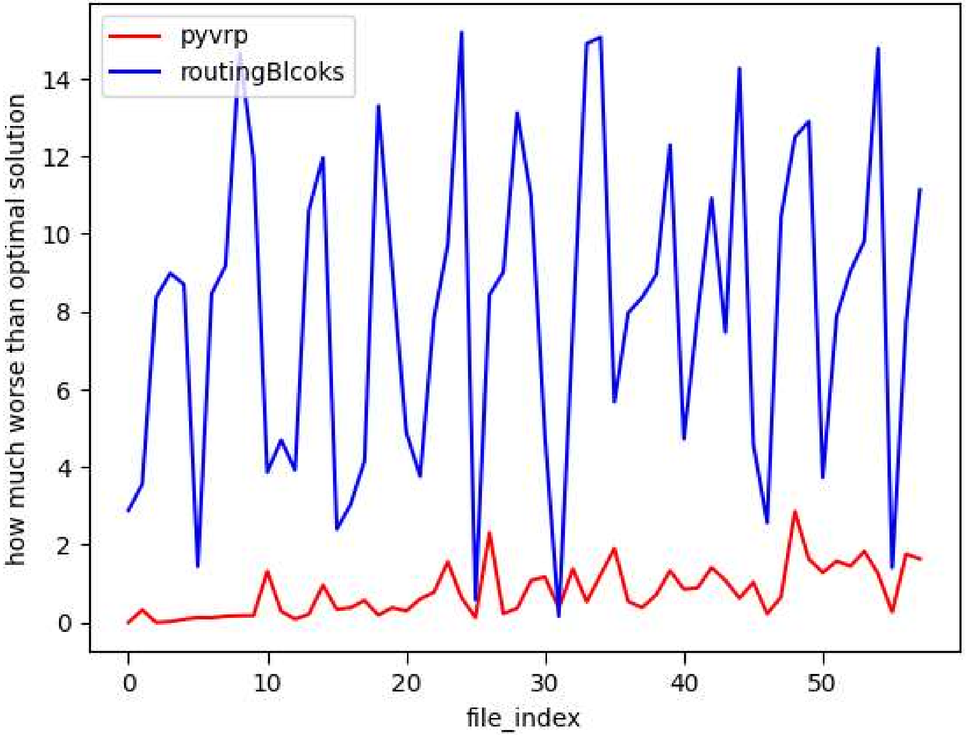
Performance comparison

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

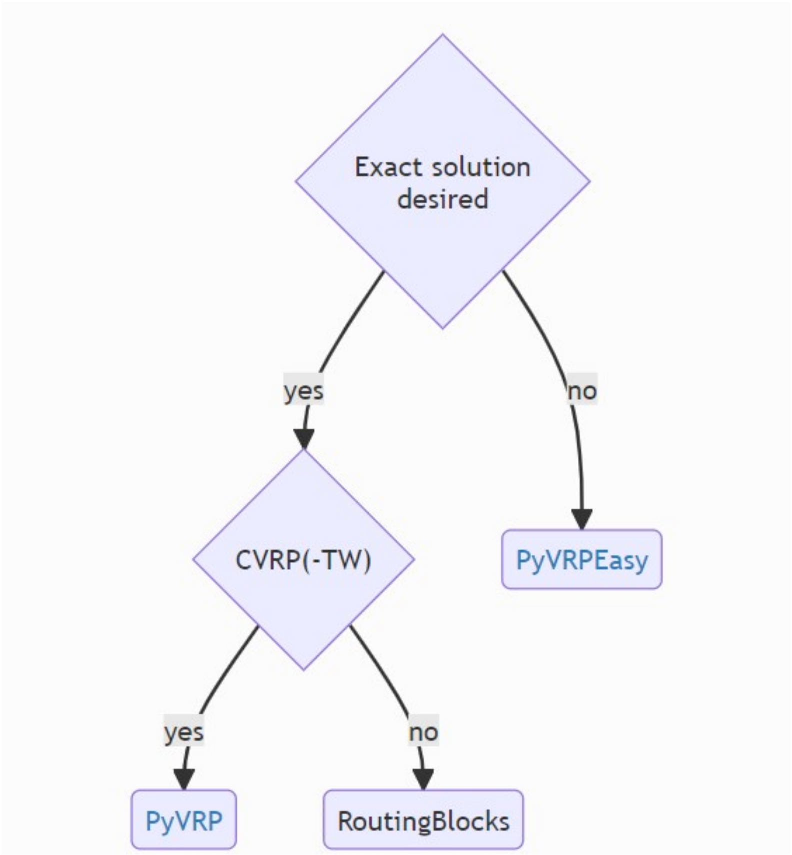


Performance comparison

- Accuracy: pyvrp much better than rb



Guide from RoutingBlocks



- You could find all my files in:
    - https://github.com/DOHZH/IE597Spring2024

- Thank you!