Cahier des charges du projet Benchtrack

Alexis Lenoir - Xin He - Zhuangzhuang Yang

Mercredi 17 février 2021

Remarques sur cette version :

— Usage de la première personne du pluriel

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, nous ne voulons pas développer des implémentations, mais en comparer des préexistantes. Le maître mot est donc comparaison.

Notre objectif est donc d'élaborer un framework python, le plus générique possible, de comparaison d'impléméntation d'algorithme complexe qui permettra d'observer et de comparer les perfomances des différentes implémentations sur différents critères. Le framework generera 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 jours automatiquement et régulièrement.

2 Besoins et contraintes

Nous formalisons notre problème de la manière suivante. Nous souhaitons générer un *benchmark* pour une *infrastructure de test*. Cette infrastructure de tests est caractérisée par un ensemble d'implémentation d'algorithme, que l'on appelera *target* ; ainsi qu'un ensemble de problèmes(*problems*). Chaque *problem*

peut être résolu par un ou plusieurs *targets*. De ce formalisme découle naturelement une arborescence *infrastructure_de_test/problem/target*. Une problématique majeure est le formalisme adopté : il doit être clair et efficace, tout en restant le plus générique possible.

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

Comme nous avons dit précédemment, le *benchmark* sera impérativement un site statique, notre framework produira donc une archive html.

Afin d'optimiser le développement, en adoptant la stratégie diviser pour mieux régner, nous adopterons une approche modulaire : le projet sera séparé en deux. La première partie consitera à générer les résultats dans un fichier *results.format*. La deuxième partie produira le benchmark (l'archive html) à l'aide de *results.format*.

Les benchmarks seront générés à l'aide d'un script python.

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

Notre framework prend en entrée une *infrastructure de test*, ce sera un répertoire. Les *problems* seront représentés par des sous-répertoires. Les *targets* seront représentés par des fichiers de codes. Chaque *target* qui traite un *problem* sera présent dans le sous-répertoire correspondant.

Après avoir aborder les données brutes, nous allons parler des fichiers indispensables de configuration. Dans le répertoire de l'*infrastructure de test* se trouvera un fichier *config.format* qui contiendra des informations sur chaque *target*. Pour chaque *target* sera spécificiée le langage et les données de sorties si il y en a, dans ce cas on indiquera un fichier qui contiendra les résultats attendus. Tous ces fichiers seront dans le répertoire *infrastructure_de_test/expected*

Les données d'entrée seront associées aux problems, dans chaque repertoire d'un *problems* il y aura sous-repertoire *data* qui contiendra toutes les données d'entrée.

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

L'infrastructure de test *PGM* est donc un répertoire. Les *problems* sont :
- infenceSansEvidence
- apprentissage

Les *targets* sont
- pyagrum.py (en python)
- pyPGM.py (en python)
- bnlearn.r (en R)

On a l'arborescence suivante :

```
PGM/config.format
PGM/expected/pyagrum.txt
PGM/expected/pyPGM.txt
PGM/infenceSansEvidence/data/data_inf.csv
PGM/infenceSansEvidence/pyagrum.py
PGM/infenceSansEvidence/pyPGM.py
PGM/apprentissage/data
PGM/apprentissage/pyagrum.py
PGM/apprentissage/bnlearn.r
```

3 Objectifs d'une première version

3.1 Fonctionnalités générales :

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

Les critères utilisées dans l'étude sont :
- La vitesse d'éxécution
- L'exactitude des réponses

3.2 Fonctionnement du framework :

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

```
python benchmark.py infrastructure_de_test
```

où *infrastructure_de_test* sera un répertoire respectant le formalisme que l'on a définit précédemment.
Plus précisément, *benchmark.py* appellera *benchmark_results.py* avec l'infrastructe de test et récupera le fichier *results.format* généré. Il appellera ensuite *benchmark2html.py* avec *results.format* pour produire *benchmark.html*.

3.3 Le critère de vitesse d'exécution :

4 Améliorations futures

On pourra ensuite implémenter 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

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.