Cahier des charges du projet Benchtrack

Alexis Lenoir - Xin He - Zhuangzhuang Yang

Vendredi 16 avril 2021

1 Contexte et présentation

L'origine de ce projet part du constat qu'il est difficile de comparer des implémentations différentes d'algorithmes avancés. En effet, celles-ci dépendent de nombreux facteurs comme une grande diversité dans les formats d'entrée et de sortie, des conditions d'utilisation et plus simplement du langage de programmation utilisé.

Il existe déjà plusieurs frameworks qui s'occupe de gèrer des tests sur des implémentations, comme le module *pytest* et la solution logicielle *Jenkins*. Cependant ils s'attachent à tester pour s'assurer de la validité du code, ils interviennent dans son élaboration, l'objectif du projet se place en aval. En effet, le but du projet n'est pas de développer des implémentations, mais en comparer des préexistantes. Le maître mot est donc comparaison.

L'objectif du projet est donc d'élaborer un framework python, le plus générique possible, de comparaison d'implémentation d'algorithmes complexes qui permettra d'observer et de comparer les perfomances des différentes implémentations sur différents critères. Le framework générera ainsi un *benchmark*, c'est-à-dire un banc d'essai qui permet d'évaluer la pertinence d'une implémentation à l'aide de comparaisons. Ce benchmark sera ici un site statique qui contiendra tous les résultats des comparaisons. Ce site de comparaison devra pouvoir être mis à jour automatiquement et régulièrement.

2 Besoins et contraintes

Le problème est formalisé de la manière suivante. Le but est de générer un *benchmark* pour une *infrastructure de benchmarks*. Cette infrastructure est caractérisée par un ensemble d'implémentations d'algorithmes, que l'on appelera *target* ; ainsi qu'un ensemble de tâches(*tasks*). Pour chaque *task*, chaque *target* a la possibilité de proposer un code résolvant cette *task*. Les tasks sont eux-même regroupés par thème (*theme*). Le regroupement des *targets* est assez structuré mais il sera toujours possible d'indiquer un nom par défaut. De ce formalisme découle naturelement une arborescence *infrastructure_de_test/theme/task*. Une

problématique majeure est le formalisme adopté : il doit être clair et efficace, tout en restant le plus générique possible.

L'intérêt de ce benchmark dépend fortement des critères pris en compte. Le plus trivial est la vitesse d'exécution. Cependant, il n'est pas chose aisée de la calculer.

Comme il a été dit précédemment, le *benchmark* sera impérativement un site statique, ce framework produira donc un site statique (html/css/javascript).
Afin d'optimiser le développement, sera adoptée une approche modulaire : le projet sera séparé en deux. La première partie consitera à générer les résultats dans un fichier *output_date.csv*. La deuxième partie produira le site benchmark à l'aide de *output_date.csv*.

Le projet sera développé en python.

2.1 Description générale d'une *infrastructure de benchmarks* :

La structure principale

Ce logiciel prend en entrée le chemin absolu d'une *infrastructure de test*, ce sera un répertoire. Les *themes* et les *tasks* seront représentés par des sous-répertoires. Les *targets* proposent un code résolvant une ou plusieurs *tasks*.
Dans le répertoire de l'infrastructure, il y aura deux sous-répertoires : *targets*(voir les fichiers annexes) et *tasks* qui contiendra les repertoires *themes*, qui eux-même contiendront les répertoires *tasks*. Chaque *target* qui décrit une *task* sera présent dans le sous-répertoire correspondant.

Les fichiers annexes

Après avoir abordé les données brutes, il y aussi les fichiers indispensables de configuration et de documentation. Dans le répertoire de l'*infrastructure de test* se trouvera un fichier *README.rst* qui présentera l'infrastructure. On trouvera aussi un répertoire *targets*. Pour chaque *target* sera associée un sous-répertoire */nom_infrastructure_test/targets/nom_target*. Il contiendra un fichier de configuration *config.ini* qui précisera le langage, comment l'exécuter et la mettre à jour. Comme par exemple :

```
[execution]
language = python
run =python {script} {arg}
upgrading=pip install --upgrade pyAgrum
```

Et un fichier *README.rst* qui décrira la target.

Pour chaque *task*, on trouvera dans le répertoire associée un sous-répertoire *data* qui contiendra éventuellement les données d'entrée de la tâche, tous les formats seront acceptées. Un fichier *README.rst* décrivant la *task* sera aussi présent dans le répertoire.

Format général :

```
/nom_infrastructure_test
    /README.rst
    /targets
        /target1
            /config.ini
            /README.rst
        /target2
            /config.ini
            /README.rst
    /tasks
        /theme1
            /task1
                /README.rst
                /data
                /target1.py
                /target2.r
            /task2
                /README.rst
                /data
                    data_task2.csv
                /target2.r
        /theme2
            /task3
                /README.rst
                /data
                /target1.py
                /target2.r
```

2.2 Exemple d'une *infrastructure* : *PGM*

Pour les modèles probabilistes graphiques (PGM), on voudrait comparer quelques librairies (pyAgrum, pyPGM, BNLearn) sur un ensemble de tâches parfois spécifiques traitant de l'apprentissage, de l'ingérence probabiliste et de la modélisation.

On a ainsi l'arborescence suivante :

```
/PGM
    /README.rst
    /targets
        /pyAgrum
            /config.ini
            /README.rst
        /pyPGM
            /config.ini
            /README.rst
        /BNLearn
            /config.ini
            /README.rst
```

```
/tasks
    /learning
        /parameterLearning
            /README.rst
            /data
            /pyAgrum.py
            /pyPGM.py
            /BNLearn.r
        /parameterLearningWithMissingValues
            /README.rst
            /data
            /pyAgrum.py
            /BNLearn.r
        /structuralLearning
            /README.rst
            /data
            /pyAgrum.py
            /pyPGM.py
            /BNLearn.r
    /inference
        /exactInference
            /README.rst
            /data
            /pyAgrum.py
            /pyPGM.py
        /multipleExactInference
            /README.rst
            /data
            /pyAgrum.py
            /pyPGM.py
        /approximatedInferenceGibbsSampling
            /README.rst
            /data
            /pyAgrum.py
            /pyPGM.py
            /BNLearn.r
    /modelisation
        /modelingAsia
            /README.rst
            /data
            /pyAgrum.py
            /pyPGM.py
            /BNLearn.r
        /modelingAlarm
            /README.rst
            /data
            /pyAgrum.py
            /pyPGM.py
            /BNLearn.r
        /modelingPRM
            /README.rst
            /data
            /pyAgrum.py
            /pyPGM.py
            /BNLearn.r
```

2.3 Autre exemple d'une *infrastructure* : *ConfigFichier*

On voudrait comparer différents formats de fichier de configuration (ini/json/xml) et leurs implémentations en python. Il s'agit donc de comparer la lecture de quelques exemples de fichier de configuration (ici deux) dans les différents formats. Il n'y a pas besoin d'une structuration en theme des deux seuls tasks.

```
/ConfigFichier

    /README.rst
    /targets
        /ini_demo
            /config.ini
            /README.rst
        /json_demo
            /config.ini
            /README.rst
        /xml_demo
            /config.ini
            /README.rst
    /tasks
        /default
            /charger_employe
                /README.rst
                /data
                    /employe.ini
                    /employe.json
                    /employe.xml
                /ini_demo.py
                /json_demo.py
                /xml_demo.py
            /charger_employe_coordonnees
                /README.rst
                /data
                    /employe_coordonnees.ini
                    /employe_coordonnees.json
                    /employe_coordonnees.xml
                /ini_demo.py
                /json_demo.py
                /xml_demo.py
```

2.4 Description de l'affichage des résultats : un site statique

Le logiciel sauvegardera dans un premier temps les résulats dans un fichier csv. Le format de ce fichier sera :

```
theme,task,target, args, run_time
...
```

La première ligne indique le nom de l'infrastructure. Ensuite pour les résultats de chaque target il y a une ligne : le thème et la tâche associée y sont renseignés, on précise aussi les éventuels arguments. Dans la première version, il y aura qu'un seul résultat : la vitesse d'exécution (run_time). Si le temps n'a pas pu être calculer une valeur d'erreur sera écrite (-2).

Une étape importante consistera à générer un fichier rst à partir du fichier csv. La librairie *pelican* sera alors utilisée pour produire des fichiers html.

Le site sera constitué des pages suivantes :

— Une page de résumé : un texte de présentation de l'infrastructure, une matrice task x target pour les résultats.
— Une page par task : une présentation de la tâche, le résultat par target.
— Une page par target : une présentation de la target, le résultat par task.
— Une page par target d'une task : le résultat, l'affichage du code associé

Une répertoire site_nameInfra qui contiendra le site statique sera généré stocké par défaut dans le répertoire Benchmark du framework, ce répertoire se trouvera par défaut dans le même répertoire que celui qui contient l'infrastructure.

3 Objectifs d'une première version

3.1 Fonctionnalités générales :

Un script python qui en prenant en paramètre une infrastructure (un repertoire) génère un site html(benchmark).

Les critères utilisées dans l'étude sont :
— La vitesse d'éxécution

3.2 Fonctionnement du framework :

Notre logiciel sera rassemblé dans le fichier *benchmark.py*. Il se servira de deux autres fichiers : *benchmark_results.py* et *benchmark2html.py*. On utilisera le logiciel de la manière suivante :

```
python Benchmark.py path_infrastructure_test
```

où *infrastructure_de_test* sera un répertoire respectant le formalisme qui a été définit précédemment.
Plus précisément, *Benchmark.py* produira dans un premier temps *output_date.csv* qui contiendra les résultats des éxécutions.

Il appellera ensuite *bench2site.py* avec *output_date.csv* pour produire site_nameInfra qui contiendra le site statique.

3.3 Le critère de vitesse d'exécution :

Dans un premiers temps, on calculera simplement la vitesse d'exécution. Cette opération sera répétée plusieurs fois pour augmenter la précision. On pourra aussi calculer le temps de chargement des données, qui pourra être soustrait pour affiner l'estimation.

4 Améliorations futures

Le framework pourra posséder les améliorations suivantes :

— Augmenter le nombre de critère dans l'évaluation de la comparaison
— Prendre en compte plus de langages : Java, C++, Julia ...
— Améliorer l'affichage du site statique
— Conserver en mémoire les résultats, pour les comparer entre eux, montrer ainsi l'évolution
— Personnaliser d'avantage l'interaction avec le framework en augmentant les commandes possibles avec le terminale :

```
python benchtrack.py --ouput HTML_FOLDER
python benchtrack.py --verbose
python benchtrack.py --targets-list
python benchtrack.py --tasks-list
python benchtrack.py --target-info TOTO
python benchtrack.py --task-info A
python benchtrack.py --targets-include=TOTO,TITI
python benchtrack.py --tasks-include=A,B,C
python benchtrack.py --targets-exclude=TOTO,TITI
python benchtrack.py --tasks-exclude=A,B,C
```

— Faciliter la création d'une infrastructure de test, en générant automatiquement l'arborescence à partir d'un fichier de configuration.
— Pouvoir vérifier la précision des résultats obtenus, en connaissant les résultats attendus qui seraient sauvegardés dans un répertoire *expected*

5 Délai

Une première version fonctionnelle doit être achever avant le lundi 1er mars 2021.
La dernière version de notre projet doit être réaliser avant le lundi 17 mai 2021.