

Benchtrack

Alexis Lenoir - Xin He - Zhuangzhuang Yang

Projet ANDROIDE - Sorbonne Université

22 mai 2021

Sommaire

- 1 Introduction
- 2 Le format de l'infrastructure de benchmark
- 3 Une démonstration vidéo
- 4 La génération des résultats
- 5 Le site statique

Introduction

Origine du projet

Il est difficile de comparer des implémentations différentes d'algorithmes avancés.

En effet, elle dépendent de nombreux facteurs:

- Une grande diversité dans les formats d'entrée et de sortie
- Des conditions d'utilisation
- Plus simplement du langage de programmation

Présentation de Benchtrack

Le but

Notre framework permet de générer un site statique qui fera office de benchmark, on y compare différentes implémentations sur différents problèmes avec leur temps d'exécution.

Sur le développement

- Le développement est divisé en deux parties: (1) la production des résultats enregistrés dans un fichier csv, (2) la génération du site statique à l'aide de ce fichier csv et du générateur pelican
- Benchtrack est écrit en python

Le format de l'infrastructure de benchmark

Benchtrack travaille sur une infrastructure suivant un format assez structuré mais générique. C'est un répertoire.

Terminologie

- Les implémentations d'algorithmes sont appelées *targets*
- Les *targets* résolvent des tâches qui sont appelées *tasks*
- Les *tasks* sont regroupées dans des thèmes et sont appelés *themes*

Le format de l'infrastructure de benchmark

```
/nom_infrastructure_benchmark
  /README.rst
  /targets
    /target
      /README.rst
      /config.ini
  /tasks
    /theme
      /task
        /README.rst
        /config.ini
        /data
        /target_run.py
        /target_before.py
```

Une démonstration vidéo

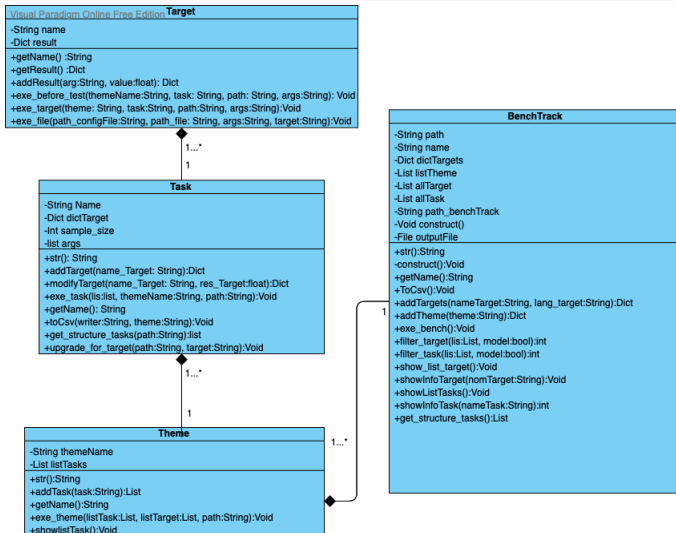
Le lien pour la vidéo de démonstration:
<https://youtu.be/EY-b0ffQfDk>

Programmation orientée objet

Les avantages:

- Gestion pratique des fonctions
 - Encapsulation des données
 - Opération d'objet
-
- Bench, Target, Theme, Task
 - tools.py: stocker d'autres méthodes

UML



Fichiers de configuration

Afin de gérer les paramètres de configuration, nous avons choisi le type de fichier INI → Plus facile à manipuler par l'utilisateur

Fichiers INI de Target

```
[execution]  
language = python  
run = python script arg
```

Fichiers INI de Task

```
[running]  
sample_size = 5  
args = 10  
display_mode = line
```

Gestion des paramètres

Paramètres généraux

a 1,b 2: deux groupe de paramètres pour une target

Paramètres avec produit cartésien

(a,b,c) * (1,2,3): Il y aura 9 groupes de paramètres

Paramètres avec itérations

(0;10;1): Il y aura 10 paramètres de 0 à 10

Calcul des résultats

On a deux étapes pour calculer les résultats:

1. Génération des objets
2. Calcul des résultats

I. Génération des Objets

Construire un objet benchTrack qui contient tous les themes, les tasks, les targets.

- Parcourir la répertoire pour créer une liste des targets
- Créer la liste des themes, des tasks, et puis ajouter les targets dans les objets tasks suivi de la liste des targets

Calcul des résultats

II. Calcul des résultats

- Parcourir et exécuter toutes les targets
- Calculer le temps d'exécution de `target_run.py` et `target_before.py` (temps de chargement)
- Calculer les moyennes des temps des targets
- Stocker les résultats avec le format csv

Fichier csv des résultats

	A	B	C	D	E
1	theme	task	target	args	run_time
2	inference	randomBN	pyAgrum	10 1	0.237978
3	inference	randomBN	pyAgrum	20 1	0.252804
4	inference	randomBN	pyAgrum	30 1	0.250955
5	inference	randomBN	pyAgrum	50 1	0.289621
6	inference	randomBN	pyAgrum	70 1	0.290224
7	inference	randomBN	pyAgrum	100 1	0.323798
8	Miscellena	BIFreading	pyAgrum	asia.bif 1	0.237922
9	Miscellena	BIFreading	pyAgrum	alarm.bif 2	0.308393
10	Miscellena	BIFreading	pyAgrum	Mildew.bif 3	18.31133
11	Miscellena	BIFreading	pyAgrum	Diabetes.bif 4	16.7662

Pour la programmation

Pendant la programmation, on a utilisé les outils:

Unittest

Unittest permet de tester l'exactitude d'un module, d'une fonction ou d'une classe

Commentaire

Pendant la programmation, on a fait des commentaire au format de Sphinx pour faire la documentation automatique.

Packaging

Faire un package et télécharger sur PyPi.

Le site statique

Pelican

- On utilise le générateur statique python pelican
- Transforme des fichiers rst en fichiers html à l'aide d'un thème
- Utilise un fichier de configuration python pour des informations générales (nom du site, url ect) et des paramètres plus techniques.

La personnalisation d'un thème

Un thème pelican contient:

- Contient un répertoire *static* (css) et un *templates* (les fichiers modèles html)
- Complète à l'aide du moteur de template Jinja avec des variables d'environnement

La génération du répertoire content

On commence par charger les résultats contenus dans le fichier csv *output_date.csv* dans la structure de donnée:

structure_run_time[task][target][arg] = run_time

Génération des fichiers rst

On génère les fichiers rst de la manière suivante:

- En utilisant les readme.rst fournis
- En complétant avec des métadonnées (auteur, date)
- Et les résultats des exécutions (plusieurs formats: tableau rst, image)
- Et parfois le code source

La personnalisation et l'exportation

La personnalisation

On peut personnaliser le site à son infrastructure en ajoutant

- Un favicon
- Un logo

L'exportation

- Le site (fichiers html et css) est exportable où on veut
- Par défaut dans le répertoire courant
- Le site est d'abord généré dans une archive temporaire pelican
- On peut le récupérer pour lancer le site en local

L'apparence du site

La structure de base

Le site est composé de trois parties:

- Une barre latérale (sidebar)
- Le corps de la page
- Un bas de page (footer)

Les pages

Les pages du site sont :

- La page d'accueil présentant l'infrastructure
- La page présentant une task
- La page présentant une target
- La page présentant une target d'une task

La page d'accueil

PGMBench - Probabilistic Graphical Models

127.0.0.1:8000/pages/probabilistic-graphical-models.html

Probabilistic Graphical Models

targets:
pyAgrum
tasks:
Eeod
reading BIF file

Probabilistic Graphical Models

Due to the lack of a common framework, it is quite difficult to compare different libraries for managing graphical models (Bayesian networks). The goal of this benchmark infrastructure is to overcome this lack.

It deals mainly (but not exclusively) with discrete Bayesian networks.

Tasks/Targets	pyAgrumTest
randomBN	
BIFreading	

Built with Benchtrack using Pelican

Figure: Page d'accueil du site

La page présentant une task/target



Figure: Page présentant une task

La page présentant une target d'une task

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/randomBN/pyAgrum.html'. The page content is divided into two main sections. On the left, a blue sidebar contains a diagram of a probabilistic graphical model (a directed acyclic graph with nodes A, C, F, P, U, M) and a list of targets and tasks. The main content area on the right has a title 'randomBN/pyAgrum' and a subtitle 'The target pyAgrum of the task randomBN'. Below this is a table showing benchmark results for different argument values. At the bottom of the main area is a code block containing Python source code for the task. The footer of the page indicates it was built with Benchtrack using Pelican.

PGMbench - randomBN/pyAgrum X

127.0.0.1:8000/randomBN/pyAgrum.html

randomBN/pyAgrum

The target `pyAgrum` of the task `randomBN`

Arg	10 1	20 1	30 1	50 1	70 1
Run_time	0.20982785224914552	0.23003029823303223	0.2534195423126221	0.25225677490234377	0.26106915473937986

Source code:

```
1 import sys
2 import pyAgrum as gum
3
4 # code to generate the random BN
5 #gen = gum.BNGenerator()
6 #for nbrnode in [10, 20, 30, 50, 70, 100]:
7 #   for arcrate in [0.1, 0.3, 0.5]:
8 #       bn=gen.generate(nbrnode, int(nbrnode * (1 + arcrate)), 3)
9 #       gum.saveBN(bn, f"data/bn{nbrnode}-{arcrate}.bif")
10
11 nbrnode = int(sys.argv[1])
12 arcrate = int(sys.argv[2])
13 filename = f"data/bn{nbrnode}-{arcrate}.bif"
14 bn = gum.loadBN(filename)
15
16 ie=gum.LazyPropagation(bn)
17 ie.makeInference()
18 for i in bn.nodes():
19     print(ie.posterior(i))
```

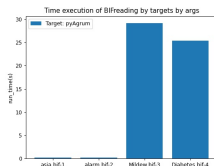
©
Built with Benchtrack using Pelican

Figure: Page task X target

L'affichage des résultats

Arg/Tasks	randomBN	BIFreading
10 1	0.2594728469848633	
20 1	0.2206024169921875	
30 1	0.24495162963867187	
50 1	0.24579596519470215	
70 1	0.2509302139282227	
100 1	0.2790544033050537	
10 3	0.1962007999420166	

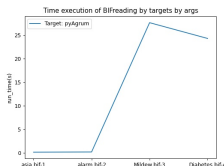
Tabular with args



Bar

Tasks	charger_employee	charger_employee_coordonnees
Run_time	0.040700531005859374	0.03851861953735351

Tabular without args



Line

Figure: Les différents affichages des résultats

Conclusion

On peut encore apporter de nombreuses améliorations:

- De nouveaux critères de comparaison (mémoire consommée par ex)
- Un meilleur site
- Compatibilité avec plus de langages de programmation

Nous avons appris:

- A résoudre les problématiques de la création d'un petit framework
- De nouveaux outils (pelican, technologies web, Unittest, packaging)
- A travailler en équipe

Fin

Merci pour votre écoute et votre attention !