

Rapport du projet

Gestion des budgets d'une ville

1 Répartition du travail

Le travail a été réparti de la manière suivante entre les membres du trinôme :

- **Hocine Boukhemza** : Conception des classes du package `equipe`, optimisation de plusieurs éléments, réalisation complète du package de tests JUnit, rédaction du rapport, et développement de la classe `VersSacADos`.
- **Slimane Ait-Ouali** : Conception des classes des packages `SacADos` et `solveur.glouton`, documentation exhaustive de l'ensemble du code, et contribution à l'amélioration de divers composants.
- **Moufdi Bellah** : Conception des classes liées à l'algorithme HillClimbing, développement du package `main`, intégration du menu interactif , et participation à l'optimisation de plusieurs parties du projet.

Chaque membre a également participé à la relecture et à l'amélioration du code et du rapport.

Arborescence des fichiers

```
Projet
+-- gk01.dat
+--- Makefile
+--- Projet.iml
`--- src
    +--- equipe
        |   +--- Elu.java
        |   +--- EquipeMunicipale.java
        |   +--- Evaluateur.java
        |   +--- Expert.java
        |   +--- Personne.java
        |   +--- Projet.java
        |   +--- Secteur.java
        |   '--- TypeCout.java
    +--- main
        |   '--- Main.java
    +--- sacADos
        |   +--- Ojet.java
        |   +--- SacADos.java
        |   '--- VersSacADos.java
    +--- solveur.glouton
        |   +--- ComparatorMax.java
        |   +--- ComparatorMV.java
        |   +--- ComparatorSum.java
        |   +--- GloutonAjouterSolveur.java
        |   '--- GloutonRetraitSolveur.java
    +--- solveurHillclimbing
```

```

|   '--- HillClimbing.java
'--- Test
    +--- EquipeMunicipaleTest.java
    +--- GloutonRetraitSolveurTest.java
    +--- HillClimbingTest.java
    +--- SacADosTest.java
    '--- TestGloutonAjouterSolveur.java

```

2 Structure du code et justification de la hiérarchie des classes

Le code du projet a été organisé en plusieurs packages :

- **Package équipe :**
 - **Elu.java** : Représente un élu municipal, capable d'évaluer un bénéfice pour chaque projet proposé, selon ses critères spécifiques.
 - **Personne.java** : Classe mère abstraite de **Elu**, **Evaluateur** et **Expert**, regroupant les attributs communs comme le nom, le prénom et l'age.
 - **Expert.java** : Représente un expert, chargé de proposer des projets dans un secteur où il a des compétences
 - **Evaluateur.java** : Représente un évaluateur, qui évalue un cout spécifique (économique, social ou environnemental)
 - **EquipeMunicipale.java** : Gère l'ensemble des élus, experts et évaluateurs, permettant de simuler un cycle et proposer des projets.
- **Package sacADos :**
 - **Objet.java** : Représente un projet avec ses attributs.
 - **SacADos.java** : Implémente la structure du sac à dos, permettant de stocker et manipuler les projets sélectionnés.
 - **VersSacADos.java** : Convertit les données du problème en sac à dos.
- **Packages solveur.glouton et solveurHillclimbing :**
 - Les classes de solveurs (gloutons et HillClimbing) implémentent différentes stratégies pour sélectionner les projets optimaux.
 - Les comparateurs (**ComparatorMax**, **ComparatorMV**, **ComparatorSum**) permettent de comparer des objets selon une fonction.
- **Package main :**
 - **Main.java** : Permet d'effectuer une simulation aléatoire ou avec un fichier.
- **Package Test :**
 - Contient les tests unitaires pour chaque classe, garantissant qu'il n'y ait pas d'erreurs.

3 Difficultés rencontrées

L'une des principales difficultés a été l'implémentation de la classe **GloutonRetraitSolveur**. Cette classe nécessitait la gestion d'un algorithme complexe, notamment pour la stratégie de retrait d'objets puis d'ajouts. Nous avons essayé de le faire avec la structure de Map avant de trouver une solution optimale .

Pour la partie des tests, nous avons rencontré un manque d'idées sur les tests qui peuvent échouer. Nous avons finalement opté pour une série de tests basés sur des cas simples, puis des scénarios représentant la logique des algorithmes .

4 Retour critique sur l'utilisation des IA génératives

4.1 Types et nombre de prompts utilisés

Durant le projet, nous avons utilisé plusieurs types de prompts :

- Idées de tests JUnit pour vérifier la robustesse des composants.
- Aide à la rédaction du rapport en LaTeX et à la résolution de bugs de syntaxe ou de structure.
- Assistance pour la création du Makefile, notamment pour la gestion des chemins et l'intégration des bibliothèques externes comme JUnit.

Environ une vingtaine de prompts ont été utilisés, répartis selon les besoins spécifiques rencontrés à chaque étape.

4.2 Retours critiques sur le code et le texte

L'utilisation des IA génératives a permis d'accélérer certaines tâches, mais nous avons constaté :

- Des propositions utiles pour les tests JUnit, mais qui devaient être adaptées à nos classes.
- Une grande aide pour la rédaction du Makefile, notamment pour intégrer le path de JUnit qui empêchait la compilation.
- Un soutien efficace pour la rédaction du rapport en LaTeX, notamment pour corriger des erreurs de syntaxe et structurer les sections de façon plus claire.

5 Conclusion

Ce projet modélise de bout en bout un système de gestion de budgets municipaux : simulation d'une équipe (création et évaluation de projets), transformation en instance de sac à dos multidimensionnel et résolution par méthodes gloutonnes puis Hill Climbing. La structure (packages `équipe`, `sacADos`, `solveur.glouton`, `main`, `Test`) rend le code clair, et plus facile à améliorer. Les résultats obtenus montrent que ces heuristiques produisent des solutions admissibles de bonne qualité