Tester, Reviewer and Aggregator of the Use of Modules Applied to Scheduling

TRAUMAS 0.1

Gautier Raimondi

Jun 02, 2020

CONTENTS:

CHAPTER ONE

INTRODUCTION

1.1 Objective

This script aims to provide a testing base for a set of module-based heuristics for the $R|prec|C_{\max}$ problem.

1.2 Conventions

This entire document is written in serif.

More over, every glossary term shall be a link to its definition and its first occurrence be followed by an asterisk (*).

1.3 Project Description

This project should allow any user to compile data related to the performances and the runtime of heuristics solving the $R|prec|C_{\max}$ problem.

To do so, three scripts are available to use. Each one of them will be described in further sections.

CHAPTER

TWO

ENTRY POINTS

2.1 Main script

The main script is main.py. The usage can be found below :

```
usage: main.py [-h] [-v] [--version] [-m] [-mf MAILFILENAME] (-g SIZE DEPTH GENERATEDFILENAME ETCOMP
ETCOMM CCR | -G GRAPHFILE) (-H PRIO COST PLACEMENT BIM BSA INS | -a N | -n) [-p NBPROC] [-s SEED]
filename

Run heuristics on generated graph

positional arguments:
  filename              output file

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         print non-necessary information
  --version             show program's version number and exit
  -m, --mail            send results by mail
  -mf MAILFILENAME, --mailfilename MAILFILENAME
                        file containing mail information
  -g SIZE DEPTH GENERATEDFILENAME ETCOMP ETCOMM CCR, --generate SIZE DEPTH GENERATEDFILENAME ETCOMP
ETCOMM CCR
                        generate graph to use.
                        the mean of computation cost is supposed to be 1, so CCR and standard deviation
are sufficient to generate the graph.
  -G GRAPHFILE, --graphfile GRAPHFILE
                        use .gml file
  -H PRIO COST PLACEMENT BIM BSA INS, --heuristic PRIO COST PLACEMENT BIM BSA INS
                        define heuristic to test.
                        for example, -H rku mean eft 0 0 1.
                        an overview of possible combinations can be found in man.pdf.
  -a N, --all N         try all heuristics N times
  -n, --nothing         generate the graph then exit
  -p NBPROC, --nbproc NBPROC
                        number of processors available
  -s SEED, --seed SEED  seed for the random generation
```

where the following values are allowed for –heuristic :

PRIO : ‘rku’, ‘random’, ‘BIL’, ‘rkd’, ‘cluHPS’, ‘rkusd’, ‘rkuad’

COST : ‘mean’, ‘median’, ‘maxmax’, ‘minmax’, ‘minmin’, ‘maxmin’

PLACEMENT : ‘eft’, ‘BIM*’, ‘OLB’, ‘MET’, ‘DL’, ‘GDL’

DESC : ‘DLS/DC’, ‘DCP’, None

BIM : 0, 1

BSA : 0, 1

INSERTION : 0, 1

It is also to be noted that –nbproc should not be higher than the value used when generating the graph.

2.2 allScript.py

2.3 resAggregator.py

CHAPTER

THREE

HELP FUNCTIONS

3.1 Converter

`Converter.`**`igraphToNetworkX`**`(`*inputFile*, *output*, *sdComp*, *sdComm*, *CCR*, *nbproc*`)`

Transform an igraph file, from recursiveGenLength.py, to an networkx one. To be simple, just adding computations and communications cost, as well as root and sink nodes.

Parameters

- **`inputFile`** (*`str`*) – Igraph file, to transform
- **`output`** (*`str`*) – Output file
- **`sdComp`** (*`float`*) – Standard Deviation of computations cost
- **`sdComm`** (*`float`*) – Standard Deviation of communications cost
- **`CCR`** (*`float`*) – Communications to Computations Ratio
- **`nbproc`** (*`int`*) – Number of processor for the graph generation

Returns Converted graph

Return type networkx.DiGraph

`Converter.`**`listEntryNodes`**`(`*g*`)`

Compute list of current entry nodes for **g**

Parameters **`g`** (*`networkx.DiGraph`*) – DAG to schedule

Returns List of entry nodes

Return type list[int]

`Converter.`**`listExitNodes`**`(`*g*`)`

Compute list of current exit nodes for **g**

Parameters **`g`** (*`networkx.DiGraph`*) – DAG to schedule

Returns List of exit nodes

Return type list[int]

3.2 FileReader

`FileReader.`**`readFile`**(*inputFile*, *converter=False*, *verbose=False*)

Read a .gml file and create the corresponding Digraph.

Parameters

- **`inputFile`** (*`str`*) – File to parse
- **`converter`** (*`bool`*) – Use id as label ?
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Returns Created digraph

Return type networkx.DiGraph

3.3 GraphGenerator

`GraphGenerator.`**`genGraph`**(*length*, *depth*, *filename*, *sdComp*, *sdComm*, *CCR*, *nbproc*)

Generate a graph using recursiveGenLength.py

Parameters

- **`length`** (*`int`*) – Length of the graph (number of nodes)
- **`depth`** (*`int`*) – Depth of the graph (number of levels)
- **`filename`** (*`str`*) – Output filename
- **`sdComp`** (*`float`*) – Standard Deviation of computations costs
- **`sdComm`** (*`float`*) – Standard Deviation of communications costs
- **`CCR`** (*`float`*) – Communications to Computations Ratio
- **`nbproc`** (*`int`*) – Number of processors used when generating the graph

Returns Generated and converted graph

Return type networkx.DiGraph

3.4 MailSender

`MailSender.`**`sendMail`**(*n*, *results*)

Send a mail containing the results to a mail address contained in “mailinfo:3”, using gmail address “mailinfo:1” and password “mailinfo:2”.

Parameters

- **`n`** (*`int`*) – Number of batches used in the run
- **`results`** (*`str`*) – Message to transfer

Return type None

3.5 Parser

`Parser.`**`defineParser`**`()`

Define the script parser

Returns Parser to use

Return type argparse.ArgumentParser

3.6 Printer

`Printer.`**`printSchedule`**`(`*schedule*`)`

Print the schedule in a more readable way (transform proc name)

Parameters **`schedule`** (*`dict[int, (int, float, float)]`*) – Schedule to print

Return type None

CHAPTER

FOUR

EXECUTIONS

4.1 Execution Strategies

`Executions.`**`execBIL`**`(`*g*, *verbose=False*`)`

Use BIL policy to compute priority

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Returns List of nodes sorted according to BIL

Return type list[int]

`Executions.`**`execCluHPS`**`(`*g*, *costFunction='mean'*, *verbose=False*`)`

Use HPS policy to compute priority and organise nodes

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`costFunction`** (*`str`*) – Function used to simplify comp/comm cost
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Returns List of nodes sorted according to LC (Link Cost)

Return type list[int]

`Executions.`**`execRKD`**`(`*g*, *costFunction='mean'*, *verbose=False*`)`

Use rkd policy to compute priority

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`costFunction`** (*`str`*) – Function used to simplify comp/comm cost (‘mean’, ‘median’, ‘maxmax’, ‘minmin’, ‘minmax’, ‘maxmin’)
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Returns List of nodes sorted according to rku

Return type list[int]

`Executions.`**`execRKU`**`(`*g*, *costFunction='mean'*, *verbose=False*`)`

Use rku policy to compute priority

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`costFunction`** (*`str`*) – Function used to simplify comp/comm cost (‘mean’, ‘median’, ‘maxmax’, ‘minmin’, ‘minmax’, ‘maxmin’)

- **verbose** (*bool*) – Print non-necessary information ?

Returns List of nodes sorted according to rku

Return type list[int]

`Executions.`**`execRKUAD`**(*g*, *costFunction='mean'*, *verbose=False*)

Use rku+rkd policy to compute priority

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **costFunction** (*str*) – Function used to simplify comp/comm cost ('mean', 'median', 'maxmax', 'minmin', 'minmax', 'maxmin')
- **verbose** (*bool*) – Print non-necessary information ?

Returns List of nodes sorted according to rku

Return type list[int]

`Executions.`**`execRKUSD`**(*g*, *costFunction='mean'*, *verbose=False*)

Use rku-rkd policy to compute priority

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **costFunction** (*str*) – Function used to simplify comp/comm cost ('mean', 'median', 'maxmax', 'minmin', 'minmax', 'maxmin')
- **verbose** (*bool*) – Print non-necessary information ?

Returns List of nodes sorted according to rku

Return type list[int]

4.2 Extensive Test

`ExtensiveTest.`**`realTryHard`**(*g*, *n*, *verbose=False*, *graphname=''*)

Try every heuristics a given number of time, to measure a meaningful runtime

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **n** (*int*) – Number of batches to realize
- **verbose** (*bool*) – Print non-necessary information ?
- **graphname** (*str*) – Name of the graph used, for error tracking purpose

Returns Results of each heuristics and runtime in ms {heuristic : [makespan, runtime]}

Return type dict[str, list[float]]

`ExtensiveTest.`**`tryEverything`**(*g*, *verbose*, *graphname*)

Try every heuristic possible according to given array

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **verbose** (*bool*) – Print non-necessary information ?
- **graphname** (*str*) – Name of the graph used, for error tracking purpose

Returns Results of each heuristics and runtime in ms {heuristic : [makespan, runtime]}

Return type dict[str, list[float]]

4.3 Total Computation

`TotalComputation.`**`computeSchedule`**(*g*, *category='list'*, *strategyPrio='rku'*, *strategyPlacement='eft'*, *costFunction='mean'*, *desc=None*, *useOfBIM=False*, *insertion=True*, *bsa=False*, *verbose=False*)

Compute schedule

Parameters

- **`g`** (*`nx.DiGraph`*) – DAG to schedule
- **`category`** (*`str`*) – Type of scheduling algorithm to use (‘list’, ‘clustering’)
- **`strategyPrio`** (*`str`*) – Priority-computations strategy (‘rku’, ‘random’, ‘BIL’)
- **`strategyPlacement`** (*`str`*) – Placement-strategy (‘eft’, ‘BIM*’, ‘OLB’, ‘MET’)
- **`costFunction`** (*`str`*) – Function used to simplify comp/comm cost (‘mean’, ‘median’, ‘maxmax’, ‘minmin’, ‘minmax’, ‘maxmin’)
- **`desc`** (*`str`*) – Lookahead strategy (‘DLS/DC’, None)
- **`useOfBIM`** (*`bool`*) – Use of BIM strategy (k-th smallest) ?
- **`insertion`** (*`bool`*) – Use of insertion-based policy ?
- **`bsa`** (*`bool`*) – Use of BSA post-treatment ?
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Returns A corresponding schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

`TotalComputation.`**`computeWithList`**(*g*, *strategyPrio='rku'*, *strategyPlacement='eft'*, *costFunction='mean'*, *desc=None*, *useOfBIM=False*, *insertion=True*, *bsa=False*, *verbose=False*)

Compute a schedule using list-based heuristic

Parameters

- **`g`** (*`nx.DiGraph`*) – DAG to schedule
- **`strategyPrio`** (*`str`*) – Priority-computations strategy (‘rku’, ‘random’, ‘BIL’, ‘rkd’, ‘rkusd’, ‘rkuad’, ‘cluHPS’)
- **`strategyPlacement`** (*`str`*) – Placement-strategy (‘eft’, ‘BIM*’, ‘OLB’, ‘MET’, ‘DL’, ‘GDL’)
- **`costFunction`** (*`str`*) – Function used to simplify comp/comm cost (‘mean’, ‘median’, ‘maxmax’, ‘minmin’, ‘minmax’, ‘maxmin’)
- **`desc`** (*`str`*) – Lookahead strategy (‘DLS/DC’, None)
- **`useOfBIM`** (*`bool`*) – Use of BIM strategy (k-th smallest) ?
- **`insertion`** (*`bool`*) – Use of insertion-based policy ?
- **`bsa`** (*`bool`*) – Use of BSA post-treatment ?
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Raises

- *`StrategyPrioException`* – If strategyPrio is unknown
- *`StrategyPlacementException`* – If strategyPlacement is unknown

Returns A corresponding schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

CHAPTER

FIVE

COMPUTATIONS

5.1 CommCost

`CommCost.`**`commCost`**`(`*g*, *i*, *j*, *m*, *n*, *verbose=False*`)` → float
Compute exact communication cost using startup time, data quantity, and transfer rate

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG used
- **`i`** (*`int`*) – Starting node of the communication
- **`j`** (*`int`*) – Ending node of the communication
- **`m`** (*`int`*) – Processor on which **i** is scheduled
- **`n`** – Processor on which **j** is scheduled
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Returns The communication cost between (**i**, **m**) and (**j**, **n**)

Return type float

`CommCost.`**`meanCommCost`**`(`*g*, *i*, *j*, *verbose=False*`)`
Compute communication cost using mean value for startup time and transfer rate

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG used
- **`i`** (*`int`*) – Starting node of the communication
- **`j`** (*`int`*) – Ending node of the communication
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Returns The mean communication cost between **i** and **j**

Return type float

5.2 CompCost

`CompCost.`**`computeCompCost`**`(`*g*, *costFunction='mean'*, *verbose=False*`)`
Compute computation cost according to costFunction

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG used
- **`costFunction`** (*`str`*) – Function used to compute computation cost ('mean', 'median', 'max', 'min')
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Return type None

5.3 EarliestTimes

`EarliestTimes.`**`computeDFT`**(*g*, *node*, *proc*, *schedule*, *verbose=False*, *estimate=False*)

Compute Data Finish Time for a given node on a given proc according to a given schedule (id est time of arrival of every communications from predecessors)

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG used
- **`node`** (*`int`*) – Node to try scheduling
- **`proc`** (*`int`*) – Proc to schedule **node** on
- **`schedule`** (*`dict[int, (int, float, float)]`*) – Tasks already scheduled to this point
- **`verbose`** (*`bool`*) – Print non-necessary information ?
- **`estimate`** (*`bool`*) – Do we know the scheduling of all predecessors ? #TODO Test this

Returns DFT and list of tasks on proc **proc**

Return type (float,list[list[int]])

`EarliestTimes.`**`computeEFT`**(*g*, *node*, *proc*, *schedule*, *verbose=False*, *insertion=False*, *estimate=False*)

Compute Earliest Finish Time for a given node on a given proc according to a given schedule

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG used
- **`node`** (*`int`*) – Node to try scheduling
- **`proc`** (*`int`*) – Proc to schedule **node** on
- **`schedule`** (*`dict[int, (int, float, float)]`*) – Tasks already scheduled to this point
- **`verbose`** (*`bool`*) – Print non-necessary information ?
- **`insertion`** (*`bool`*) – Use of insertion-based policy ?
- **`estimate`** (*`bool`*) – Do we know the scheduling of all predecessors ? #TODO Test this

Returns EST and EFT for given node

Return type (float,float)

5.4 LBMatrix

`LBMatrix.`**`computeLB`**(*g*, *costFunction='mean'*, *verbose=False*)

Compute communication cost according to costFunction

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG used
- **`costFunction`** (*`str`*) – Function used to compute communication cost (‘mean’, ‘median’, ‘max’, ‘min’)
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Return type None

5.5 Lookahead

`Lookahead.`**`DC`**(*g*, *i*, *m*)

Compute the DC term of DLS *id est* a Descendant Consideration term

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **i** (*int*) – Task to schedule
- **m** (*int*) – Proc to schedule **i** on

Returns DC(i,m)

Return type float

`Lookahead.`**`DCP`**(*g*, *i*, *m*, *nodes*, *placeStrat*, *schedule*, *placeValue*)

Compute the DCP lookahead strategy

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **i** (*int*) – Task to schedule
- **m** (*int*) – Current scheduling proc for **i**
- **placeValue** (*float*) – Current placement value

Returns Final placement value

Return type float

`Lookahead.`**`DLSDC`**(*g*, *i*, *m*, *nodes*, *placeStrat*, *schedule*, *placeValue*)

Compute and add the Descendant Consideration term to the current placement value

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **i** (*int*) – Task to schedule
- **m** (*int*) – Current scheduling proc for **i**
- **placeValue** (*float*) – Current placement value

Returns Final placement value

Return type float

`Lookahead.`**`F`**(*g*, *i*, *j*, *m*)

Compute the F term of DLS *id est* how quickly succ(i) can be completed on any other processor than PE(i)

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **i** (*int*) – Task to schedule
- **j** (*int*) – Descendant to which **i** passes the most data
- **m** (*int*) – Processor on which is scheduled **i**

Returns F(i, j, m)

Return type float

`Lookahead.`**`NOP`**(**args*)

Return the placement value passed in parameters, doing nothing

Parameters **`args`** (*`List[Any, .., float]`*) – Set of params, should end by the current placement value

Returns Current placement value

Return type float

`Lookahead.`**`getLookAheadFun`**(*name*)

Return the function associated with the name stored in **name**

Parameters **`name`** (*`str`*) – Lookahead strategy to apply

Returns The function to run to compute the actual placement value

Return type (networkx.DiGraph, int, int, list[int], str, dict[int, (int, float, float)], float) -> float

5.6 Placements

`Placements.`**`aux`**(*g*, *CP*, *serialOrder*, *tx*)

Construct the serial order of nodes using the CP

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`CP`** (*`list[int]`*) – Critical Path of **g**
- **`serialOrder`** (*`list[int]`*) – Serial Order to complete
- **`tx`** (*`int`*) – Node to treat

`Placements.`**`computeCurrentNodeBIM`**(*g*, *readyTasks*, *schedule*, *verbose=False*, *insertion=True*)

Determine node to schedule using the BIM policy

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`readyTasks`** (*`list[int]`*) – List of ready tasks
- **`schedule`** (*`dict[int, (int, float, float)]`*) – Schedule at this point
- **`verbose`** (*`bool`*) – Print non-necessary information ?
- **`insertion`** (*`bool`*) – Use of insertion policy ?

Returns Node to schedule

Return type int

`Placements.`**`findBestProcBIMStar`**(*g*, *currentNode*, *schedule*, *k*, *desc*, *verbose*, *insertion*, *nodes*)

Find the best proc to schedule **currentNode** according to the BIM* policy and schedule **currentNode** to this proc

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`currentNode`** (*`int`*) – Node to schedule
- **`schedule`** (*`dict[int, (int, float, float)]`*) – Schedule at this point
- **`k`** (*`int`*) – Number of ready tasks at this point
- **`desc`** (*`str`*) – Lookahead strategy
- **`verbose`** (*`bool`*) – Print non-necessary information ?
- **`insertion`** (*`bool`*) – Use of insertion policy ?
- **`nodes`** (*`list[int]`*) – List of nodes sorted

`Placements.`**`findBestProcEFT`**(*g*, *currentNode*, *schedule*, *desc*, *verbose*, *insertion*, *useEST*, *nodes*)
Find the best proc to schedule **currentNode** according to a EFT policy and schedule **currentNode** to this proc

Parameters

- **`nodes`** (*`list[int]`*) – List of nodes sorted
- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`currentNode`** (*`int`*) – Node to schedule
- **`schedule`** (*`dict[int, (int, float, float)]`*) – Schedule at this point
- **`desc`** (*`str`*) – Lookahead strategy
- **`verbose`** (*`bool`*) – Print non-necessary information ?
- **`insertion`** (*`bool`*) – Use of insertion policy ?
- **`useEST`** (*`bool`*) – Use EST to compare instead of EFT ?

`Placements.`**`placeBIMStar`**(*g*, *nodes*, *desc*, *verbose*, *insertion*)
Use BIM* policy to schedule node on processors

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`nodes`** (*`list[int]`*) – Ordered list of nodes
- **`desc`** (*`str`*) – Lookahead strategy
- **`verbose`** (*`bool`*) – Print non-necessary information ?
- **`insertion`** (*`bool`*) – Use of insertion policy ?

Returns A corresponding schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

`Placements.`**`placeBIMStarBIM`**(*g*, *nodes*, *desc*, *verbose*, *insertion*)
Use of BIM* policy along with a BIM selection of node to schedule node on processors

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`nodes`** (*`list[int]`*) – Ordered list of nodes
- **`desc`** (*`str`*) – Lookahead strategy
- **`verbose`** (*`bool`*) – Print non-necessary information ?
- **`insertion`** (*`bool`*) – Use of insertion policy ?

Returns A corresponding schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

`Placements.`**`placeDL`**(*g*, *nodes*, *desc*, *verbose*, *insertion*, *costFunction='mean'*)
Use DL policy to schedule node on processors

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`nodes`** (*`list[int]`*) – Ordered list of nodes
- **`desc`** (*`str`*) – Lookahead strategy
- **`verbose`** (*`bool`*) – Print non-necessary information ?
- **`insertion`** (*`bool`*) – Use of insertion policy ?
- **`costFunction`** (*`str`*) – Function used to simplify comp/comm cost

Returns A corresponding schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

`Placements.`**`placeDLBIM`**(*g*, *nodes*, *desc*, *verbose*, *insertion*, *costFunction='mean'*)
Use DL policy along with a BIM selection of nodes to schedule node on processors

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`nodes`** (*`list[int]`*) – Ordered list of nodes
- **`desc`** (*`str`*) – Lookahead strategy
- **`verbose`** (*`bool`*) – Print non-necessary information ?
- **`insertion`** (*`bool`*) – Use of insertion policy ?
- **`costFunction`** (*`str`*) – Function used to simplify comp/comm cost

Returns A corresponding schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

`Placements.`**`placeEFT`**(*g*, *nodes*, *desc=None*, *verbose=False*, *insertion=True*)
Use EFT policy (minimize Earliest Finish Time) to schedule node on processors

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`nodes`** (*`list[int]`*) – Ordered list of nodes
- **`desc`** (*`str`*) – Lookahead strategy
- **`verbose`** (*`bool`*) – Print non-necessary information ?
- **`insertion`** (*`bool`*) – Use of insertion policy ?

Returns A corresponding schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

`Placements.`**`placeEFTBIM`**(*g*, *nodes*, *desc=None*, *verbose=False*, *insertion=True*)
Use EFT policy along with a BIM selection of node to schedule node on processors

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`nodes`** (*`list[int]`*) – Ordered list of nodes
- **`desc`** (*`str`*) – Lookahead strategy
- **`verbose`** (*`bool`*) – Print non-necessary information ?
- **`insertion`** (*`bool`*) – Use of insertion policy ?

Returns A corresponding schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

`Placements.`**`placeGDL`**(*g*, *nodes*, *desc*, *verbose*, *insertion*, *costFunction='mean'*)
Use GDL policy to schedule node on processors

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`nodes`** (*`list[int]`*) – Ordered list of nodes
- **`desc`** (*`str`*) – Lookahead strategy
- **`verbose`** (*`bool`*) – Print non-necessary information ?

- **insertion** (*bool*) – Use of insertion policy ?
- **costFunction** (*str*) – Function used to simplify comp/comm cost

Returns A corresponding schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

`Placements.`**placeGDLBIM**(*g*, *nodes*, *desc*, *verbose*, *insertion*, *costFunction='mean'*)

Use GDL policy along with a BIM selection to schedule node on processors

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **nodes** (*list[int]*) – Ordered list of nodes
- **desc** (*str*) – Lookahead strategy
- **verbose** (*bool*) – Print non-necessary information ?
- **insertion** (*bool*) – Use of insertion policy ?
- **costFunction** (*str*) – Function used to simplify comp/comm cost

Returns A corresponding schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

`Placements.`**placeMET**(*g*, *nodes*, *desc=None*, *verbose=False*, *insertion=True*)

Use MET (each task to best proc) policy to schedule node on processors

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **nodes** (*list[int]*) – Ordered list of nodes
- **desc** (*str*) – Lookahead strategy
- **verbose** (*bool*) – Print non-necessary information ?
- **insertion** (*bool*) – Use of insertion policy ?

Returns A corresponding schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

`Placements.`**placeNode**(*g*, *currentNode*, *proc*, *schedule*, *nodes*, *readyTasks*, *verbose*, *insertion*)

Place node in schedule, and update ready tasks list

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **currentNode** (*int*) – Node to schedule
- **proc** (*int*) – Processor to schedule **currentNode** on
- **schedule** (*dict[int, (int, float, float)]*) – Schedule at this point
- **nodes** (*list[int]*) – Ordered list of nodes
- **readyTasks** (*list[int]*) – List of ready tasks
- **verbose** (*bool*) – Print non-necessary information ?
- **insertion** (*bool*) – Use of insertion policy ?

Return type None

`Placements.`**placeOLB**(*g*, *nodes*, *desc=None*, *verbose=False*, *insertion=True*)

Use OLB (balancing work load) policy to schedule node on processors

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **nodes** (*list[int]*) – Ordered list of nodes
- **desc** (*str*) – Lookahead strategy
- **verbose** (*bool*) – Print non-necessary information ?
- **insertion** (*bool*) – Use of insertion policy ?

Returns A corresponding schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

Placements.**placeSerial**(*g*, *strategyPrio*, *verbose*)
Use serial order to schedule node on processors

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **strategyPrio** (*str*) – Priority-computations strategy
- **verbose** (*bool*) – Print non-necessary information ?

Returns A corresponding schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

Placements.**updateReadyTasks**(*g*, *readyTasks*, *nodes*, *scheduledNode*, *deletion=True*, *verbose=False*)
Update the list of ready tasks after scheduling of a node

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **readyTasks** (*list[int]*) – List of ready tasks
- **nodes** (*list[int]*) – Ordered list of nodes
- **scheduledNode** (*int*) – Node scheduled
- **deletion** (*bool*) – Should we delete readyTask from nodes ?
- **verbose** (*bool*) – Print non-necessary information ?

Return type None

5.7 PostTraitement

PostTraitement.**applyBSA**(*g*, *schedule: dict*, *verbose=False*)
Apply BSA to an already-computed schedule

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **schedule** (*dict[int, (int, float, float)]*) – Schedule computed, to improve using BSA
- **verbose** (*bool*) – Print non-necessary information ?

Returns A *possibly* improved schedule in format {task : [proc, est, eft],..}

Return type dict[int, (int, float, float)]

PostTraitement.**verifBSA**(*rs*, *r*, *verbose=False*)
Check over the schedule if using BSA really improve the performance

Parameters

- **rs** (*str[]*) – Sorted heuristics by (makespan, computation time)
- **r** (*dict[int, (int, float, float)]*) – Full results of the extensive testing
- **verbose** (*bool*) – Print non-necessary information ?

Returns Analysis of BSA-related improvement in performance

Return type str

5.8 Priorities

`Priorities.`**`computeBIL`**(*g*, *verbose=False*)

Compute BIL for every node and every proc of **g** and store it in g.graph['prio']

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **verbose** (*bool*) – Print non-necessary information ?

`Priorities.`**`computeLC`**(*g*, *lvl*, *verbose=False*)

Compute LC, *id est* Link Cost, for each node of **g**, and store it in g.graph['prio']

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **lvl** (*dict[int, list[int]]*) – List of nodes per level of **g**
- **verbose** –

Returns List of nodes sorted by LC

Return type list[int]

`Priorities.`**`computeLevels`**(*g*, *verbose=False*)

Compute level/depth for each node of **g** *id est*

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **verbose** (*bool*) – Print non-necessary information ?

Returns Dict containing nodes for each level

Return type dict[int, list[int]]

`Priorities.`**`computeRankD`**(*g*, *costFunction='mean'*, *verbose=False*, *add=False*, *sub=False*)

Compute rkd for every node of **g** and store it in g.graph['prio']

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **costFunction** (*str*) – Function used to compute meanCompCost and meanComm-Cost
- **verbose** (*bool*) – Print non-necessary information ?
- **add** (*bool*) – Add rkd to current prio ?
- **sub** (*bool*) – Subtract rkd to current prio ?

`Priorities.`**`computeRankU`**(*g*, *costFunction='mean'*, *verbose=False*)

Compute rku for every node of **g** and store it in g.graph['prio']

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule

- **`costFunction`** (*`str`*) – Function used to compute meanCompCost and meanCommCost
- **`verbose`** (*`bool`*) – Print non-necessary information ?

`Priorities.`**`getCP`**(*g*, *costFunction='mean'*, *verbose=False*)
Compute Critical-Path of given graph

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`costFunction`** (*`str`*) – Function used to compute computation cost ('mean', 'median', 'max', 'min')
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Returns Critical Path of **g**

Return type str

`Priorities.`**`getEntryTask`**(*g*, *verbose=False*)
Compute if necessary the entry task of the graph, else just return it

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Returns Entry task of **g**

Return type int

`Priorities.`**`getExitTask`**(*g*, *verbose=False*)
Compute if necessary the exit task of the graph, else just return it

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Returns Exit task of **g**

Return type int

`Priorities.`**`identifyCP`**(*g*, *strategyPrio*, *verbose*)
Identify the CP of **g** using **strategyPrio**

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`strategyPrio`** (*`str`*) – Prioritization strategy, affect the process of CP selection
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Returns The identified CP

Return type list[int]

CHAPTER

SIX

TESTS

6.1 Precedence Verification

`VerifPrecedence.`**`verifPrec`**`(`*g*, *schedule*, *verbose*`)`

Verify precedence constraint and job length of a given schedule

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule
- **`schedule`** (*`dict[int, (int, float, float)]`*) – Schedule to check
- **`verbose`** (*`bool`*) – Print non-necessary information ?

CHAPTER

SEVEN

METRICS

7.1 Sequential Makespan

`Sequential.`**`sequentialScheduleLength`**(*g*, *verbose=False*)

Compute sequential schedule on processor minimizing total scheduling time

Parameters

- **g** (*networkx.DiGraph*) – Used DAG
- **verbose** (*bool*) – Print non-necessary information ?

Returns Sequential makespan

Return type int

7.2 SLR

`SLR.`**`computeSLR`**(*g*, *schedule*, *verbose=False*)

Compute the SLR of the computed schedule, *id est* ration of obtained makespan over makespan of CP.

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule
- **schedule** (*dict[int, (int, float, float)]*) – Schedule obtained
- **verbose** (*bool*) – Print non-necessary information ?

Returns SLR, the smaller the better

Return type float

7.3 Speedup

`Speedup.`**`measureGeneralEfficiency`**(*g*, *schedule*, *verbose=False*)

Compute general efficiency of computed schedule, *id est* sequential makespan over (makespan × total number of procs) ratio

Parameters

- **g** (*networkx.DiGraph*) – DAG to schedule:
- **schedule** (*dict[int, (int, float, float)]*) – Schedule obtained
- **verbose** (*bool*) – Print non-necessary information ?

Returns Computed general efficiency, higher the better

Return type float

`Speedup.`**`measureSpecificEfficiency`**(*g*, *schedule*, *verbose=False*)

Compute specific efficiency of computed schedule, *id est* sequential makespan over (makespan $\times$ number of used procs) ratio

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule:
- **`schedule`** (*`dict[int, (int, float, float)]`*) – Schedule obtained
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Returns Computed specific efficiency, higher the better

Return type float

`Speedup.`**`measureSpeedup`**(*g*, *schedule*, *verbose=False*)

Compute speedup of computed schedule, *id est* sequential makespan over makespan ratio

Parameters

- **`g`** (*`networkx.DiGraph`*) – DAG to schedule:
- **`schedule`** (*`dict[int, (int, float, float)]`*) – Schedule obtained
- **`verbose`** (*`bool`*) – Print non-necessary information ?

Returns Computed speedup, higher the better

Return type float

CHAPTER

EIGHT

EXCEPTIONS

exception `StrategyLookAheadException.`**`StrategyLookAheadException`**(*message*)
Unknown lookahead strategy Exception

exception `StrategyPlacementException.`**`StrategyPlacementException`**(*message*)
Unknown placement strategy Exception

exception `StrategyPrioException.`**`StrategyPrioException`**(*message*)
Unknown prioritization strategy Exception

PYTHON MODULE INDEX