GRENOBLE INP UGA Génie industriel

Projet Informatique

UE Mathématiques et Informatique Décisionnelles

Pierre Lemaire

pierre.lemaire@grenoble-inp.fr

Grenoble INP – Génie Industriel, 1A

2021–2022

Modalités

- Concevoir et implémenter un logiciel pour résoudre un problème de génie industriel
- groupes de 4 élèves (au sein de demi-promo A+B ou C+D)
  1 rendu par groupe
- Livrables : un programme en python + documentation explicitant les choix de conception et les sources utilisées
- Critères d'évaluation
  - qualité des solutions produites par le logiciel
  - qualité de la conception logicielle et des algorithmes
  - qualité du code

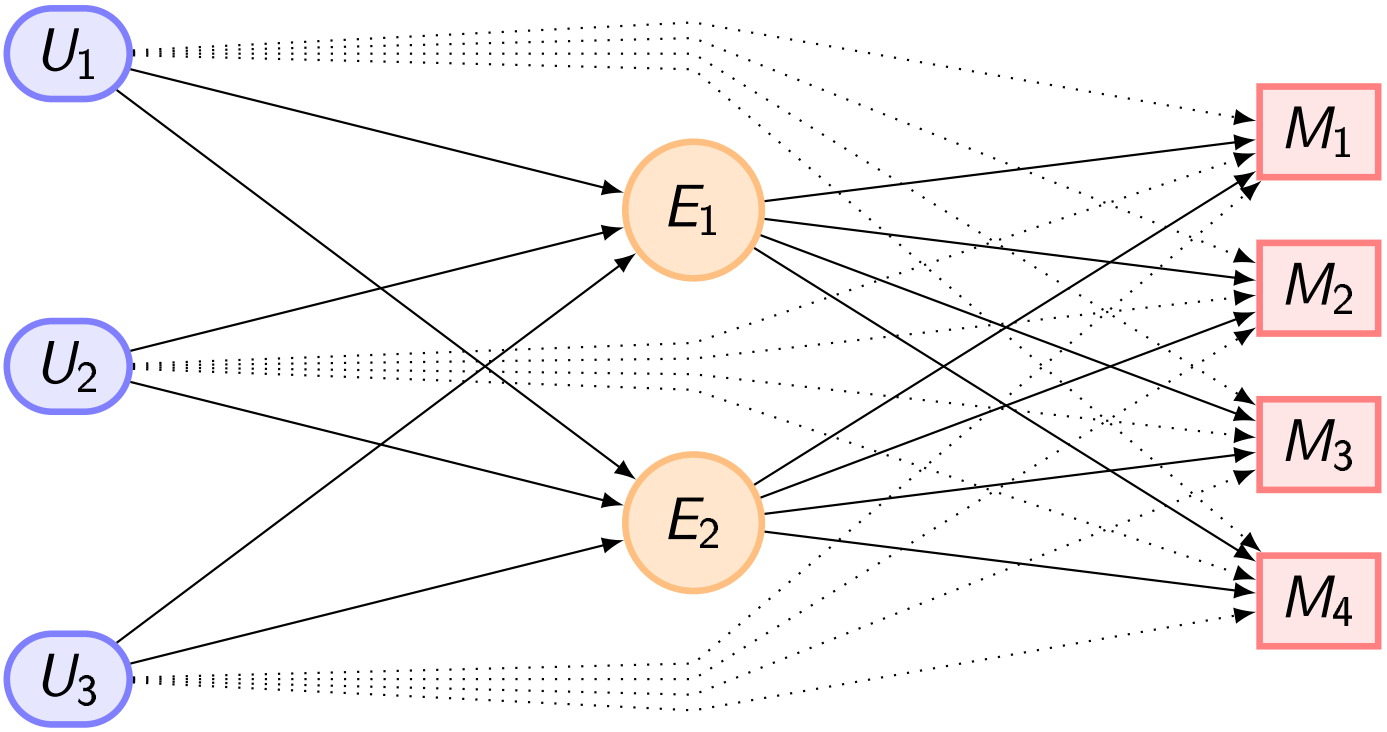
Calendrier

- 28/02 : CM ..... présentation du projet
- 28/02 : TD1 ..... compréhension du sujet
- 09/03 : TD2 ..... réflexion sur architecture et stratégie
- 11/03 ou 14/03 : TP1 ..... lecture des données
- 16/03 : TP2 ..... analyse de la demande
- 21/03 – 25/03 ..... semaine bloquée
- 25/03 ..... rendu du projet
- 11/05 : CM ..... bilan du projet

Encadrants

- Ernest Foussard
- Margaux Nattaf
- Olivier Briant / Étienne Cuisinier
- Pierre Lemaire / Fayçal Touzout

Une chaîne logistique



Les magasins vendent les produits que les usines fabriquent

Le but : planifier toutes les décisions (production des usines et transports entre sites) pour satisfaire au mieux la demande sur un horizon de planification $1..H$

Les produits

- Tous les produits sont identiques
- Tous les produits sont vendus au même prix (€/produit)

U Les usines

- Les usines fabriquent les produits
- Chaque usine u a
  - une capacité de stockage ..... $Capa\_stock_u$ (produits)
  - un coût de stockage ..... $Cout\_stock_u$ (€/produit/jour)
  - un stock initial ..... $Stock\_init_u$ (produits)
  - un stock final cible ..... $Stock\_final_u$ (produits)
  - une capacité de production ..... $Capa\_prod_u$ (produits/jour)
  - un coût de production ..... $Cout\_prod_u$ (€/produit)

E Les entrepôts

- Les entrepôts stockent les produits
- Chaque entrepôt e a
  - une capacité de stockage ..... $Capa\_stock_e$ (produits)
  - un coût de stockage ..... $Cout\_stock_e$ (€/produit/jour)
  - un stock initial ..... $Stock\_init_e$ (produits)
  - un stock final cible ..... $Stock\_final_e$ (produits)

M Les magasins

- Les magasins vendent les produits
- Chaque magasin m a
  - une capacité de stockage ..... $Capa\_stock_e$ (produits)
  - un coût de stockage ..... $Cout\_stock_e$ (€/produit/jour)
  - un stock initial ..... $Stock\_init_m$ (produits)
  - un stock final cible ..... $Stock\_final_m$ (produits)
  - un historique de demande .... $Hist\_demande_e$ (#produits sur plusieurs jours)

Les transports

- Les produits sont transportés d'un site à l'autre
- Chaque liaison entre deux sites i et j a
  - une capacité de transport ..... $Capa\_transp_{i \to j}$ (produits)
  - un coût de transport ..... $Cout\_transp_{i \to j}$ (€/produit)
- Le transport est possible seulement :
  - d'une usine à un entrepôt
  - d'un entrepôt à un magasin
  - d'une usine à un magasin (en général, plus cher)

Le stockage

- Chaque site a son propre stockage (comme défini précédemment)
- Les produits qui sont fabriqués ou qui arrivent un jour sont stockés le soir et peuvent être utilisés à partir du lendemain ; les produits qui sont expédiés ou vendus le sont le matin
  - les stocks initiaux sont disponibles le matin du jour 1
  - les stocks finaux sont les cibles pour le soir du jour H (chaque produit manquant est acheté au prix d'un produit)
- Si la capacité de stockage est dépassée, les produits en trop ne sont pas stockés mais sont définitivement perdus
- Le coût de stockage est à payer pour chaque produit présent en fin de journée

La demande

- Horizon de planification connu . . . . . H (jours)
- Chaque magasin a sa propre demande
  - la demande est connue grâce à un historique
  - la demande à satisfaire est *similaire* à celle de l'historique
    - horizon de planification : $1..H$ (inclus)
      peut-être plus ou moins long que l'historique
    - demande aléatoire de même loi (inconnue) que l'historique
    - le début de l'horizon de planification est cohérent avec le début de l'historique
  - les ventes sont exactement la demande que l'on peut satisfaire
  - la demande non satisfaite un jour est définitivement perdue

Programme (1)

```
class Probleme:
    def __init__(self, instance:str) -> None:
        '''Resout l'instance indiquee.'''

if __name__ == "__main__":
    Probleme("mini") # resout l'instance mini
```

- classe `Probleme` définie dans le fichier `probleme.py` avec le constructeur indiqué qui
  - lit les fichiers `[instance]*.txt` décrivant l'instance
  - écrit la solution dans le fichier `[instance].sol`
  - en moins de 10 secondes

Programme (2)

- votre programme doit être rendu via le VPL sur Caseine
  - RUN (fusée) exécute votre programme
  - DEBUG (insecte) exécute le checker
  - il n'y a pas d'évaluation automatique

Programme (3)

- un squelette `probleme.py` est fourni
  - il «suffit» de compléter ce fichier...
  - le `__main__` de ce squelette récupère automatiquement le nom de l'instance à résoudre dans un fichier `CONFIG` (comportement conseillé)
  - vous pouvez (devez !) ajouter d'autres méthodes, classes, fichiers, mais seul `probleme.Probleme()` sera directement appelé
  - prenez le temps de bien décomposer votre logiciel en classes et à séparer ces classes dans des fichiers de manière cohérente
  - bien structurer votre programme est indispensable à un bon travail collectif

Fichiers associés à une instance

Une instance *nom* est associée à plusieurs fichiers :

- Entrées (description de l'instance)
  - *nom*-params.txt . . . . . paramètres généraux
  - *nom*-sites.txt . . . . . description des sites
  - *nom*-transport.txt . . . . . description des transports
  - *nom*-historiques.txt . . . . . historiques des demandes
- Sorties (solution produite par votre programme)
  - *nom*.sol . . . . . décisions prises
- Indicateurs (produits par le checker)
  - *nom*.log . . . . . messages du checker
  - *nom*.kpi . . . . . états du système simulé

Format des fichiers

- Les noms des fichiers doivent être scrupuleusement respectés
- Chaque fichier est un fichier texte selon un format propre (décrit ci-après)

nom-params.txt

```
10
51.0
```

- ligne 1 : horizon de planification H (entier)
- ligne 2 : prix d'un produit (nombre)

nom-sites.txt

```
usine    :100:1.0:100:0.2:50:50
entrepot:0   :0.0:200:0.1:50:50
magasin :0   :0.0:110:0.3:50:50
```

- la ligne i décrit le site i
  - champs séparés par le caractère `:`

    espaces optionnels en début et en fin de champs [`strip()`]
  - ordre des champs :
    - type du site i (`usine` ou `entrepot` ou `magasin`)
    - capacité de production $Capa\_prod_i$ (entier) — 0 si pas usine
    - coût de production $Cout\_prod_i$ (nombre) — 0 si pas usine
    - capacité de stockage $Capa\_stock_i$ (entier)
    - coût de stockage $Cout\_stock_i$ (nombre)
    - stock initial $Stock\_init_i$ (entier)
    - stock final $Stock\_final_i$ (entier)
- l'ordre est toujours : usines puis entrepôts puis magasins

nom-transport.txt

```
0  100  10
0    0 100
0    0   0
0   .2 5.5
0    0  .3
0    0   0
```

- matrice $n \times n$ des capacités de transport (entiers)
  - la i-ème ligne donne les capacités des transports à partir du site i (l'élément ligne i colonne j est $Capa\_transp_{i \to j}$)
- puis matrice $n \times n$ des coûts de transport (nombres)
  - la i-ème ligne donne les capacités des transports à partir du site i (l'élément ligne i colonne j est $Cout\_transp_{i \to j}$)
- nombre n de sites et ordre connus grâce au fichier précédent
- valeurs séparées par un ou plusieurs espaces
- termes diagonaux et sous-diagonaux sont toujours 0

nom-historiques.txt

```
75,70,71,78,74,79,78,81,52,80
```

- la ligne i décrit la demande du $i$ème magasin
- valeurs (entiers) séparées par ,
- le nombre de valeurs est le même pour chaque ligne (chaque magasin) et est indépendant de H

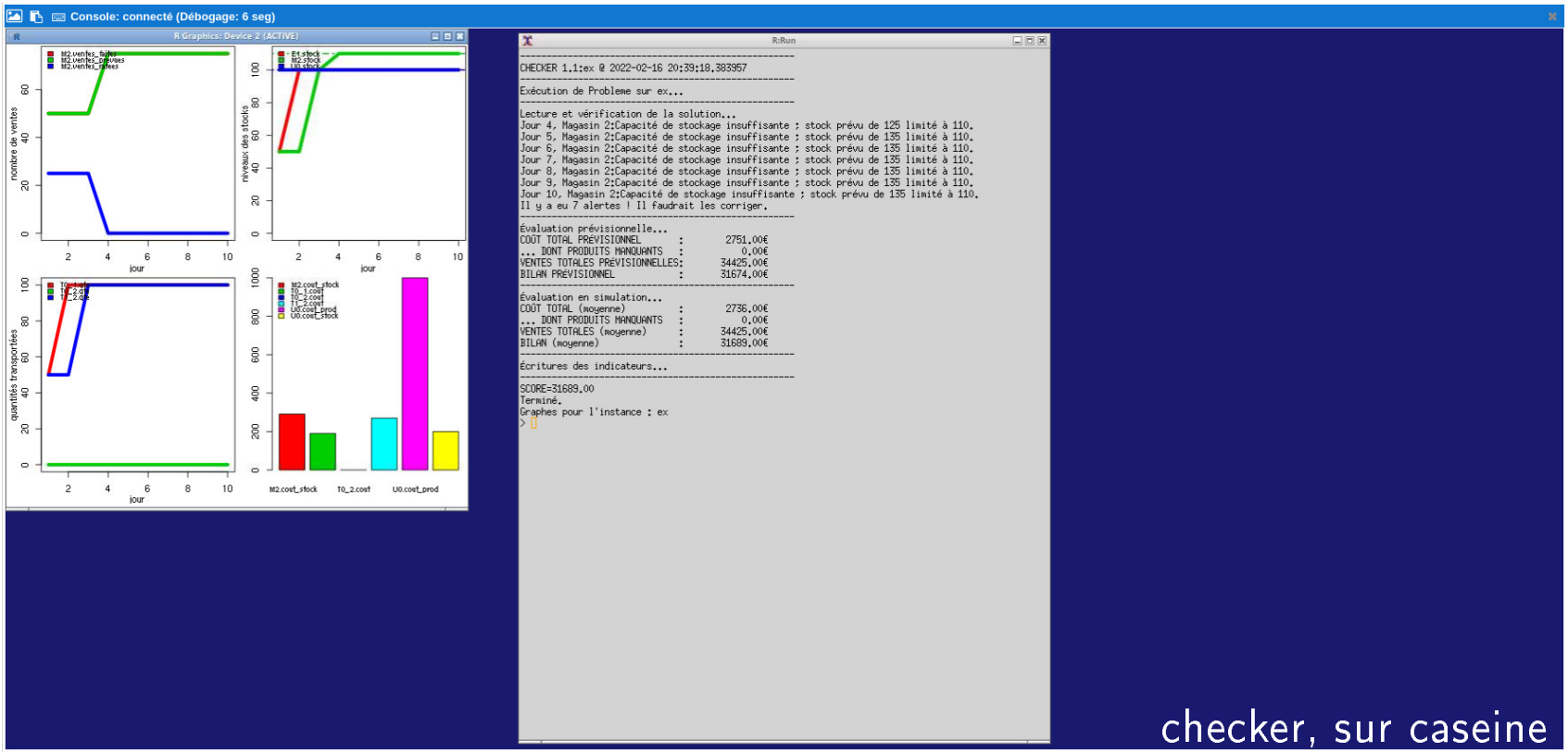
nom.sol

```
1;100;0 50 0 0 0 50 0 0 0;50;100.0;40.0;25.0
2;100;0 100 0 0 0 50 0 0 0;50;100.0;45.0;35.0
3;100;0 100 0 0 0 100 0 0 0;50;100.0;60.0;50.0
```

- la ligne *i* décrit les décisions du jour *i*
  - champs séparés par le caractère ;
  - ordre des champs :
    - numéro du jour (entier) — nécessairement dans l'ordre
    - production de chaque usine (entier) — un champ par usine
    - transports (entiers) — matrice «à plat» par ligne, éléments séparés par un espace
    - ventes prévues (entier) — un champ par magasin
    - coût total de production pour ce jour (nombre)
    - coût total de stockage pour ce jour (nombre) — prévision
    - coût total de transport pour ce jour (nombre)

Le checker (1)

Un *checker* est fourni. Ce logiciel exécute votre programme puis vérifie la solution proposée et calcule de nombreux indicateurs (KPI) associés à celle-ci qui peuvent être affichés par `kpi.R`



checker, sur caseine

Le checker (2)

- Checker sur Caseine :
  - bouton «DEBUG» (insecte)
- Checker sur sa machine :
  - récupérer `checker.zip` et le décompresser
  - intégrer `checker.py` et `checker.exe` à son projet
  - exécuter `checker.py`

    note : `checker.exe` est un exécutable particulier (compatible avec linux/unix/mac-os/windows) et ne doit être exécuté que via `checker.py`
  - exécuter `kpi.R` via `rstudio` pour avoir les graphiques
- Le comportement du checker est défini par `CONFIG`
- Les graphiques sont définis par `kpi.R`

CONFIG (1)

```
INSTANCE=mini
DEMANDE=probleme.ma_demande
NOKPI
```

Chaque ligne est une instruction de configuration pour le checker :

- `INSTANCE=...` : définit le nom de l'instance

  (`CONFIG` doit contenir une et une seule ligne de ce format)

- `NOKPI` : n'afficher que les messages, pas graphiques associés aux KPI de la solution (cette option n'a d'effet que sur Caseine)

- `DEMANDE=...` : indique le générateur de demande à utiliser

  (détails diapo suivante)

CONFIG (2)

`DEMANDE` permet de définir les générateurs de demande

- on peut indiquer une instruction `DEMANDE=...` par magasin, dans l'ordre des magasins (si aucune demande n'est indiquée pour un magasin, la valeur par défaut dépend de l'instance)
- les valeurs possibles sont :
  - `historique` pour utiliser l'historique (limité aux H premières valeurs si elles existent)
  - le nom d'une fonction définie par vos soins
    - le nom doit être complet, par exemple `probleme.ma_demande` si la fonction est définie dans le module `probleme`.
    - un générateur de demande est de type `int -> List[int]` et renvoie, pour un horizon H la demande pour tout cette horizon

kpi.R

`kpi.R` lit les indicateurs produits par le checker (*nom*`.kpi`) et trace les graphiques associés

- lancé automatique sur Caseine, à lancer à la main sur votre machine (avec `rstudio`, un simple clic sur `source` et tout se fait tout seul)
- les deux premières parties du programme (lignes 11 à 108) définissent des fonctions bien pratiques... que vous n'avez qu'à utiliser ; voir les exemples proposés à partir de la ligne 110
- ne pas hésiter à personnaliser les graphiques : trouver les bons indicateurs est un élément d'une bonne résolution

Instances

- Plusieurs instances sont fournies dans `instances.zip`
  - brèves descriptions dans README (fichier de `instances.zip`)
  - commencez par les petites instances...
  - ces instances sont «pré-installées» sur caseine, il suffit de donner leur nom dans `CONFIG`
- Vous pouvez créer vos propres instances
  - utilisez l'instance `ex` du VPL pour fournir votre propre instance

Groupes

- 2 x 7 groupes de 4 élèves par demi-promo (A+B ou C+D)
- inscriptions d'ici le 28/02 9h20 (heure de caseine)

  1 point de pénalité chaque fois que je dois intervenir

- partage du code au sein d'un groupe
  - sur `vscode` : installer l'extension `live share`
  - caseine (en faisant attention)

Livrables

- un programme en python
  - conforme aux spécifications
  - doit s'exécuter sans erreur sur n'importe quelle instance valide
- une documentation
  - format PDF, 6 pages
  - nom, prénom et bonus/malus de chaque membre
  - synthétique et correctement rédigé
  - doit aborder les éléments suivants :
    - conception logicielle (modèle objet, structure de données)
    - stratégie de résolution (algorithmes principaux, indicateurs)
    - bilan (résultats numériques, limites et améliorations possibles)
  - expliquez vos choix
  - précisez correctement les sources utilisées

Évaluation

- Analyse du problème et synthèse (6pts)
  votre analyse des différents aspects de votre logiciel (conception logicielle, stratégie de résolution...).
- Code et exécution (7 pts)
  adéquation de votre conception (modularité, structures de données...) à votre analyse du problème ; qualité du code (commentaires, noms des variables/méthodes...) ; tests
- Qualité des résultats (7 pts)
  capacités à obtenir des résultats adéquats et qualité de ces résultats

- bonus/malus individuels
  (entiers dans $[-3; +3]$, de somme nulle pour le groupe)

Méthodologie proposée

- Compréhension du problème
- Première analyse
- Conception premier modèle objet

- Maîtrise des outils
- Implémentation de votre squelette de classes
- Implémentation solution triviale

} TD/TP

- Validation solution triviale
- Répartition des tâches
- Raffinement des méthodes
- Raffinement du modèle objet

} semaine bloquée

Questions ?

- TA* : 13h30, F101 (Pierre Lemaire / Fayçal Touzout)
- TB* : 13h30, H203 (Ernest Foussard)
- TC* : 09h30, H205 (Olivier Briant / Étienne Cuisinier)
- TD* : 09h30, H204 (Margaux Nattaf)