1 World module 3
2 Ant module 5
3 Solver module 9
4 Indices and tables 13
Python Module Index 15
A Python3 implementation of the Ant Colony Optimization Meta-Heuristic.

**Pants** provides you with the ability to quickly determine how to visit a collection of interconnected nodes such that the work done is minimized. Nodes can be any arbitrary collection of data while the edges represent the amount of “work” required to travel between two nodes. Thus, **Pants** is a tool for solving traveling salesman problems.

The world is built from a list of nodes and a function responsible for returning the length of the edge between any two given nodes. The length function need not return actual length. Instead, “length” refers to that the amount of “work” involved in moving from the first node to the second node - whatever that “work” may be. For a silly, random example, it could even be the number of dishes one must wash before moving to the next station at a least dish-washing dish washer competition.

Solutions are found through an iterative process. In each iteration, several ants are allowed to find a solution that “visits” every node of the world. The amount of pheromone on each edge is updated according to the length of the solutions in which it was used. The ant that traveled the least distance is considered to be the local best solution. If the local solution has a shorter distance than the best from any previous iteration, it then becomes the global best solution. The elite ant(s) then deposit their pheromone along the path of the global best solution to strengthen it further, and the process repeats.

You can read more about **Ant Colony Optimization on Wikipedia.**
**World module**

**class** pants.world.Edge (start, end, length=None, pheromone=None)

This class represents the link between starting and ending nodes.

In addition to start and end nodes, every Edge has length and pheromone properties. length represents the static, a priori information, whereas pheromone level represents the dynamic, a posteriori information.

**Parameters**

- **start** (node) – the node at the start of the Edge
- **end** (node) – the node at the end of the Edge
- **length** (float) – the length of the Edge (default=1)
- **pheromone** (float) – the amount of pheromone on the Edge (default=0.1)

**class** pants.world.World (nodes, lfunc, **kwargs)

The nodes and edges of a particular problem.

Each World is created from a list of nodes, a length function, and optionally, a name and a description. Additionally, each World has a UID. The length function must accept nodes as its first two parameters, and is responsible for returning the distance between them. It is the responsibility of the create_edges() to generate the required Edges and initialize them with the correct length as returned by the length function.

Once created, World objects convert the actual nodes into node IDs, since solving does not rely on the actual data in the nodes. These are accessible via the nodes property. To access the actual nodes, simply pass an ID obtained from nodes to the data() method, which will return the node associated with the specified ID.

Edges are accessible in much the same way, except two node IDs must be passed to the data() method to indicate which nodes start and end the Edge. For example:

```python
ids = world.nodes
assert len(ids) > 1
node0 = world.data(ids[0])
nodel = world.data(ids[1])
edge01 = world.data(ids[0], ids[1])
assert edge01.start == node0
assert edge01.end == nodel
```

The reset_pheromone() method provides an easy way to reset the pheromone levels of every Edge contained in a World to a given level. It should be invoked before attempting to solve a World unless a “blank slate” is not desired. Also note that it should not be called between iterations of the Solver because it effectively erases the memory of the Ant colony solving it.

**Parameters**
- **nodes** (*list*) – a list of nodes
- **lfunc** (*callable*) – a function that calculates the distance between two nodes
- **name** (*str*) – the name of the world (default is “world#”, where “#” is the uid of the world)
- **description** (*str*) – a description of the world (default is None)

`create_edges()`
Create edges from the nodes.

The job of this method is to map node ID pairs to `Edge` instances that describe the edge between the nodes at the given indices. Note that all of the `Edges` are created within this method.

**Returns** a mapping of node ID pairs to `Edge` instances.

**Return type** `dict`

`data(idx, idy=None)`
Return the node data of a single id or the edge data of two ids.

If only `idx` is specified, return the node with the ID `idx`. If `idy` is also specified, return the `Edge` between nodes with indices `idx` and `idy`.

**Parameters**
- **idx** (*int*) – the id of the first node
- **idy** (*int*) – the id of the second node (default is None)

**Returns** the node with ID `idx` or the `Edge` between nodes with IDs `idx` and `idy`.

**Return type** node or `Edge`

`nodes`
Node IDs.

`reset_pheromone(level=0.01)`
Reset the amount of pheromone on every edge to some base `level`.

Each time a new set of solutions is to be found, the amount of pheromone on every edge should be equalized to ensure un-biased initial conditions.

**Parameters** `level` (*float*) – amount of pheromone to set on each edge (default=0.01)
class pants.ant.Ant (alpha=1, beta=3)
A single independent finder of solutions to a World.

Each Ant finds a solution to a world one move at a time. They also represent the solution they find, and are capable of reporting which nodes and edges they visited, in what order they were visited, and the total length of the solution.

Two properties govern the decisions each Ant makes while finding a solution: alpha and beta. alpha controls the importance placed on pheromone while beta controls the importance placed on distance. In general, beta should be greater than alpha for best results. Ants also have a uid property that can be used to identify a particular instance.

Using the initialize() method, each Ant must be initialized to a particular World, and optionally may be given an initial node from which to start finding a solution. If a starting node is not given, one is chosen at random. Thus a few examples of instantiation and initialization might look like:

ant = Ant()
ant.initialize(world)

ant = Ant().initialize(world)

ant = Ant(alpha=0.5, beta=2.25)
ant.initialize(world, start=world.nodes[0])

Note: The examples above assume the world has already been created!

Once an Ant has found a solution (or at any time), the solution may be obtained and inspected by accessing its tour property, which returns the nodes visited in order, or it's path property, which returns the edges visited in order. Also, the total distance of the solution can be accessed through its distance property. Ants are even sortable by their distance:

ants = [Ant() for ...]
# ... have each ant in the list solve a world
ants = sorted(ants)
for i in range(1, len(ants)):
    assert ants[i - 1].distance < ants[i].distance

Ants may be cloned, which will return a shallow copy while not preserving the uid property. If this behavior is not desired, simply use the copy.copy() or copy.deepcopy() methods as necessary.

The remaining methods mainly govern the mechanics of making each move. can_move() determines whether all possible moves have been made, remaining_moves() returns the moves not yet made, choose_move() returns a single move from a list of moves, make_move() actually performs the move,
and `weight()` returns the weight of a given move. The `move()` method governs the move-making process by gathering the remaining moves, choosing one of them, making the chosen move, and returning the move that was made.

```python
can_move()
    Return True if there are moves that have not yet been made.
    
    Return type  bool

choose_move(choices)
    Choose a move from all possible moves.
    
    Parameters choices (list) – a list of all possible moves
    
    Returns  the chosen element from choices
    
    Return type  node

cloned()  
    Return a shallow copy with a new UID.
    
    If an exact copy (including the uid) is desired, use the `copy.copy()` method.
    
    Returns  a clone
    
    Return type  Ant

initialize(world, start=None)
    Reset everything so that a new solution can be found.
    
    Parameters
    
        • world (World) – the world to solve
        • start (Node) – the starting node (default is chosen randomly)
    
    Returns  self
    
    Return type  Ant

make_move(dest)
    Move to the dest node and return the edge traveled.
    
    When `dest` is `None`, an attempt to take the final move back to the starting node is made. If that is not possible (because it has previously been done), then `None` is returned.
    
    Parameters  dest (node) – the destination node for the move
    
    Returns  the edge taken to get to dest
    
    Return type  Edge

move()
    Choose, make, and return a move from the remaining moves.
    
    Returns  the Edge taken to make the move chosen
    
    Return type  Edge

node
    Most recently visited node.

path
    Edges traveled by the Ant in order.

remaining_moves()
    Return the moves that remain to be made.
Return type  list

tour
Nodes visited by the Ant in order.

weigh (edge)
Calculate the weight of the given edge.

The weight of an edge is simply a representation of its perceived value in finding a shorter solution. Larger weights increase the odds of the edge being taken, whereas smaller weights decrease those odds.

Parameters  edge (Edge) – the edge to weigh

Returns  the weight of edge

Return type  float
class pants.solver.Solver(**kwargs)
    This class contains the functionality for finding one or more solutions for a given World.

    Parameters
    • alpha (float) – relative importance of pheromone (default=1)
    • beta (float) – relative importance of distance (default=3)
    • rho (float) – percent evaporation of pheromone (0..1, default=0.8)
    • q (float) – total pheromone deposited by each Ant after each iteration is complete (>0, default=1)
    • t0 (float) – initial pheromone level along each Edge of the World (>0, default=0.01)
    • limit (int) – number of iterations to perform (default=100)
    • ant_count (float) – how many Ants will be used (default=10)
    • elite (float) – multiplier of the pheromone deposited by the elite Ant (default=0.5)

aco(colony)
    Return the best solution by performing the ACO meta-heuristic.
    
    This method lets every Ant in the colony find a solution, updates the pheromone levels according to the
    solutions found, and returns the Ant with the best solution.
    
    This method is not meant to be called directly. Instead, call either solve() or solutions().

    Parameters colony (list) – the Ants to use in finding a solution
    
    Returns the best solution found
    
    Return type Ant

create_colony(world)
    Create a set of Ants and initialize them to the given world.
    
    If the ant_count is less than 1, round_robin_ants() are used and the number of Ants will be equal to
    the number of nodes. Otherwise, random_ants() are created instead, and the number of Ants will be
equal to the ant_count.

    Parameters world (World) – the world from which the Ants will be given starting nodes.
    
    Returns list of Ants
    
    Return type list
find_solutions (ants)
Let each Ant find a solution.

Makes each Ant move until each can no longer move.

Parameters ants (list) – the ants to use for solving

global_update (ants)
Update the amount of pheromone on each edge according to the fitness of solutions that use it.
This accomplishes the global update performed at the end of each solving iteration.

Note: This method should never let the pheromone on an edge decrease to less than its initial level.

Parameters ants (list) – the ants to use for solving

local_update (edge)
Evaporate some of the pheromone on the given edge.

Note: This method should never let the pheromone on an edge decrease to less than its initial level.

Parameters edge (Edge) – the Edge to be updated

random_ants (world, count, even=False)
Returns a list of Ants distributed to the nodes of the world in a random fashion.

Note that this does not ensure at least one Ant begins at each node unless there are exactly as many Ants as there are nodes. This method is used to create the Ants before solving if ant_count is not 0.

Parameters
• world (World) – the World in which to create the ants.
• count (int) – the number of Ants to create
• even (bool) – True if random.random() should avoid choosing the same starting node multiple times (default is False)

Returns the Ants initialized to nodes in the World

Return type list

reset_colony (colony)
Reset the colony of Ants such that each Ant is ready to find a new solution.

Essentially, this method re-initializes all Ants in the colony to the World that they were initialized to last. Internally, this method is called after each iteration of the Solver.

Parameters colony (list) – the Ants to reset

round_robin_ants (world, count)
Returns a list of Ants distributed to the nodes of the world in a round-robin fashion.

Note that this does not ensure at least one Ant begins at each node unless there are exactly as many Ants as there are nodes. However, if ant_count is 0 then ant_count is set to the number of nodes in the World and this method is used to create the Ants before solving.

Parameters
• world (World) – the World in which to create the Ants
• count (int) – the number of Ants to create
Returns the Ants initialized to nodes in the World

Return type list

`solutions(world)`
Return successively shorter paths through the given world.

Unlike `solve()`, this method returns one solution for each improvement of the best solution found thus far.

Parameters world (World) – the World to solve

Returns successively shorter solutions as Ants

Return type list

`solve(world)`
Return the single shortest path found through the given world.

Parameters world (World) – the World to solve

Returns the single best solution found

Return type Ant

`trace_elite(ant)`
Deposit pheromone along the path of a particular ant.

This method is used to deposit the pheromone of the elite Ant at the end of each iteration.

Note: This method should never let the pheromone on an edge decrease to less than its initial level.

Parameters ant (Ant) – the elite Ant
Indices and tables

- genindex
- modindex
- search
Python Module Index

a
ant *(Linux, Unix, Windows)*, 5

p
pants, 3
pants.ant, 5
pants.solver, 9
pants.world, 3

s
solver *(Linux, Unix, Windows)*, 9

w
world *(Linux, Unix, Windows)*, 3