rwa_wdm

Release 0.2.1

Cassio Batista

Nov 08, 2020

CONTENTS

RWA WDM: Routing and Wavelength Assignment Simulator over WDM Networks

This is a simulator for RWA algorithms over wavelength-multiplexed, all-optical networks with static traffic.

Contents:

CHAPTER

ONE

INTRODUCTION

`rwa_wdm` implements some mainstream algorithms for both routing and wavelength assignment subproblems as standalone packages, such as Dijkstra and Yen, and first-fit, random-fit, and vertex-coloring, respectively.

Besides, a self-made genetic algorithms is also implemented to solve the RWA problem.

1.1 Features

`rwa_wdm` is supports the following algorithms for RWA:

- Routing algorithms
  - Dijkstra
  - Yen
- Wavelength assignment algorithms
  - First-fit
  - Random-fit
  - Vertex-coloring
- RWA as one
  - Genetic algorithm with GOF

1.2 Installation

You can install `rwa_wdm` directly from Python Package Index via `pip`:

```
$ pip install rwa_wdm
```

Alternatively, you can build it from source:

```
$ git clone https://github.com/cassiobatista/rwa-wdm-sim
$ cd rwa-wdm-sim/
$ python setup.py install --skip-build
```

1.3 Example usage and output

`rwa_wdm` offers the possibility to be execute as a Python module via `-m` flag. To run Dijkstra's algorithm for routing and first-fit heuristics for wavelength assignment, just provide both `-r` and `-w` flags to `rwa_wdm` command line utility:

```
$ python -m rwa_wdm -r dijkstra -w first-fit -s 5
[2020-11-05 10:35:56] [__main__] INFO     Simulating 150 connection requests over nsf␣
↪topology with 8  per link using dijkstra + first-fit combination as RWA algorithm
Load:      1    2    3    4    5    6    7    8    9   10   11   12   13   14   15  ␣
↪16   17   18   19   20   21   22   23   24   25   26   27   28   29   30
Blocks: 0001 0000 0000 0004 0008 0022 0024 0034 0031 0057 0048 0055 0055 0057 0077␣
↪0098 0093 0082 0087 0092 0098 0097 0091 0111 0106 0128 0105 0118 0120 0115
BP (%):  0.7  0.0  0.0  2.7  5.3 14.7 16.0 22.7 20.7 38.0 32.0 36.7 36.7 38.0 51.3 65.
↪3 62.0 54.7 58.0 61.3 65.3 64.7 60.7 74.0 70.7 85.3 70.0 78.7 80.0 76.7 [sim 1: 7.
↪68 secs]
[2020-11-05 10:36:04] [io] INFO     Creating result dir in /tmp/rwa_results
[2020-11-05 10:36:04] [io] INFO     Writing blocking probability results to file "/
↪tmp/rwa_results/dijkstra_first-fit_8ch_150req_nsf.bp"
Blocks: 0000 0000 0002 0000 0010 0024 0021 0041 0039 0061 0050 0070 0060 0066 0079␣
↪0081 0085 0083 0108 0087 0089 0117 0103 0108 0105 0103 0095 0098 0114 0113
BP (%):  0.0  0.0  1.3  0.0  6.7 16.0 14.0 27.3 26.0 40.7 33.3 46.7 40.0 44.0 52.7 54.
↪0 56.7 55.3 72.0 58.0 59.3 78.0 68.7 72.0 70.0 68.7 63.3 65.3 76.0 75.3 [sim 2: 7.
↪54 secs]
[2020-11-05 10:36:11] [io] INFO     Writing blocking probability results to file "/
↪tmp/rwa_results/dijkstra_first-fit_8ch_150req_nsf.bp"
Blocks: 0000 0000 0001 0000 0001 0016 0024 0031 0054 0056 0049 0071 0074 0080 0066␣
↪0089 0086 0085 0095 0092 0089 0103 0101 0104 0107 0105 0106 0104 0115 0119
BP (%):  0.0  0.0  0.7  0.0  0.7 10.7 16.0 20.7 36.0 37.3 32.7 47.3 49.3 53.3 44.0 59.
↪3 57.3 56.7 63.3 61.3 59.3 68.7 67.3 69.3 71.3 70.0 70.7 69.3 76.7 79.3 [sim 3: 7.
↪58 secs]
[2020-11-05 10:36:19] [io] INFO     Writing blocking probability results to file "/
↪tmp/rwa_results/dijkstra_first-fit_8ch_150req_nsf.bp"
Blocks: 0000 0000 0000 0011 0015 0015 0010 0022 0063 0055 0075 0064 0072 0088 0071␣
↪0081 0083 0085 0109 0094 0088 0100 0104 0103 0107 0097 0108 0108 0100 0100
BP (%):  0.0  0.0  0.0  7.3 10.0 10.0  6.7 14.7 42.0 36.7 50.0 42.7 48.0 58.7 47.3 54.
↪0 55.3 56.7 72.7 62.7 58.7 66.7 69.3 68.7 71.3 64.7 72.0 72.0 66.7 66.7 [sim 4: 7.
↪56 secs]
[2020-11-05 10:36:26] [io] INFO     Writing blocking probability results to file "/
↪tmp/rwa_results/dijkstra_first-fit_8ch_150req_nsf.bp"
Blocks: 0001 0000 0000 0007 0016 0015 0021 0032 0048 0040 0061 0064 0071 0068 0070␣
↪0082 0076 0100 0101 0093 0106 0116 0103 0103 0109 0109 0100 0120 0121 0108
BP (%):  0.7  0.0  0.0  4.7 10.7 10.0 14.0 21.3 32.0 26.7 40.7 42.7 47.3 45.3 46.7 54.
↪7 50.7 66.7 67.3 62.0 70.7 77.3 68.7 68.7 72.7 72.7 66.7 80.0 80.7 72.0 [sim 5: 7.
↪59 secs]
```

(continues on next page)

(continued from previous page)

```
[2020-11-05 10:36:34] [io] INFO     Writing blocking probability results to file "/
↪tmp/rwa_results/dijkstra_first-fit_8ch_150req_nsf.bp"
[2020-11-05 10:37:13] [io] INFO     Writing simulation profiling times to file "/tmp/
↪rwa_results/dijkstra_first-fit_8ch_150req_nsf.it"
```

A quick help list is also available on the command line interface:

```
$ python -m rwa_wdm -h
usage: python -m rwa_wdm [-h] [-r <alg> -w <alg> | --rwa <alg>] [options]

RWA WDM Simulator: routing and wavelength assignment simulator for WDM networks

optional arguments:
  -h, --help                show this help message and exit

Network options:
  -t <topology>             network topology (default: nsf)
  -c <channels>             number of  per link (default: 8)

RWA algorithms options:
  -r <algorithm>            routing algorithm (default: None)
  -w <algorithm>            wavelength assignment algorithm (default: None)
  --rwa <algorithm>         routing *and* wavelength assigment algorithm (default:␣
↪None)
  -y <yen-alt-paths>        number of routing alternate paths (Yen's) (default: 2)

RWA simulator options:
  -l <max-load>             maximum network load, in Erlangs (default: 30)
  -k <conn-requests>        number of connection requests to arrive (default: 150)
  -d <result-dir>           dir to store blocking probability results (default: /tmp/
↪rwa_results)
  -s <num-simulations>      number of times to run the simulation (default: 1)
  -p                        plot blocking probability graph after simulation?␣
↪(default: False)

Genetic algorithm options:
  --pop-size POP_SIZE       number of individuals in the population (default: 25)
  --num-gen NUM_GEN         number of generations for population to evolve (default:␣
↪25)
  --cross-rate CROSS_RATE   crossover rate (default: 0.4)
  --mut-rate MUT_RATE       mutation rate (default: 0.02)
```

CHAPTER

TWO

RWA_WDM MODULES

2.1 Network

class `rwa_wdm.net.`**`Lightpath`**(*route: List[int]*, *wavelength: int*)

Emulates a lightpath composed by a route and a wavelength channel

Lightpath is pretty much a regular path, but must also specify a wavelength index, since WDM optical networks span multiple wavelength channels over a single fiber link on the topology.

A Lightpath object also store a holding time parameter, which is set along the simulation to specify how long the connection may be alive and running on network links, and therefore taking up space in the traffic matrix, before it finally terminates and resources are deallocated.

Parameters

- **`route`** – a liste of nodes encoded as integer indices
- **`wavelength`** – a single number representing the wavelength channel index

`__init__`(*route: List[int]*, *wavelength: int*)

Initialize self. See help(type(self)) for accurate signature.

`__str__`()

Return str(self).

`__weakref__`

list of weak references to the object (if defined)

property `holding_time`

Time that the lightpath remains occupying net resources

property `id`

A unique identifier to the Lightpath object

property links
Network links as a sequence of pairs of nodes

property r
The path as a sequence of router indices

property w
The wavelength channel index

class rwa_wdm.net.**Network**(*num_channels: int*, *num_nodes: int*, *num_links: int*)
Network base class

Hols network properties such as adjacency, wavelength-availability and traffic graph matrices, fixed source and destination nodes for all connections, number of channels per link

Parameters

- **num_channels** – number of wavelength channels per link
- **num_nodes** – number of routes along the path
- **num_links** – number of links along the path, typically *num_nodes* - 1

__init__(*num_channels: int*, *num_nodes: int*, *num_links: int*) → None
Initialize self. See help(type(self)) for accurate signature.

__weakref__
list of weak references to the object (if defined)

property a
The adjacency matrix graph

property d
The destination node

property n
The wavelength availability matrix graph

property name
The short name tag idenfier of the network topology

property nchannels
The number of wavelength channels per fiber link

property nlinks
The number of links (edges) in the network

property nnodes
The number of router nodes (vertices) in the network

plot_topology(*bestroute: List[int] = None*) → None
Plots the physical topology in a 2D Cartesian plan

Parameters **bestroute** – a route encoded as a list of router indices to be highlighted in red over some network edges

property s
The source node

property t
The traffic matrix

class rwa_wdm.net.**NationalScienceFoundation**(*ch_n*)
U.S. National Science Foundation Network (NSFNET)

__init__(*ch_n*)
Initialize self. See help(type(self)) for accurate signature.

get_edges() → List[Tuple[int, int]]
get

get_nodes_2D_pos() → Dict[str, Tuple[float, float]]
Get position of the nodes on the bidimensional Cartesian plan

class rwa_wdm.net.**CooperacionLatinoAmericana**(*ch_n*)
Cooperación Latino Americana de Redes Avanzadas (RedClara)

__init__(*ch_n*)
Initialize self. See help(type(self)) for accurate signature.

get_edges() → List[Tuple[int, int]]
get

get_nodes_2D_pos() → Dict[str, Tuple[float, float]]
Get position of the nodes on the bidimensional Cartesian plan

class rwa_wdm.net.**JointAcademicNetwork**(*ch_n*)
U.K. Joint Academic Network (JANET)

__init__(*ch_n*)
Initialize self. See help(type(self)) for accurate signature.

get_edges() → List[Tuple[int, int]]
get

get_nodes_2D_pos() → Dict[str, Tuple[float, float]]
Get position of the nodes on the bidimensional Cartesian plan

class rwa_wdm.net.**RedeNacionalPesquisa**(*ch_n*)
Rede (Brasileira) Nacional de Pesquisa (Rede Ipê / RNP)

__init__(*ch_n*)
Initialize self. See help(type(self)) for accurate signature.

get_edges() → List[Tuple[int, int]]
get

get_nodes_2D_pos() → Dict[str, Tuple[float, float]]
Get position of the nodes on the bidimensional Cartesian plan

2.2 RWA algorithms

rwa_wdm.rwa.rwa.**dijkstra_first_fit**(*net: rwa_wdm.net.net.Network*, *k: int*) → Optional[rwa_wdm.net.net.Lightpath]
Dijkstra and first-fit combination as RWA algorithm

Parameters

- **net** – Network topology instance
- **k** – number of alternate paths (ignored)

Returns

if successful, returns both route and wavelength index as a lightpath

Return type *Lightpath*

`rwa_wdm.rwa.rwa.dijkstra_vertex_coloring`(*net: rwa_wdm.net.net.Network*, *k: int*) → Optional[rwa_wdm.net.net.Lightpath]

Dijkstra and vertex coloring combination as RWA algorithm

Parameters

- **net** – Network topology instance
- **k** – number of alternate paths (ignored)

Returns

if successful, returns both route and wavelength index as a lightpath

Return type *Lightpath*

`rwa_wdm.rwa.rwa.genetic_algorithm`(*pop_size: int*, *num_gen: int*, *cross_rate: float*, *mut_rate: float*) → Callable

Genetic algorithm as both routing and wavelength assignment algorithm

This function just sets the parameters to the GA, so it acts as if it were a class constructor, setting a global variable as instance to the *GeneticAlgorithm* object in order to be further used by a callback function, which in turn returns the lightpath itself upon RWA success. This split into two classes is due to the fact that the class instance needs to be executed only once, while the callback may be called multiple times during simulation, namely one time per number of arriving call times number of load in Erlags (calls * loads)

Note: Maybe this entire script should be a class and *ga* instance could be an attribute. Not sure I'm a good programmer.

Parameters

- **pop_size** – number of chromosomes in the population
- **num_gen** – number of generations towards evolve
- **cross_rate** – percentage of individuals to perform crossover
- **mut_rate** – percentage of individuals to undergo mutation

Returns

a callback function that calls the *GeneticAlgorithm* runner class, which finally and properly performs the RWA procedure

Return type callable

`rwa_wdm.rwa.rwa.yen_first_fit`(*net: rwa_wdm.net.net.Network*, *k: int*) → Optional[rwa_wdm.net.net.Lightpath]

Yen and first-fit combination as RWA algorithm

Parameters

- **net** – Network topology instance
- **k** – number of alternate paths (ignored)

Returns

if successful, returns both route and wavelength index as a lightpath

Return type *Lightpath*

`rwa_wdm.rwa.rwa.yen_vertex_coloring`(*net: rwa_wdm.net.net.Network*, *k: int*) → Optional[rwa_wdm.net.net.Lightpath]

Yen and vertex coloring combination as RWA algorithm

Parameters

- **net** – Network topology instance
- **k** – number of alternate paths (ignored)

Returns

if successful, returns both route and wavelength index as a lightpath

Return type *Lightpath*

2.2.1 Standalone routing algorithms

Dijkstra shortest path algorithm as routing strategy

`rwa_wdm.rwa.routing.dijkstra.dijkstra`(*mat: numpy.ndarray*, *s: int*, *d: int*) → List[int]

Dijkstra routing algorithm

Parameters

- **mat** – Network's adjacency matrix graph
- **s** – source node index
- **d** – destination node index

Returns sequence of router indices encoding a path

Return type `list` of `int`

Yen's algorithm, a.k.a. k-shortest paths algorithm as routing strategy

`rwa_wdm.rwa.routing.yen.yen`(*mat: numpy.ndarray*, *s: int*, *d: int*, *k: int*) → List[List[int]]

Yen's routing algorithm, a.k.a. K-shortest paths

Parameters

- **mat** – Network's adjacency matrix graph
- **s** – source node index
- **d** – destination node index
- **k** – number of alternate paths

Returns a sequence of *k* paths

Return type `list` of `list`

2.2.2 Standalone wavelength assignment algorithms

First-fit wavelength assignment strategy

`rwa_wdm.rwa.wlassignment.ff.`**`first_fit`**(*net: rwa_wdm.net.net.Network*, *route: List[int]*) → Optional[int]

First-fit algorithm

Select the wavelength with the lowest index available at the first link of the path, starting of course from the source node.

Parameters

- **net** – Network object
- **route** – path encoded as a sequence of router indices

Returns

upon wavelength assignment success, return the wavelength index to be used on the lightpath

Return type `int`

Random-fit wavelength assignment strategy

`rwa_wdm.rwa.wlassignment.rf.`**`random_fit`**(*net: rwa_wdm.net.net.Network*, *route: List[int]*) → Optional[int]

Random-fit algorithm

Select a random wavelength index from the fixed set of available wavelengths

Parameters

- **net** – Network object
- **route** – path encoded as a sequence of router indices

Returns

upon wavelength assignment success, return the wavelength index to be used on the lightpath

Return type `int`

Vertex coloring wavelength assignment strategy

`rwa_wdm.rwa.wlassignment.vcolor.`**`vertex_coloring`**(*net: rwa_wdm.net.net.Network*, *lightpath: rwa_wdm.net.net.Lightpath*) → Optional[int]

Vertex coloring algorithm

Parameters

- **net** – Network object
- **lightpath** – the lightpath we are trying to allocate a to

Returns

upon wavelength assignment success, return the wavelength index to be used on the lightpath

Return type `int`

2.2.3 Genetic algorithm

class `rwa_wdm.rwa.ga.ga.`**`GeneticAlgorithm`**(*pop_size: int*, *num_gen: int*, *cross_rate: float*, *mut_rate: float*)

Genetic algorithm

Chromosomes are encoded as routes and fitness is based on a general objective funcion (GOF)'s labels to each wavelength index supported. Chromosome creation and mutation procedures are based on depth-first search (DFS) operation, crossover is based on the one-point strategy, and selection takes place under a k=3-size tournament rule.

`_population_size`
number of individuals that comprise a population

`_num_generations`
number of generations a population has to evolve

`_crossover_rate`
percentage of individuals to undergo crossover

`_mutation_rate`
percentage of individuals to undergo mutation

`_best_fits`
collection of best fitness values across generations

property `bestfit`
A lisf of the best fitness values across all generations

`run`(*net: rwa_wdm.net.net.Network*, *k: int*) → Tuple[List[int], Optional[int]]
Run the main genetic algorithm's evolution pipeline

Parameters

- **`net`** – Network instance object
- **`k`** – number of alternative paths (ignored)

Returns

obj: *tuple*: route as a list of router indices and wavelength index upon RWA success

class `rwa_wdm.rwa.ga.pop.`**`Population`**
Class to store a collection of Chromosome objects

Population is also responsible for sorting chromosomes by their fitness values, always keeping tracking of the best one.

`add_chromosome`(*chromosome:* rwa_wdm.rwa.ga.chromo.Chromosome) → None
Adds a Chromosome into the population

Parameters **`chromosome`** – an individual encoded as a Chromosome instance

property `best`
The fittest chromosome (requires sorting)

`copy`() → *rwa_wdm.rwa.ga.pop.Population*
Deep copy of the Population's own instance

property `individuals`
The population as a sequence of Chromosomes

`make_chromosome`(*mat: numpy.ndarray*, *s: int*, *d: int*, *allels: Set[int]*, *max_size: int*) → Union[None, *rwa_wdm.rwa.ga.chromo.Chromosome*]
Creates a single Chromosome via DFS-like procedure

Parameters

- **mat** – Network's wavelength availability matrix
- **s** – source node of the connection
- **d** – destination node of the connection
- **allels** – values the chromosome's genes are allowed to assume, which basically comprises router indices
- **max_size** – value to prevent chromosomes from being too long

Returns

returns an individual if random procedure is successfull

Return type *Chromosome*

remove_chromosome_by_id(*_id: int*) → None

Removes a Chromosome from the population

Parameters **_id** – unique index identifying a particular individual

sort() → int

Sorts the population following some criteria

Returns

obj: *int*: number of fit individuals, i.e., with at least one available on each link

class rwa_wdm.rwa.ga.chromo.**Chromosome**(*genes: List[int], fitness:* rwa_wdm.rwa.ga.chromo.Fitness = *None*)

Encodes an invididual with genes and a fitness attribute

Parameters

- **genes** – sequence of router indices comprising a route
- **fitness** – a Fitness object comprising GOF labels, number of available, and number of hops

property fit

The Fitness object

property genes

The route encoded as chromosome's genes

property id

A unique identifier to the Chromosome object

class rwa_wdm.rwa.ga.chromo.**Fitness**(*labels: numpy.ndarray*, *lambdas: int*, *hops: int*)

Fitness 'namedtuple'-like object

Easy to handle ready-to-use properties such as number of wavelengths available per route, and number of hops in the route.

Parameters

- **labels** – general objective function (GOF)'s label *L*
- **lambdas** – number of wavelengths available on a single link
- **hops** – number of hops in the route

property hops

The number of hops comprising a route

property labels
The labels L produced by the general objective function (GOF)

property lambdas
The total number of available on a single link

Environment related procedures

`rwa_wdm.rwa.ga.env.cross`(*parents:* rwa_wdm.rwa.ga.pop.Population, *pop_size: int*, *tc: float*) → *rwa_wdm.rwa.ga.pop.Population*

One-point crossover strategy

Parameters

- **parents** – set of chromosomes ready to mate
- **pop_size** – number of individuals in the offspring after crossover
- **tc** – crossover rate, which defines the percentage of the selected individuals to undergo crossover

Returns set of children in offspring to undergo mutation operation

Return type *Population*

`rwa_wdm.rwa.ga.env.evaluate`(*net: rwa_wdm.net.net.Network*, *chromosome:* rwa_wdm.rwa.ga.chromo.Chromosome) → *rwa_wdm.rwa.ga.chromo.Fitness*

Fitness calculation

Parameters

- **net** – Network instance object
- **chromosome** – Chromosome object

Returns

Fitness object storing GOF labels, number of available per link, and number of hops in the route

Return type *Fitness*

`rwa_wdm.rwa.ga.env.mutate`(*children:* rwa_wdm.rwa.ga.pop.Population, *pop_size: int*, *tm: float*, *net: rwa_wdm.net.net.Network*) → *rwa_wdm.rwa.ga.pop.Population*

Custom mutation procedure based on DFS-like path creation

Parameters

- **children** – Chromosome offspring after crossover
- **pop_size** – number of individuals to compose the new population
- **tm** – mutation rate, which defines the percentage of individuals to undergo mutation
- **net** – Network instance

Returns set of chromosomes to compose the new population

Return type *Population*

`rwa_wdm.rwa.ga.env.select`(*population:* rwa_wdm.rwa.ga.pop.Population, *pop_size: int*, *tourn_size: int = 3*) → *rwa_wdm.rwa.ga.pop.Population*

Tournament selection strategy

First we choose a random candidate from population. Then, under trials, we choose another candidate and compare the two fitnesses. The winner becomes the top candidate; loser is eliminated.

Parameters

- **population** – Population instance object after evaluation
- **pop_size** – number of individuals in the mating pool pre-crossover
- **tourn_size** – number of individuals to compete under the tournament pool

Returns set of parents ready to mate under crossover operation

Return type *Population*

2.3 Input / Output

I/O related operations such as R/W data from disk and viz-related ops

`rwa_wdm.io.`**`plot_bp`**(*result_dir: str*) → None
Reads blocking probabilities from file and plot overlapping graph

Parameters **result_dir** – directory that stores files to be read

`rwa_wdm.io.`**`write_bp_to_disk`**(*result_dir: str*, *filename: str*, *bplist: List[float]*) → None
Writes blocking probabilities to text file

Parameters

- **result_dir** – directory to write files to
- **filename** – name of the file to be written
- **itlist** – list of blocking probability values, as percentages, to be dumped to file

`rwa_wdm.io.`**`write_it_to_disk`**(*result_dir: str*, *filename: str*, *itlist: List[float]*) → None
Writes profiling time information to text file

Parameters

- **result_dir** – directory to write files to
- **filename** – name of the file to be written
- **itlist** – list of times, in seconds, to be dumped to file

- Source code and issue tracker: https://github.com/cassiobatista/rwa-wdm-sim

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search

CHAPTER

FOUR

CREDITS

Created by Cassio Batista

CHAPTER

FIVE

LICENSE

The `rwa_wdm` latest package is provided under GNU Public License v3.0.

PYTHON MODULE INDEX

r

INDEX

Symbols

A

B

C

D

E

F

G