ACOpy Documentation

Release 0.6.4

Robert Grant

May 23, 2019

Contents:

This project implements the Ant Colony Optimization Meta-Heuristic. Solutions are found through an iterative process. In each iteration, several ants find a solution that visits every city by considering not just the distance involved but also the amount of pheromone along each edge. At the end of each iteration, the ants deposit pheromone along the edges of the solution they found in inverse proportion to the total distance. In this way, the ants remember which edges are useful and which are not.

CHAPTER 1

ACOpy

Ant Colony Optimization for the Traveling Salesman Problem.

- Free software: Apache Software License 2.0
- Documentation: https://acopy.readthedocs.io.

1.1 Features

- Uses NetworkX for graph representation
- Solver can be customized via plugins
- Has a utility for plotting information about the solving process
- CLI tool that supports reading graphs in a variety of formats (including tsplib95)
- Support for plotting iteration data using matplotlib and pandas

ACOpy was formerly called "Pants."

For now, only Python 3.6+ is supported. If there is demand I will add support for 3.4+.

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install ACOpy, run this command in your terminal:

```
$ pip install acopy
```

There are some optional extras available that enable the ability to plot iteration data.

```
$ pip install acopy[plot]
```

This is the preferred method to install ACOpy, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for ACOpy can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/rhgrant10/acopy
```

Or download the tarball:

```
$ curl  -OL https://github.com/rhgrant10/acopy/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

3.1 Quickstart

To use ACOpy in a project, you simply use it to create a `Solver` and a `Colony`:

```
>>> import acopy
>>> solver = acopy.Solver(rho=.03, q=1)
>>> colony = acopy.Colony(alpha=1, beta=3)
```

We can use the solver and the colony to solve any weighted networkx graph. Let's use `tsplib95.utils.load_problem()` to read a TSPLIB file into a `networkx.Graph`:

```
>>> import tsplib95
>>> problem = tsplib95.load_problem('bayg29.tsp')
>>> G = problem.get_graph()
```

Solving is easy. Let's do 100 iterations with a default number of ants:

```
>>> tour = solver.solve(G, colony, limit=100)
```

How good was the best tour found? Let's look:

```
>>> tour.cost
1719
```

You can list the solution tour in terms of the nodes or edges:

```
>>> tour.nodes
[19,
 25,
 7,
...
>>> tour.path
[(19, 25),
```

(continues on next page)

(continued from previous page)

```
(25, 7),
(7, 23),
...
```

3.2 Solver Plugins

Adding plugins to a solver can either change how the solver works or add additional functionality. Adding a plugin is easy. Let's add a plugin that times the solver:

```
>>> timer = acopy.plugins.TimerPlugin()
>>> solver.add_plugin(timer)
```

Now after we solve we can get the duration and average time per iteration:

```
>>> best = solver.solve(G, colony, limit=100)
>>> timer.duration
4.946878910064697
>>> timer.time_per_iter
0.049468789100646976
```

3.2.1 Writing New Plugins

Writing a new plugin is realtively easy. Simply subclass *acopy.solvers.SolverPlugin* and provide one of the following hooks:

on_start called before the first iteration

on_iteration called upon completion of each iteration

on_finish called after the last iteration

Each hook takes as its only argument an instance of *acopy.solvers.State* that contains information about the state of the solver.

For example, let's write a plugin that increases the number of ants each iteration.

```
class IncreasingAnts(acopy.solvers.SolverPlugin):

    def __init__(self, delta=1):
        super().__init__(delta=delta)
        self.delta = delta

    def on_iteration(self, state):
        ant = state.colony.get_ants(self.delta)
        state.ants.append(ant)
```

Note that you must pass the parameters you want to appear in the `repr()` to `super()` as keyword arguments:

```
>>> IncreasingAnts(2)
<IncreasingAnts(delta=2)>
```

3.2.2 Built-in Plugins

There are several plugins built into acopy. Below is a description of what they do.

Printout

Print information about the solver as it works.

EliteTracer

Let the best ant from each iteration deposit more pheromone.

You can control how much pheromone is deposited by specifying the `factor`. For example, to deposit an additional two times the amount of pheromone set the factor to 2:

```
>>> elite = acopy.plugins.EliteTracer(factor=2)
```

You can also think of this as how many additional times the best ant from each iteration deposits her pheromone.

Timer

Time the total duration of the solver as well as the average time per iteration.

Darwin

Apply variation to the alpha and beta values on each iteration.

You can control the sigma value for the guassian distribution used to choose the next values:

```
>>> darwin = acopy.plugins.Darwin(sigma=.25)
```

StatsRecorder

Record data about the solutions and pheromone levels on each iteration.

Specifically the plugin records the amount of pheromone on every edge as well as the min, max, and average pheromone levels. It records the best, worst, average, and global best solution found for each iteration. Lastly, it tracks the number of unique soltions found for the each iteration, for all iterations, and how many unique solutions were new.

Periodic action plugins

Perform some action periodically.

Set the number of iterations that constitute a period using the `period` paramter:

```
>>> periodic = acopy.plugins.PeriodicActionPlugin(period=100)
```

By itself, the periodic action plugin does nothing but instead is designed to be subclassed. Just provide a defintion for the `action` method:

```
>>> import time

>>> # plugin that periodically prints the current time
>>> class PrintTime(acopy.plugins.PeriodicActionPlugin):
...     def action(self, state):
...         print(time.time())
...
```

There are two built-in subclasses: `PeriodicReset` and `PheromoneFlip`.

PeriodicReset

Periodically reset the pheromone levels.

PheromoneFlip

Periodically invert the pheromone levels so that the best edges become the worst, and vice versa.

Early termination plugins

Terminate the solver prematurely.

Like the *PeriodicActionPlugin* this plugin does nothing by itself. You must subclass it and provide a defintion for `should_terminate`:

```
>>> import time
```

```
>>> # plugin that stops the solver if the time is a pallendrome
>>> class PallendromicTerminator(acopy.plugins.EarlyTerminationPlugin):
...     def should_terminate(self, state):
...         seconds = str(int(time.time()))
...         return list(seconds) == list(reversed(seconds))
...
```

There are two such plugins: `Threshold` and `TimeLimit`.

Threshold

Set a minimum threshold cost for the solver. If a solution is found that meets or dips below the threshold then the solver terminates early.

```
>>> threshold = acopy.plugins.Threshold(threshold=1719)
```

TimeLimit

Set a time limit for the solver.

The maximum number of seconds is of course configurable. The plugin will stop the solver from iterating again if the number of seconds exceeds the value set:

```
>>> time_limit = acopy.plugins.TimeLimit(seconds=30)
```

Note this means that it is possible to exceed the time limit since it will not stop the solver mid-iteration.

3.3 CLI Tool

The CLI tool included provides a quick way to solve graphs saved as files in a variety of formats.

```
$ acopy solve --file ~/Downloads/ALL_tsp/burma14.tsp --file-format tsplib95 --limit 50
SEED=172438059386129273
Solver(rho=0.03, q=1.0, top=None)
Registering plugin: <Printout()>
Registering plugin: <Timer()>
Registering plugin: <Darwin(sigma=3.0)>
Using 33 ants from Colony(alpha=1.0, beta=3.0)
Performing 50 iterations:
Iteration   Cost    Solution
        0   42      1 14 13 12 11  9 10  8  7  6  5  4  3  2
        2   38      1 13 11  9 10  2  8  7  6  5  4  3 12 14
        3   34      1 11  9 10  2  8  7  6  5  4  3 14 12 13
        4   33      1 11  9 10  2  8 13  7  6  5  4 12  3 14
       28   32      1 11  9 10 14  3  4 12  6  5  7 13  8  2
       29   31      1 11  9 10  2  8 13  7  5  6 12  4  3 14
Done
Total time: 0.2856738567352295 seconds
Avg iteration time: 0.00571347713470459 seconds
```

CHAPTER 4

API Documentation

4.1 acopy package

4.1.1 acopy.ant module

class acopy.ant.**Ant**(*alpha=1*, *beta=3*)

Bases: object

An ant.

Ants explore a graph, using alpha and beta to guide their decision making process when choosing which edge to travel next.

Parameters

- **alpha** (*float*) – how much pheromone matters
- **beta** (*float*) – how much distance matters

alpha

How much pheromone matters. Always kept greater than zero.

beta

How much distance matters. Always kept greater than zero.

choose_destination(*graph*, *current*, *unvisited*)

Return the next node.

Parameters

- **graph** (networkx.Graph) – the graph being solved
- **current** – starting node
- **unvisited** (*list*) – available nodes

Returns chosen edge

choose_node(*choices*, *scores*)
Return one of the choices.

Note that `scores[i]` corresponds to `choices[i]`.

Parameters

- **choices** (*list*) – the unvisited nodes
- **scores** (*list*) – the scores for the given choices

Returns one of the choices

get_scores(*graph*, *current*, *destinations*)
Return scores for the given destinations.

Parameters

- **graph** (networkx.Graph) – the graph being solved
- **current** – the node from which to score the destinations
- **destinations** (*list*) – available, unvisited nodes

Returns scores

Return type list

get_starting_node(*graph*)
Return a starting node for an ant.

Parameters **graph** (networkx.Graph) – the graph being solved

Returns node

get_unvisited_nodes(*graph*, *solution*)
Return the unvisited nodes.

Parameters

- **graph** (networkx.Graph) – the graph being solved
- **solution** (*Solution*) – in progress solution

Returns unvisited nodes

Return type list

initialize_solution(*graph*)
Return a newly initialized solution for the given graph.

Parameters **graph** (networkx.Graph) – the graph to solve

Returns intialized solution

Return type *Solution*

score_edge(*edge*)
Return the score for the given edge.

Parameters **edge** (*dict*) – the edge data

Returns score

Return type float

tour(*graph*)
Find a solution to the given graph.

Parameters **graph** (networkx.Graph) – the graph to solve

Returns one solution

Return type *Solution*

class acopy.ant.**Colony**(*alpha=1*, *beta=3*)
Bases: object

Colony of ants.

Effectively this is a source of *Ant* for a *Solver*.

Parameters

- **alpha** (*float*) – relative factor for edge weight
- **beta** (*float*) – relative factor for edge pheromone

get_ants(*count*)
Return the requested number of *Ant* s.

Parameters **count** (*int*) – number of ants to return

Return type list

4.1.2 acopy.solvers module

class acopy.solvers.**Solution**(*graph*, *start*, *ant=None*)
Bases: object

Tour for a graph.

Parameters

- **graph** (networkx.Graph) – a graph
- **start** – starting node
- **ant** (*Ant*) – ant responsible

add_node(*node*)
Reocrd a node as visited.

Parameters **node** – the node visited

close()
Close the tour so that the first and last nodes are the same.

get_easy_id(*sep=' '*, *monospace=True*)

get_id()
Return the ID of the solution.

The default implementation is just each of the nodes in visited order.

Returns solution ID

Return type tuple

trace(*q*, *rho=0*)
Deposit pheromone on the edges.

Note that by default no pheromone evaporates.

Parameters

- **q** (*float*) – the amount of pheromone
- **rho** (*float*) – the percentage of pheromone to evaporate

class acopy.solvers.**Solver**(*rho=0.03*, *q=1*, *top=None*, *plugins=None*)
Bases: object

ACO solver.

Solvers control the parameters related to phomone deposit and evaporation. If top is not specified, it defaults to the number of ants used to solve a graph.

Parameters

- **rho** (*float*) – percentage of pheromone that evaporates each iteration
- **q** (*float*) – amount of pheromone each ant can deposit
- **top** (*int*) – number of ants that deposit pheromone
- **plugins** (*list*) – zero or more solver plugins

add_plugin(*plugin*)
Add a single solver plugin.

If plugins have the same name, only the last one added is kept.

Parameters **plugin** (acopy.plugins.SolverPlugin) – the plugin to add

add_plugins(**plugins*)
Add one or more solver plugins.

find_solutions(*graph*, *ants*)
Return the solutions found for the given ants.

Parameters

- **graph** (networkx.Graph) – a graph
- **ants** (*list*) – the ants to use

Returns one solution per ant

Return type list

get_plugins()
Return the added plugins.

Returns plugins previously added

Return type list

global_update(*state*)
Perform a global pheromone update.

Parameters **state** (*State*) – solver state

optimize(*graph*, *colony*, *gen_size=None*, *limit=None*)
Find and return increasingly better solutions.

Parameters

- **graph** (networkx.Graph) – graph to solve
- **colony** (*Colony*) – colony from which to source each *Ant*
- **gen_size** (*int*) – number of *Ant* s to use (default is one per graph node)

- **limit** (*int*) – maximum number of iterations to perform (default is unlimited so it will run forever)

Returns better solutions as they are found

Return type iter

solve(**args*, ***kwargs*)
Find and return the best solution.

Accepts exactly the same paramters as the *optimize()* method.

Returns best solution found

Return type *Solution*

class acopy.solvers.**SolverPlugin**(***kwargs*)
Bases: object

Solver plugin.

Solver plugins can be added to any solver to customize its behavior. Plugins are initilized once when added, once before the first solver iteration, once after each solver iteration has completed, and once after all iterations have completed.

Implenting each hook is optional.

initialize(*solver*)
Perform actions when being added to a solver.

Though technically not required, this method should be probably be idempotent since the same plugin could be added to the same solver multiple times (perhaps even by mistake).

Parameters **solver** (*acopy.solvers.Solver*) – the solver to which the plugin is being added

name = 'plugin'
unique name

on_finish(*state*)
Perform actions once all iterations have completed.

Parameters **state** (*acopy.solvers.State*) – solver state

on_iteration(*state*)
Perform actions after each iteration.

Parameters **state** (*acopy.solvers.State*) – solver state

on_start(*state*)
Perform actions before the first iteration.

Parameters **state** (*acopy.solvers.State*) – solver state

class acopy.solvers.**State**(*graph*, *ants*, *limit*, *gen_size*, *colony*)
Bases: object

Solver state.

This class tracks the state of a solution in progress and is passed to each plugin hook. Specically it conatins:

Attribute	Description
graph	graph being solved
colony	colony that generated the ants
ants	ants being used to solve the graph
limit	maximum number of iterations
gen_size	number of ants being used
solutions	solutions found this iteration
best	best solution found this iteration
is_new_record	whether the best is a new record
record	best solution found so far
previous_record	previously best solution

Parameters

- **graph** (networkx.Graph) – a graph
- **ants** (*list*) – the ants being used
- **limit** (*int*) – maximum number of iterations
- **gen_size** (*int*) – number of ants to use
- **colony** (*Colony*) – source colony for the ants

best

4.2 acopy.utils package

4.2.1 acopy.utils.data module

acopy.utils.data.**get_demo_graph**()

acopy.utils.data.**get_formats**()

acopy.utils.data.**read_graph_data**(*path*, *format_*)

acopy.utils.data.**read_json**(*path*)

acopy.utils.data.**read_tsplib95**(*path*)

4.2.2 acopy.utils.general module

acopy.utils.general.**is_plot_enabled**()
Return true if plotting is enabled.

Plotting requires matplotlib and pandas to be installed.

Returns indication of whether plotting is enabled

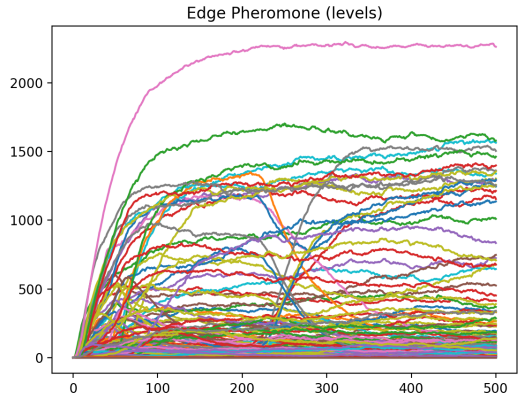
Return type bool

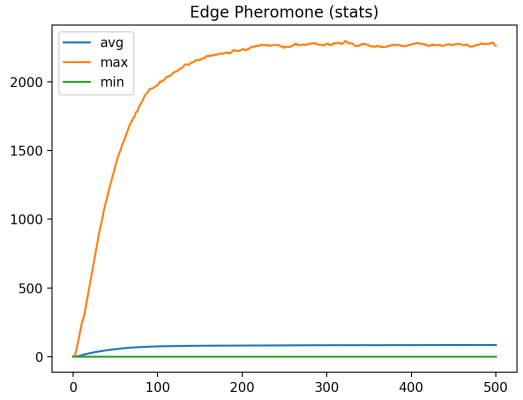
acopy.utils.general.**looper**(*limit*)
Return an optionally endless list of indexes.

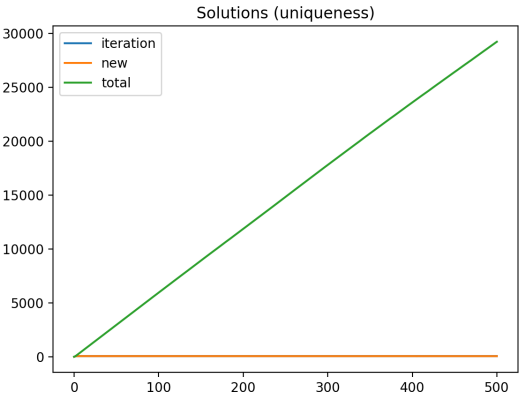
acopy.utils.general.**positive**(*value*)

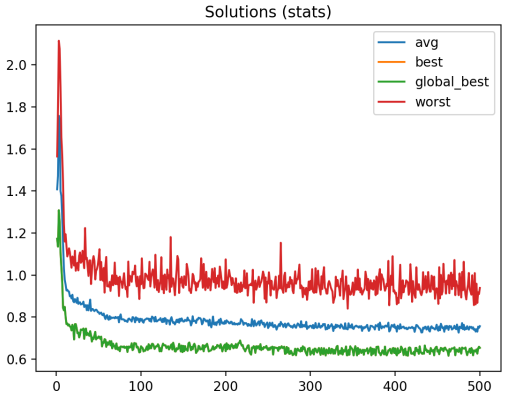
4.2.3 acopy.utils.plot module

The plot utility works with the StatsRecorder plugin to generate interesting plots of solver iterations.









class acopy.utils.plot.**Plotter**(*stats*)

Bases: object

Utility for plotting iteration data using matplotlib.

This is meant to be used in combination with the StatsRecorder plugin which collects stats about solutions and pheromone levels on each iteration.

Parameters **stats** (*dict*) – map of stats by name

extract_ant_distances()

plot()

Create and show the plot.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/rhgrant10/acopy/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

ACOpy could always use more documentation, whether as part of the official ACOpy docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/rhgrant10/acopy/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *acopy* for local development.

1. Fork the *acopy* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/acopy.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv acopy
$ cd acopy/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 acopy tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/rhgrant10/acopy/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_acopy
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Robert Grant <rhgrant10@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.6.4 (2019-05-17)

- Fix the missing *acopy.utils* package problem

7.2 0.6.3 (2019-05-17)

- Freshen up the dev dependencies
- Add the Python 3.7 classifier
- Actually fix import issue

7.3 0.6.2 (2019-02-02)

- Fix import issue

7.4 0.6.1 (2018-10-07)

- Bump dependency on tsplib95 to 0.3.2

7.5 0.6.0 (2018-08-18)

- First release on PyPI as `acopy`
- Complete rewrite
- Support for `networkx`

- Support for `tsplib95`
- Customizable solver
- Plotting capabilities
- Now uses Apache 2.0 License (formerly GPLv3)
- Supports only python 3.6+

7.6 0.5.2 (2014-09-09)

- Last release on the PyPI as `ACO-Pants`

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

T