Benchtrack

Alexis Lenoir - Xin He - Zhuangzhuang Yang
Encadrant: Pierre-Henri Wuillemin

Projet ANDROIDE - Sorbonne Université

22 mai 2021

Sommaire

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 implémantations 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 un 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-bOffQfDk`

Programmation orientée objet

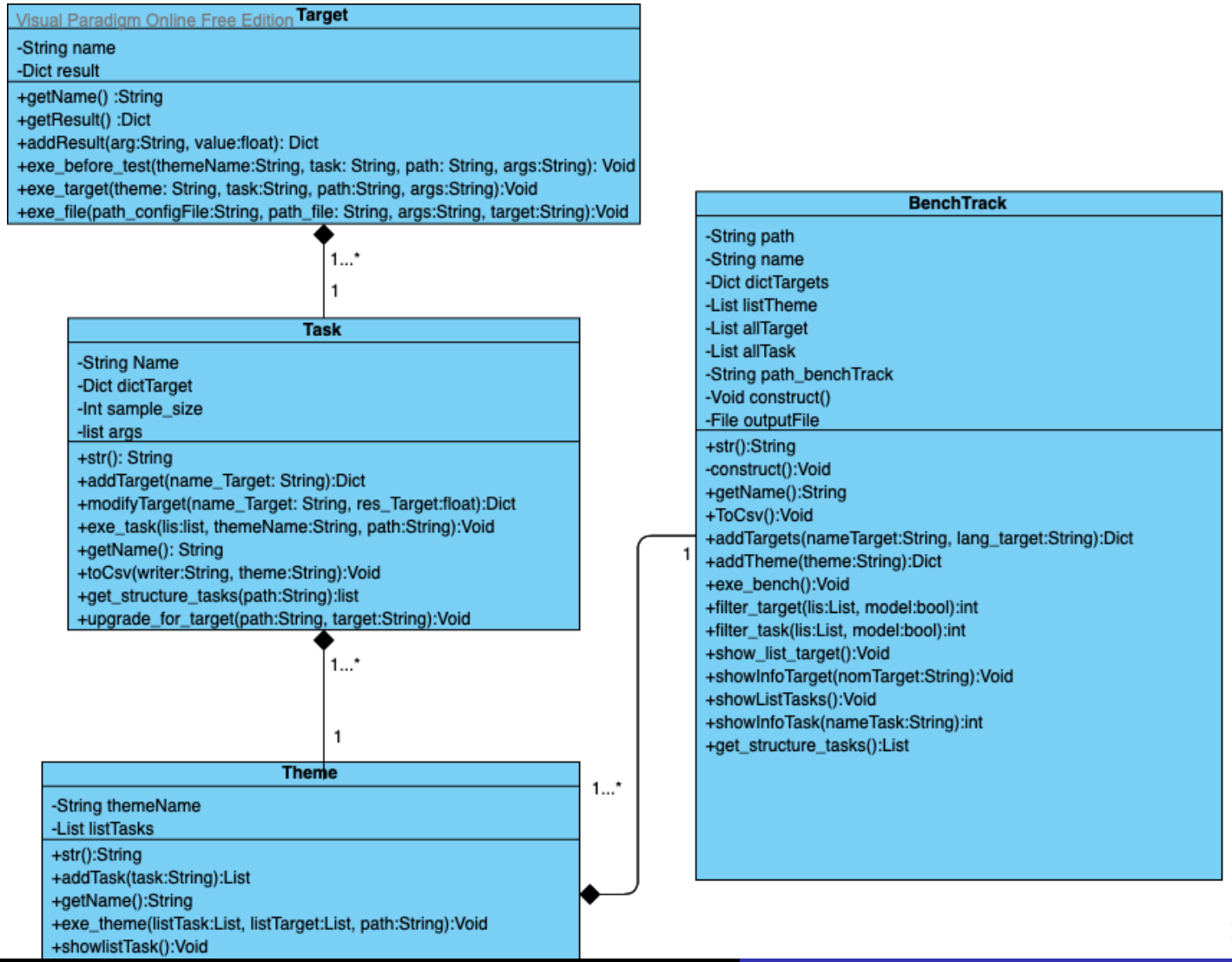
Les avantages:

- Gestion pratique des fonctions
- Encapsulation des données
- Opération d'objet

$\rightarrow$

- Bench, Target, Theme, Task
- tools.py: stocker d'autres méthodes

UML


Visual Paradigm Online Free Edition
Target
-String name
-Dict result
+getName() :String
+getResult() :Dict
+addResult(arg:String, value:float): Dict
+exe_before_test(themeName:String, task: String, path: String, args:String): Void
+exe_target(theme: String, task:String, path:String, args:String):Void
+exe_file(path_configFile:String, path_file: String, args:String, target:String):Void
1...*
1
Task
-String Name
-Dict dictTarget
-Int sample_size
-list args
+str(): String
+addTarget(name_Target: String):Dict
+modifyTarget(name_Target: String, res_Target:float):Dict
+exe_task(lis:list, themeName:String, path:String):Void
+getName(): String
+toCsv(writer:String, theme:String):Void
+get_structure_tasks(path:String):list
+upgrade_for_target(path:String, target:String):Void
1...*
1
Theme
-String themeName
-List listTasks
+str():String
+addTask(task:String):List
+getName():String
+exe_theme(listTask:List, listTarget:List, path:String):Void
+showlistTask():Void
1...*
1
BenchTrack
-String path
-String name
-Dict dictTargets
-List listTheme
-List allTarget
-List allTask
-String path_benchTrack
-Void construct()
-File outputFile
+str():String
-construct():Void
+getName():String
+ToCsv():Void
+addTargets(nameTarget:String, lang_target:String):Dict
+addTheme(theme:String):Dict
+exe_bench():Void
+filter_target(lis:List, model:bool):int
+filter_task(lis:List, model:bool):int
+show_list_target():Void
+showInfoTarget(nomTarget:String):Void
+showListTasks():Void
+showInfoTask(nameTask:String):int
+get_structure_tasks():List

Fichers de configuration

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

Fichers 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

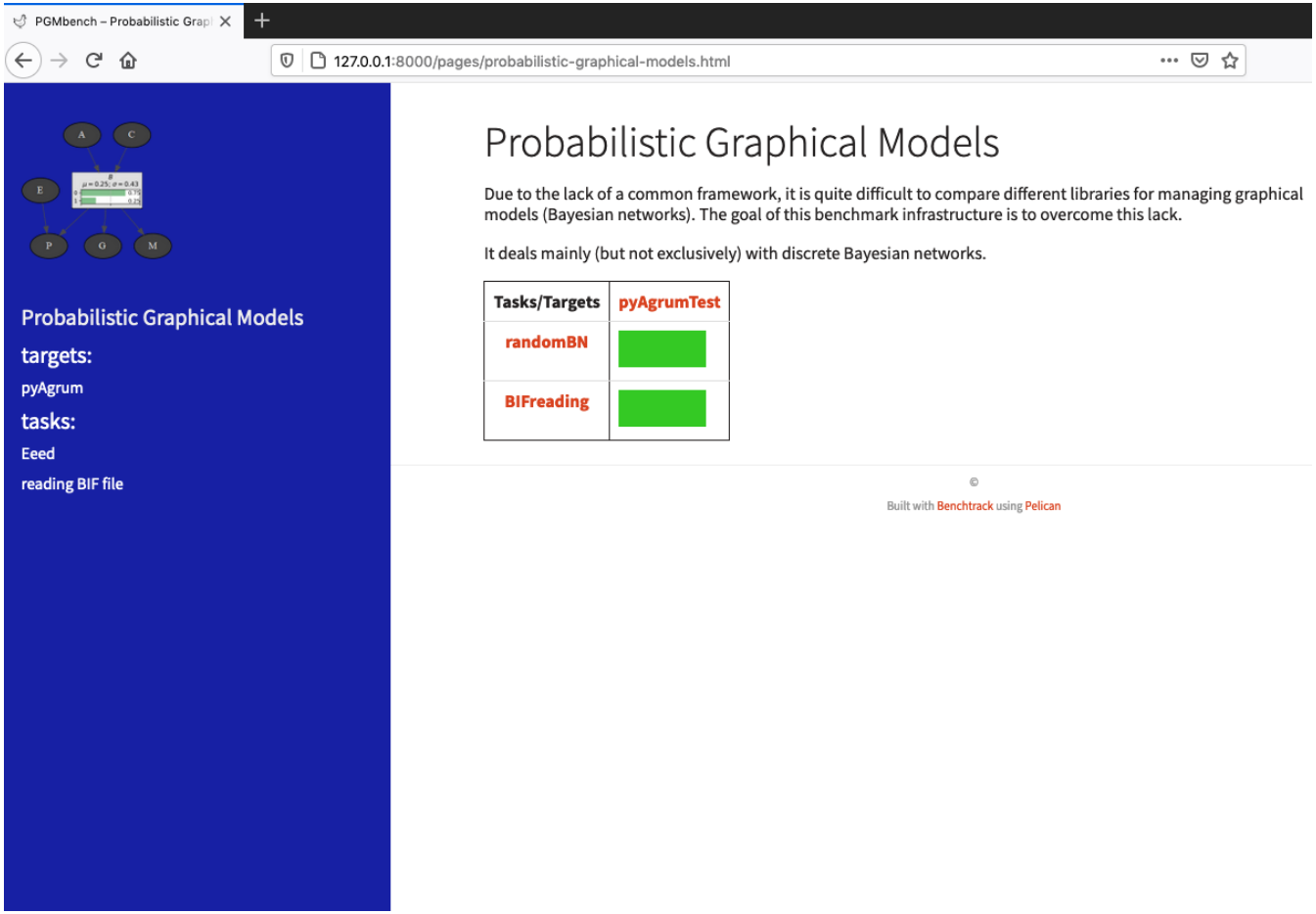


Figure: Page d'accueil du site

La page présentant une task/target

Figure: Page présentant une task

La page présentant une target d'une task

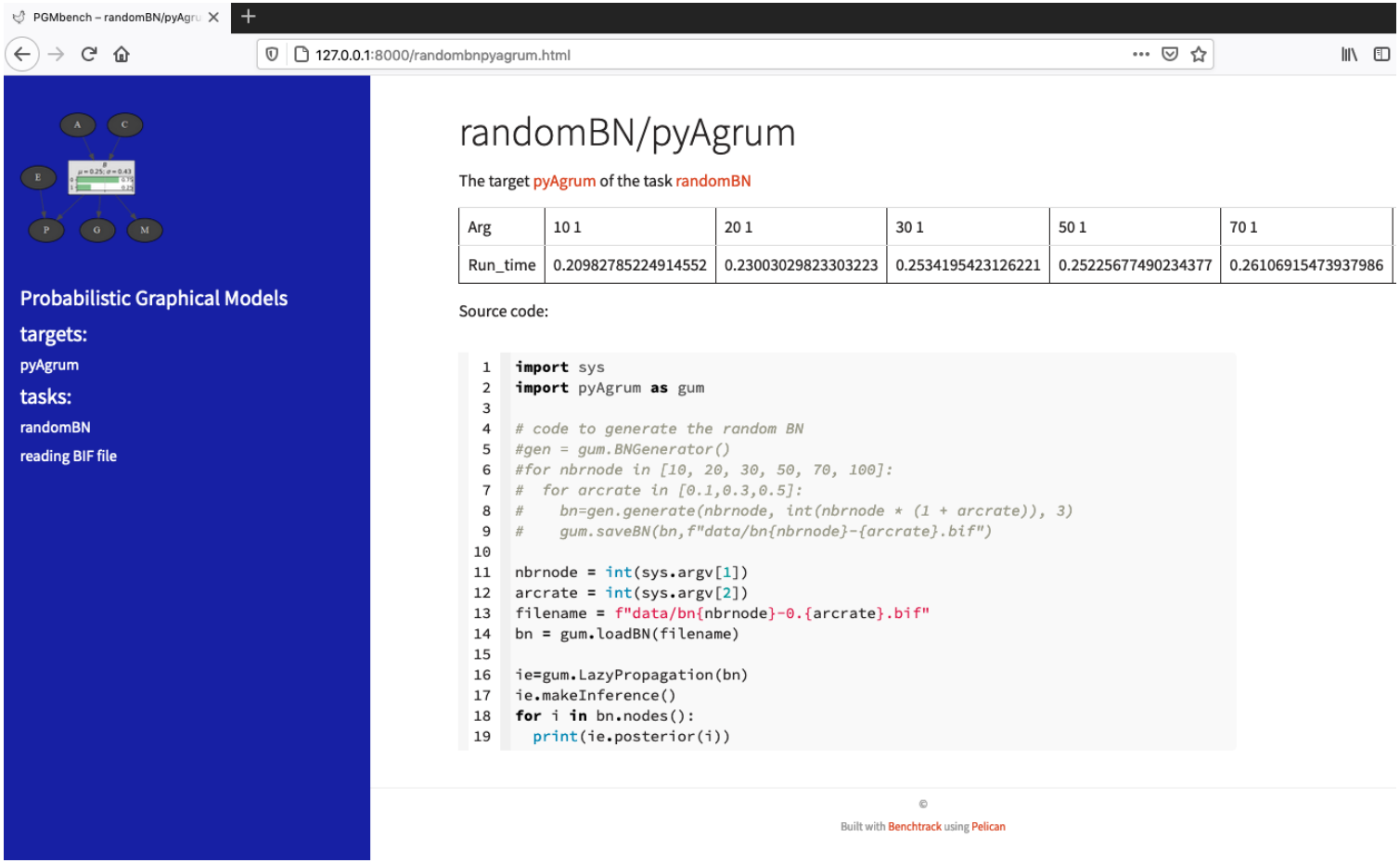


Figure: Page task X target

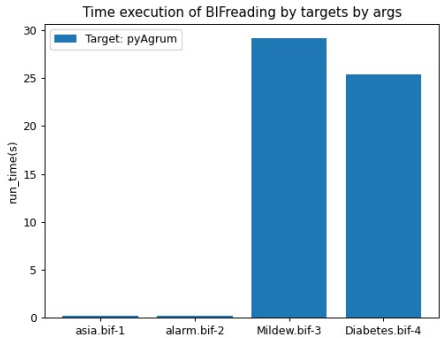
L'affichage des résultats

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	

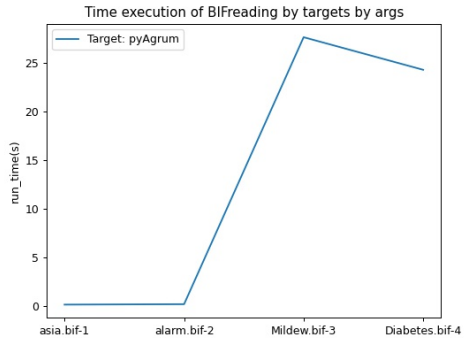
Tabular with args

Tasks	charger_employe	charger_employe_coordonnees
Run_time	0.040700531005859374	0.03851861953735351

Tabular without args



Bar



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 !