Projet Informatique

Guo Sylvain, Rachdi Mustapha, Hoarau Romain, Paterni Thomas

Mars 2022

Table des matières

1 Organisation

En choisissant de travailler dans l'environnement Visual Studio Code, nous nous sommes naturellement tournés vers une des solutions proposées par Microsoft : Live Share. Cette extension est basée sur la technologie WebRTC, ce qui permet de communiquer des données en temps réel. Cependant, cette extension ne nous permettait pas de travailler sur un même fichier à n'importe quelle heure, car il fallait qu'un membre du groupe héberge la session.

A défaut de mettre en place un serveur hébergeur, nous avons préféré nous tourner vers d'autres méthodes telles que celle proposée par Git. Nous avons choisi *Github* comme hébergeur de code, puis nous avons décidé d'utiliser le logiciel *Gitkraken*, qui propose une interface graphique, facilitant ainsi la compréhension. Ce logiciel nous a permis de nous concentrer sur différentes parties afin de nous répartir au mieux les tâches. La présence de l'historique nous a permis de comparer et de visualiser les différences entre plusieurs versions d'un même fichier de manière simultanée par le biais d'un écran partagé (voir 1).

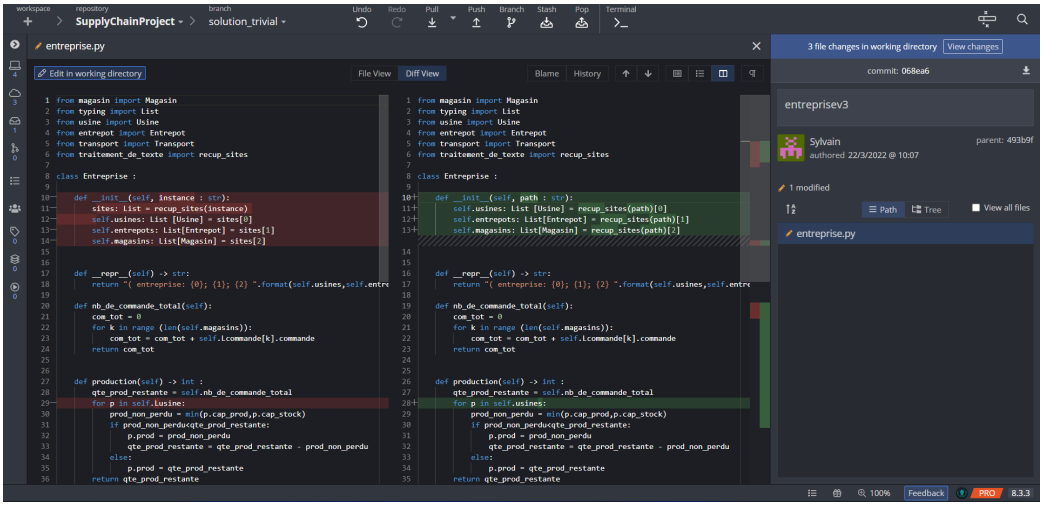

FIGURE 1 – Split view de Gitkraken

Le lien vers le dépôt du code est *disponible ici*.

2 Conception logicielle

2.1 Modèle objet

2.1.1 Classes

Nous avons fait le choix de diviser chaque entité en une classe. La classe *Site* recense les attributs communs des trois lieux (usines, entrepôts et magasins). Nous avons privilégié l'utilisation de l'héritage proposé par Python en utilisant l'initialisation avec *build*. Nous avons pu ainsi créer trois classes filles à partir de la classe mère Site.

— La classe *Usine*
— La classe *Entrepot*
— La classe *Magasin*

Par ailleurs, nous avons créé une classe *Transport* qui modélise le transport d'un site i vers un site j. Comme il ne s'agit pas d'un site, nous n'avons pas utilisé d'héritage pour la construire.

De même, nous avons construit une classe *Entreprise* qui comporte les méthodes principales de résolution du problème. Cette classe comporte une liste de sites, donc une liste d'usines, d'entrepôts et de magasins. Elle contient également les informations nécessaires à la résolution du problème, comme l'horizon de planification, le prix et toutes les données récupérées depuis les fichiers d'instances.

Enfin, la classe *Probleme* fournie joue le rôle du chef d'orchestre de toutes les autres. C'est elle qui sera appelée pour la résolution du problème.

2.1.2 Auxiliaires

— *traitement_de_texte* : Ce fichier comporte toutes les méthodes afin de réaliser tout le travail de lecture des différents fichiers *nom-*.txt*. Il est utilisé dans le programme principal.
— *generateur_de_demandes* comporte toutes les méthodes de génération de demandes.

2.2 Structure de données

Les classes *Usine*, *Entrepot* et *Magsin* sont définies comme filles de la classe *Site*. Même si peu de méthodes sont rajoutées dans les classes *Entrepot* et *Magsin* en particulier, l'utilisation de l'héritage permet toutefois d'assurer la clarté et la logique du code, notamment lors de différents appels de ces classes.

La classe *Site* comporte tous les attributs que peuvent contenir l'un des trois sites :

— La capacité de production
— Le coût de production
— La capacité de stockage
— Le coût de stockage
— Le stock initial
— Le stock final

Ainsi, pour les sites ne comprenant pas certains de ces attributs, il est nécessaire de définir ces-derniers comme égaux à 0. Cela permet d'assurer une cohérence notamment avec la lecture des fichiers (notamment *nom-site.txt*).

3 Stratégie de résolution

3.1 Stratégie

Dans un premier temps, nous avons modélisé une solution simple, la production se basant sur la somme des moyennes des demandes. Nous avons choisi de répartir cette production de manière linéaire : l'usine n°1 produit en premier dans la limite de sa capacité sans perte, puis l'usine n°2 et ainsi de suite.

Dans un second temps, nous avons voulu mettre en place une meilleure solution en comparant les coûts de l'ensemble des chemins possibles pour chaque magasin. Notre nouvelle méthode consistait à classer les coûts par ordre croissant puis à construire un quadruplet *(usine, entrepôt, magasin, coût)*. A partir de ce classement il est alors possible de déterminer la quantité envoyée entre les sites sans perte.

3.2 Algorithmes principaux

3.2.1 L'écriture

L'algorithme d'écriture se trouve dans la classe *Probleme*. Il se base sur le résultat de la méthode *sol()* de la classe *Entreprise*. Il crée un fichier *nom.sol* dans le format demandé. La méthode *sol()* est la méthode clef de la classe *Entreprise* car elle renvoie le résultat sans mise en forme. Elle se base sur plusieurs autres méthodes de la classe *Entreprise*. Nous allons nous pencher sur les principales.

3.2.2 Classe Entreprise

Comme nous l'avons évoqué précédemment, la classe *Entreprise* comporte les méthodes principales. La méthode :

- *commande()* renvoie la demande prévisionnelle moyenne du magasin passé en argument (conformément à notre permière stratégie).
- *commande_totale()* renvoie la somme des moyennes des demandes.
- *production()* se base sur notre stratégie de production. Pour chaque usine, elle calcule la quantité maxmimale qui peut être produite sans perte. Elle renvoie les quantités produites par chaque usine.
- *trans()* crée une matrice de taille $n \times n$ (n : nombre de sites). Selon la demande de chaque magasin, on transporte une certaine quantité. La logique est linéaire : le transport de la quantité produite vers la quantité demandée se fait dans l'ordre des usines et des magasins.
- *liste_com()* renvoie les commandes prévues pour chaque magasin
- *cout_prod_tot()* renvoie le coût total de production du jour
- *stock_sites()* rend compte de l'état des stocks de chaque site jour après jour
- *cout_total_stock()* renvoie le cout total du stock du jour
- *cout_trans()* renvoie le cout total de transport du jour

4 Bilan

4.1 Résultats numériques

En exécutant le fichier *checker.py* pour l'instance *A3b*, nous obtenons les résultats suivants :

```
INFO    :-------------------------------------------------
INFO    :Évaluation prévisionnelle...
INFO    :COÛT TOTAL PRÉVISIONNEL       :        4030.00€
INFO    :... DONT PRODUITS MANQUANTS   :           0.00€
INFO    :VENTES TOTALES PRÉVISIONNELLES:       37500.00€
INFO    :BILAN PRÉVISIONNEL            :       33470.00€
INFO    :-------------------------------------------------
```

FIGURE 2 – Résultat pour l'instance A3b

Les résultats obtenus sont le reflet de notre première solution : comme celle-ci réagit de manière linéaire et sans recul, nous obtenons des bilans qui peuvent parfois poser problème. De plus, notre programme met nettement

plus de temps à s'exécuter que les 10 secondes exigées. Malgré tout, nous parvenons à obtenir des résultats pour chaque instance.

4.2 Limites et améliorations possibles

Notre première solution a pu être correctement implémentée et fonctionne. En revanche, cette méthode ne permet pas d'aboutir à de bonnes combinaisons entre sites car elle est triviale.

Notre seconde idée n'a pu être implémentée dans les délais mais nous forgions en elle un espoir de gestion efficace des flux. Nous pouvions la situer à la frontière entre solution acceptable et solution optimale. Cependant, la comparaison des coûts de l'ensemble des chemins possibles pour chaque magasin aurait engendré une grande complexité algorithmique. Nous aurions dû ajouter des méthodes de tri efficaces telles que le tri fusion.