

# Benchmarking Combinatorially with Python

Massimo Bono

Università degli Studi di Brescia

December 3, 2019



UNIVERSITÀ  
DEGLI STUDI  
DI BRESCIA



# Introduction

You have an awesome algorithm, **but**:

How can you ensure that it behaves better w.r.t. state-of-the-art?

- ◇ Theoretically? What if it's too complex?
- ◇ By comparing its *performances* on different *scenarios* with different *state-of-the-art algorithms*?



**Problem:** the scenarios and the comparing algorithms may lead to too many combinations!

**Goal:** design an extensible framework that automatically performs all the combinations required for your performance testing;

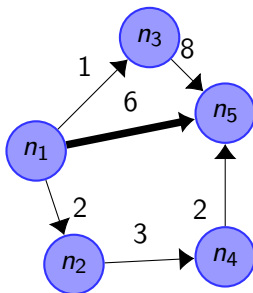
**Result** a new Python software called `phd-tester` which performs automatic testing. Code available at

[HTTPS://GITHUB.COM/KOLDAR/PHDTESTER](https://github.com/Koldar/phdtester)

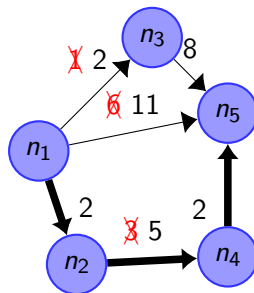
# A Motivating Example (1)

**Context:** *dynamic* single agent pathfinding (given a weighted directed graph  $\mathcal{G} = \langle N, E \rangle$ , a *start* vertex  $s$ , a *goal* vertex  $t$ , what is the optimal path allowing an agent to reach  $t$  from  $s$ ?); *dynamic* in the sense that, at the beginning of each pathfinding, some (arbitrary) edges temporary increase their cost (*perturbations*).

**Example:** going from  $s = n_1$  to  $t = n_5$



(a) original graph



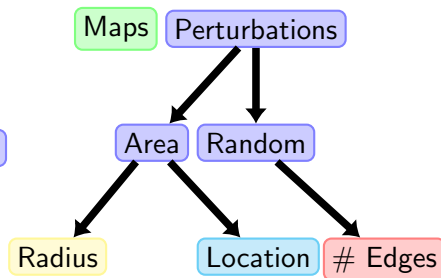
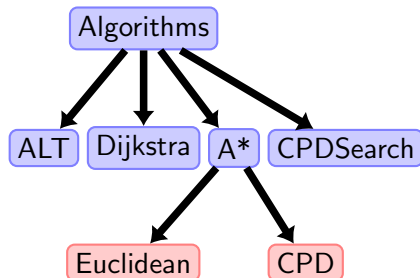
(b) Graph temporary altered

## A Motivating Example (2)

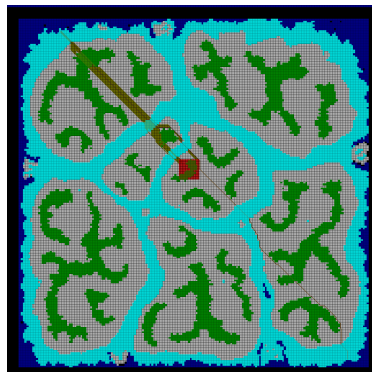
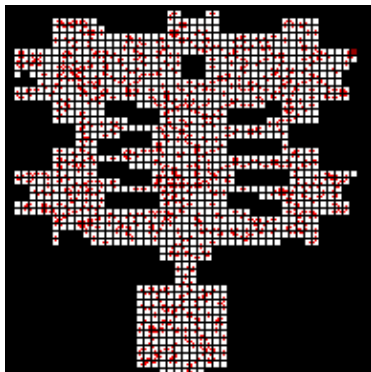
We can solve it:

- ◇ 4 Algorithms: Dijkstra, ALT,  $A^*$  + Euclidean Distance,  $A^*$  + CPD, **CPD search** (our technique!);
- ◇ On which map? hrt201, dustwallowkeys, mazes512-1-4, rooms16-003, ...;
- ◇ How to generate perturbations? **R**andomly? If so how many **E**edges should we affect? Or by affecting a specific **A**rea along the optimal path? If so, how **W**ide the area is? Where should we affect the **P**ath?

# A Motivating Example (3)



# A Motivating Example (4)



**Figure:** Example of perturbation policies. perturbations are shown in red.  
Left: random; Right: area

# A Motivating Example (5)

$$|Algs| = 5, |Maps| = 4, |Perturbations| = 2$$



$$Algs \times Maps \times Perturbations = 32$$

But **R** depends on parameter **E** while **A** depends on **W** and **P** parameters!

If we can choose the parameters:

- $E \in \{1\%, 5\%, 10\%\};$
- $P \in \{20\%, 50\%, 100\%\};$
- $W \in \{5, 10, 15\};$

Maximum combinations:  $32 \cdot |E \times P \times W| = 864!$  If we add new parameters (which, in research, it happens quite often) maximum combinations significantly increases!

# Talk Outline

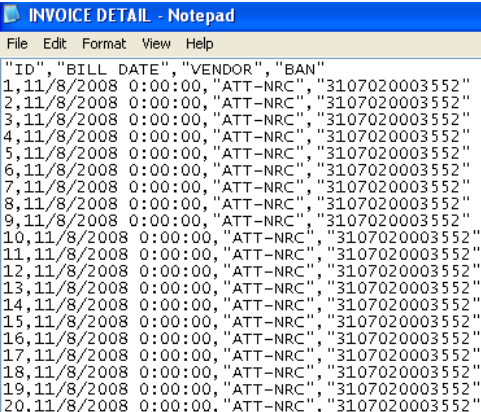
The talk will be outlined as follows:

- ◇ Background: Software and design patterns used;
- ◇ Proposed Technique: KS001, phd-tester and the general framework flow;
- ◇ Conclusion and Future Works: integration with CSPs and generates automatic report pdf;



# CSV file format

- Easy for storing data in “tabular” format;
- Human readable;
- can be read by tabular softwares (i.e., Excel, LibreOffice Calc);



ID	BILL DATE	VENDOR	BAN
1	11/8/2008 0:00:00	ATT-NRC	3107020003552
2	11/8/2008 0:00:00	ATT-NRC	3107020003552
3	11/8/2008 0:00:00	ATT-NRC	3107020003552
4	11/8/2008 0:00:00	ATT-NRC	3107020003552
5	11/8/2008 0:00:00	ATT-NRC	3107020003552
6	11/8/2008 0:00:00	ATT-NRC	3107020003552
7	11/8/2008 0:00:00	ATT-NRC	3107020003552
8	11/8/2008 0:00:00	ATT-NRC	3107020003552
9	11/8/2008 0:00:00	ATT-NRC	3107020003552
10	11/8/2008 0:00:00	ATT-NRC	3107020003552
11	11/8/2008 0:00:00	ATT-NRC	3107020003552
12	11/8/2008 0:00:00	ATT-NRC	3107020003552
13	11/8/2008 0:00:00	ATT-NRC	3107020003552
14	11/8/2008 0:00:00	ATT-NRC	3107020003552
15	11/8/2008 0:00:00	ATT-NRC	3107020003552
16	11/8/2008 0:00:00	ATT-NRC	3107020003552
17	11/8/2008 0:00:00	ATT-NRC	3107020003552
18	11/8/2008 0:00:00	ATT-NRC	3107020003552
19	11/8/2008 0:00:00	ATT-NRC	3107020003552
20	11/8/2008 0:00:00	ATT-NRC	3107020003552

# Background and technologies used

- ◇ Language: Python 3[.6]:  
duck typing, scripting, ABC  
library, usually installed by  
default;
- ◇ ArangoDB: NoSQL  
graph-based database;  
basically we can store  
JSON; Compactly represent  
data;
- ◇ Pandas: allows operations  
on “dataframe” ( $\approx$  CSV);

```
{ ▼ 6 properties, 427 bytes
  "manifest_version": 2,
  "name": "JSON Lite",
  "version": "0.12",
  "description":
    large files, collapse items",
  "browser_action": { ▼ 1 property, 40 bytes
    "default_popup": "README.html"
  },
  "content_scripts": [ ▼ 1 item, 167 bytes
    { ▼ 4 properties, 157 bytes
      "all_frames": true,
      "matches": [ ▼ 1 item, 30 bytes
        "<all_urls>"
      ],
      "js": [ ▼ 1 item, 30 bytes
        "content.js"
      ],
      "run_at": "document_end"
    }
  ]
}
```

# Design Patterns

- ◇ template: abstract class provides general behaviour with an concrete unmodifiable method  $m$  whilst derived classes implements single tasks used in  $m$  (e.g., `java.awt.Component`);
- ◇ Interface: allows usage of methods from a not-well specified object abstracting from its actual implementation;
- ◇ API: a method or a service a library or a third-party system provides to the developer in order to perform a task;

# Main Ideas

Each **test**  $t \in \mathcal{T}$  involves a **stuff**  $s \in \mathcal{S}$  to test (e.g., algorithm) and an **environment**  $e \in \mathcal{E}$  where the test occurs (e.g., path finding map) Each **stuff(environment)** is a tuple of  $n(m)$  values of parameters  $p_i \in \mathcal{P}$  (e.g., number of edge to perturbate, heuristic to use), where each parameter value  $v_i$  has a specific domain  $\mathcal{D}_i \cup \{\text{nil}\}$  (NIL). All Elements in  $\mathcal{S}(\mathcal{E})$  share the same tuple structure.

The final application is composed by 2 parts:

- 1 a **library** which the user call and provide a *template design pattern* with easy APIs (provided by phd-tester);
- 2 a **user code** which implements the *template design pattern*, providing mandatory information about the test as well as actually call the program to test (pro: test can be coded with any language!).

# Template to follow

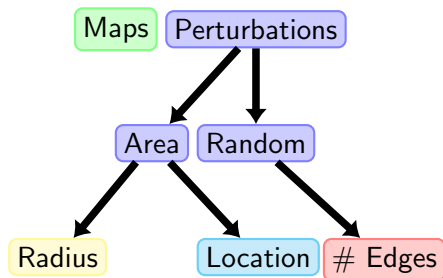
High-level steps in the *template* provided by the framework:

- 1 the user specifies all the parameter values he wants (e.g., perturbation generation policy  $\in \{\text{random}, \text{area}\}$ );
- 2 the application generates all the required tests  $\mathcal{T}^* \subseteq \mathcal{T}$ ;
- 3 the application executes all the tests  $t \in \mathcal{T}^*$ ;
- 4 the application produces generate  $n$  CSVs representing test outcome (note  $n \geq |\mathcal{T}^*|$ );

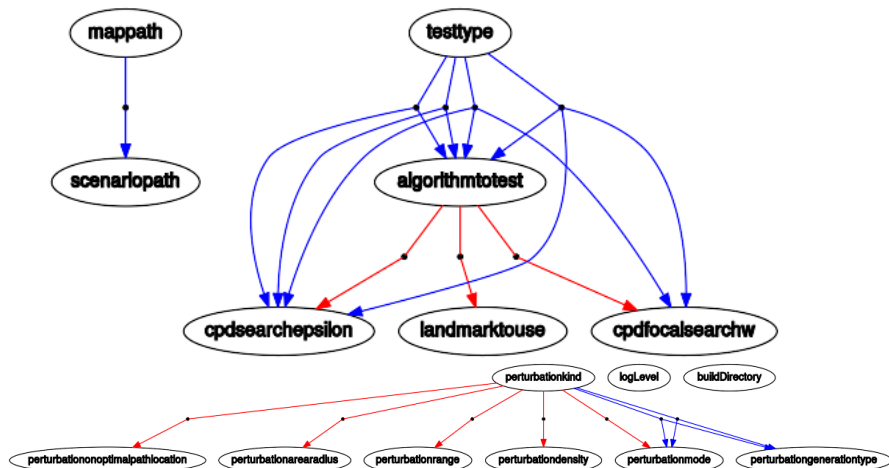
# Option Graph

The user specifies all the parameter values he wants (e.g., perturbation generation policy  $\in \{\text{random}, \text{area}\}$ );

By command line. In order to build the CLI interface, developer needs in *user code* to declare the **option graph**:  $\forall p_i \in \mathcal{P}$  say its domain  $\mathcal{D}_i$ , and what constraints it implies (e.g., if parameter `perturbationPolicy` is set to `random`, then the CLI needs to specify parameter value `edge number`).



# Example of Option graph



```
usage: main.py [-h] [--algorithmtotest_values ALGORITHMTOTEST_VALUES]
               [--cpdsearchepsilon_values CPDSEARCHEPSILON_VALUES]
               [--cpdfocalsearchw_values CPDFOCALSEARCHW_VALUES]
               [--landmarktouse_values LANDMARKTOUSE_VALUES]
               [--testtype_values TESTTYPE_VALUES]
               [--mappath_values MAPPATH_VALUES]
               [--scenariopath_values SCENARIOPATH_VALUES]
               [--perturbationkind_values PERTURBATIONKIND_VALUES]
               [--perturbationgenerationtype_values PERTURBATIONGENERATIONTYPE_VALUES]
               [--perturbationmode_values PERTURBATIONMODE_VALUES]
               [--perturbationnonoptimalpathlocation_values PERTURBATIONNONOPTIMALPATHLOCATION_VALUES]
               [--perturbationarearadius_values PERTURBATIONAREARADIUS_VALUES]
               [--perturbationrange_values PERTURBATIONRANGE_VALUES]
               [--perturbationdensity_values PERTURBATIONDENSITY_VALUES]
               [--logLevel LOGLEVEL] [--buildDirectory BUILDDIRECTORY]
```

optional arguments:

- h, --help show this help message and exit
- algorithmtotest\_values ALGORITHMTOTEST\_VALUES  
Accepted values are CPD-SEARCH CPD-FOCAL-SEARCH ALT
- cpdsearchepsilon\_values CPDSEARCHEPSILON\_VALUES
- cpdfocalsearchw\_values CPDFOCALSEARCHW\_VALUES  
Focal Search W bound
- landmarktouse\_values LANDMARKTOUSE\_VALUES
- testtype\_values TESTTYPE\_VALUES  
Accepted values are OPTIMAL BOUNDED



# Generation of $\mathcal{T}^*$

the application generates all the required tests  $\mathcal{T}^* \subseteq \mathcal{T}$ .

$\mathcal{T}^*$  represents all the tests which we are required to execute. Some cartesian products in  $\mathcal{S} \times \mathcal{E}$  are simply invalid (e.g., number of edges to perturbate set to 'nil' when the perturbation policy is *random*).

phd-tester exploits the option graph not only to generate the CLI, but also to detect parameters dependencies.

**Example:** Given a  $t \in \mathcal{T}$  if perturbation policy is *random*, require that number of edges  $\neq$  'nil': if it is true, add  $t$  in  $\mathcal{T}^*$ ).

# Execute tests

The application produces generate  $n$  CSVs representing test outcome (note  $n \geq |\mathcal{T}^*|$ );

In this step the **user code** has all the freedom it desires. Usually the developer is expected to call an external program (e.g., coded in C or in C++) that actually performs the test. As a side effect, one or more CSVs are expected to be produced somewhere (i.e., file system, ArangoDB, MySQL  $\Rightarrow$  **Data source**).

- ◇ **Problem:** how to identify which CSVs have been generated by which test  $t \in \mathcal{T}^*$ ?
- ◇ **Solution:** by creating a new naming convention of file: KS001;

# KS001 standard

## Idea

The filename is composed by basename and extension.  
basename is a **ordered** sequence of substring separated by |, and it is optionally prefixed by a label. Each substring is a unordered sequence of keyvalues string, separated by \_. Each keyvalue follows the pattern key=value.

```
filename := identifier? '|' dictionaries '|' extension?  
dictionaries := dictionary ( '|' dictionaries )*  
dictionary := name ':' keyvalues* | keyvalues  
keyvalues := keyvalue ( '_' keyvalues )*  
keyvalue := key '=' value
```

```
name := string  
identifier := [^|]+  
extension := string  
key := string  
value := string
```

```
string := ([^=_|:]|'| '='{2}|'_'{2}|'|' '{2}|':'{2})+
```

# Example of KS001

```
|mp=16room__003.map_pd=0.1_pgt=PER-SCENARIO_pk=RANDOM  
_pm=MULTIPLY_pr=[3,3]_sp=16room__003.map.scen  
|generated:type=id-over-time|.csv
```

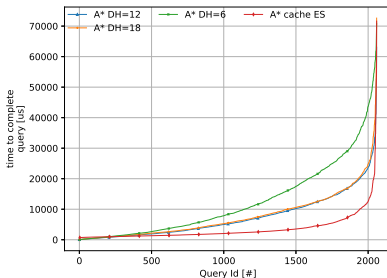
- ◇ doubling special characters if shown in key(value);
- ◇ aliasing of key(values) to avoid long names (you may reach path size limit);
- ◇ labelling of dictionary increase readability;
- ◇ phd-tester offers a set of function to query and manage KS001 compliant strings;

# Generation of interesting data (1)

The application produces generate  $n$  CSVs representing test outcome (note  $n \geq |\mathcal{T}^*|$ );

We use all the CSVs produced in the previous phase and stored in a **data source** to generate *new* CSVs representing the benchmarks (and optionally, the related graphs as well).

Performance over query id (times SORTED)  
use\_bound=False use\_anytime=False perturbation\_mode=MULTIPLY perturbation\_range=[10,10] area\_radius=[15,15] optimal\_path\_ratio=[0,95]



## Generation of interesting data (2)

`phd-tester` dynamically generate these `csv(image)`.

- ◇ each `csv(image)` is often produced by analyzing one or more outputs of tests in  $\mathcal{T}^*$ ;
- ◇ `phd-tester` provides **masks**, which acts as filter allowing the software to fetch a subset of  $\mathcal{T}^*$  to consider to produce a new `csv(image)` (e.g., produce time used by all algorithm depending on the number of edge perturbed in the map);
- ◇ The number of CSVs to analyze may be huge (maybe over 30000, each with 100 an more rows!). `phd-tester` used `pandas` along with `dask` to concurrently computes the operations  $\Rightarrow$  fast;
- ◇ `CurveChanger`: allows to further alters the functions generated by adding/removing/sorting values;

# Conclusions and Future Works

In conclusion:

- ◇ We have developed `phd-tester`, an extensible framework to combinatorially testing algorithms, parametrized both in regard of what we're testing and of the environment of the test;
- ◇ proposed a naming convention of files which is both human readable and software parsable (KS001);
- ◇ developed a mechanism to concurrently computed generated data and plot it;
- ◇ **Future works:** integrate MySQL as data source; generate automatic pdf reports of the tests done; Use a CSP to build the option graph. See <https://github.com/Koldar/phdTester/issues> for further works to be done.