IUT A - Département informatique - N2 - Université de Lille
2021-2022

SAÉ 2.01 & 2.02

Attribution de tutorat

L'objectif général de cette SAÉ est de développer un outil d'aide à la décision pour résoudre des problèmes trop complexes pour être résolus manuellement. Cet outil s'appuiera sur des algorithmes d'optimisation et une interface graphique utilisée pour visualiser les solutions proposées par l'algorithme d'optimisation et de guider l'algorithme dans sa recherche de solutions.

Contexte

Une formation souhaite mettre en place une plateforme de tutorat afin d'aider les étudiants volontaires mais fragiles pour différentes matières jugées essentielles. Le tutorat est destiné aux étudiants de première année en difficulté et est réalisé par des étudiants de deuxième ou troisième année. La plateforme est supervisée par des enseignants intervenant dans une ou plusieurs ressources.

Les étudiants de première année doivent s'inscrire dans les ressources pour lesquelles ils souhaitent bénéficier d'un tutorat. Les étudiants de deuxième et troisième année candidatent pour dispenser le tutorat dans une seule ressource. Chaque candidature (tant de candidats en difficulté que de tuteurs potentiels) peut être acceptée ou rejetée par un enseignant de la matière selon le niveau du tuteur-candidat dans celle-ci. Les données seront *in-fine* fournies dans un fichier qu'il faudra parcourir et récupérer.

Chaque inscription doit être validée par un enseignant intervenant dans ladite ressource. Par exemple, un enseignant participant uniquement à la ressource `R1.02` ne peut valider l'inscription d'un étudiant à un tutorat portant sur une autre ressource. Le nombre de places de tutorat par ressource étant limité, une inscription au tutorat peut être refusée si le niveau de l'étudiant est jugé suffisamment correct. Les enseignants doivent pouvoir choisir des critères de filtrage automatique pour ne pas avoir à trier à la main tous les étudiants. Ces critères doivent être génériques et paramétriques : par exemple, *filtrage sur la moyenne inférieure/supérieure à X* (où X est la valeur choisie par l'enseignant) ou *filtrage sur le nombre d'absences*, etc. Ces filtres doivent être applicables tant aux étudiants en difficultés qu'aux tuteurs potentiels, pour éviter le filtrage manuel.

Une fois les viviers de candidats et de tuteurs constitués, il faut affecter les étudiants. Selon la taille des viviers, il n'y aura pas forcément de place pour tout le monde ! L'affectation est soumise à différentes contraintes :

- les étudiants de troisième année ont la priorité pour suivre les tutorés ;
- les étudiants bénéficiant des meilleures moyennes sont privilégiés pour être tuteurs ;
- les étudiants de première année avec les résultats les plus faibles sont privilégiés ;
- les critères de motivation peuvent être pris en compte ;
- ...

Même si l'affectation est réalisée automatiquement, on souhaite avoir la possibilité d'affecter manuellement un étudiant à un tuteur spécifique. De même, la plateforme doit être en mesure de gérer les aléas : par exemple, les inscriptions tardives (aussi bien d'un tuteur que d'un étudiant fragile), le désistement d'un tuteur ou d'un étudiant... On doit alors relancer l'algorithme d'affectation sans modifier les affectations déjà réalisées. Si le nombre de tuteurs est trop faible, on peut autoriser les tuteurs de 3ème année à encadrer plusieurs étudiants de première année.

Le travail à réaliser consiste en :

- proposer une modélisation du problème sous forme d'un diagramme de classes UML,
- implémenter sous forme de boîte à outils l'algorithme d'affectation,
- développer une première version fonctionnelle du programme, permettant de tester et valider l'utilisation de la boîte à outils,
- réaliser une interface graphique qui permette de gérer les différentes opérations.

Version du sujet : 28 mars 2022

1 Organisation du travail

1.1 Gestion des équipes

Le travail est à réaliser en trinôme. Ceux-ci sont fixés par les étudiants eux-mêmes au début du projet de manière définitive, aucune modification ne sera alors possible, même en cas d'abandon d'un des membres. Les membres restants continueront le travail seuls, mais cela sera pris en compte lors de l'évaluation. Aucun trinôme inter-groupe ne sera accepté, ni aucun groupe de plus de trois étudiants.

Des dépôts `Git`, sur le `https://gitlab.univ-lille.fr` de l'université ont été mis en place par les responsables. Vous devez être affectés à un de ces projets git par un des enseignants qui encadrent les SAÉs. La visibilité du *repository* doit rester privée pour éviter les problèmes de plagiat.

1.2 Évaluations et calendrier

Le travail est décomposé en trois parties distinctes selon les ressources impliquées. Des rendus sont attendus pour chacune des parties selon un calendrier précis. Tout manquement au respect du calendrier sera sanctionné lors de l'évaluation. La progression attendue selon les ressources est illustrée Figure 1. **Dans la suite du document, les sections 2.1, 2.2 et 2.3 décrivent en détail les spécifications à respecter, de ce qu'il faut rendre exactement, ainsi qu'un calendrier prévisionnel.**

Le travail se décompose de la manière suivante :

1. Pour le calcul d'une affectation optimale grâce à la modélisation par les graphes, vous devez produire :
   - un rapport décrivant la modélisation utilisée,
   - la partie du code source qui permet de construire le modèle et exploiter le résultat,
   - une classe de tests unitaires montrant le fonctionnement de votre code.
2. la modélisation UML et l'implémentation des différents éléments du problème avec :
   - un diagramme UML (hors interface graphique détaillée)
   - une archive `jar` et le code source et les classes de tests nécessaires
   - un rapport d'analyse avec mise en évidence des mécanismes orientés objets nécessaires à la bonne structuration du code ;
3. le développement d'une interface répondant aux critères
   - un rapport, des mockups, une archive `jar` et une vidéo

Le calendrier à respecter sera le suivant :

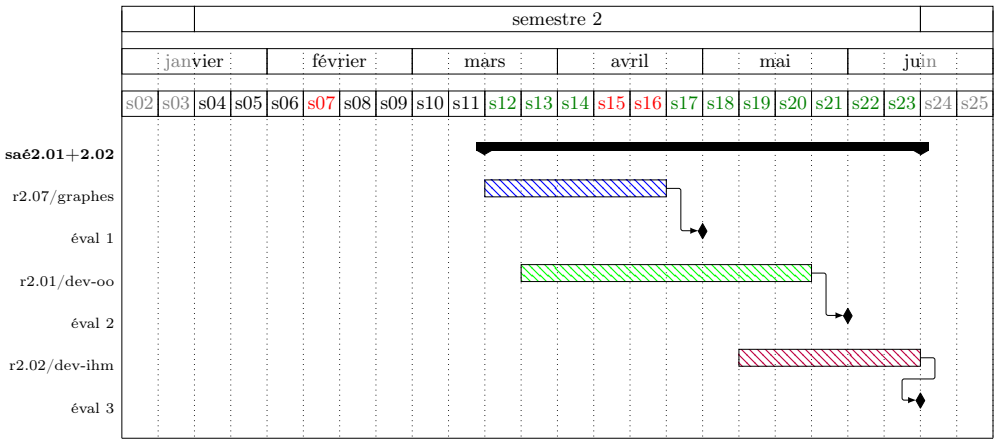

FIGURE 1 – Progression prévisionnelle par ressources.

2 Spécifications techniques à respecter

2.1 Partie I : calcul d'une affectation optimale

2.1.1 Résultats attendus

Vous devez rendre un rapport au format `pdf` ou `html` contenant :

1. Un exemple d'illustration avec 4 à 7 candidats tuteurs et 4 à 7 candidats tutorés, avec toutes les informations pertinentes les concernant (année d'études, notes, etc.). Les nombres de tuteurs et de tutorés doivent être différents. Cet exemple vous servira également comme cas de test dans votre code. L'exemple sera sélectionné pour illustrer au mieux les points suivants.
2. La description de la modélisation par les graphes du problème d'affectation, dans le cas général. C'est-à-dire, étant donné un ensemble T de candidats tuteurs et un ensemble U de candidats tutorés, vous décrirez comment définir le graphe modèle tel qu'une affectation de coût minimal dans ce graphe correspond à une affectation des tuteurs–tutorés respectant les contraintes. Puis, vous décrirez comment à partir de l'affectation dans le graphe on peut déduire les couples tuteur–tutoré.
3. Le résultat de la modélisation proposée dans le point 2. appliquée sur l'exemple d'illustration du point 1. C'est-à-dire, vous donnerez le graphe modèle correspondant à l'exemple d'illustration, le résultat du calcul d'une affectation de coût minimal dans ce graphe et les couples tuteur–tutoré. Pour plus de lisibilité, le graphe sera donné par sa **matrice d'adjacence**. Ce graphe modèle est noté **G** dans la suite.

De plus, pour chaque fonctionnalité supplémentaire demandée, vous devez illustrer sur l'exemple comment elle se traduit sur le graphe modèle.

Fixer des affectations On veut pouvoir fixer certains couples tuteur–tutoré. Fixer deux tels couples dans l'exemple d'illustration, puis donner le graphe modèle permettant de calculer l'affectation qui respecte les couples fixés. Montrer comment ce graphe diffère du graphe **G** et donner le résultat de l'algorithme d'affectation sur ce graphe, ainsi que les couples tuteur–tutoré qui en résultent.

Pondération des critères d'affectation Quelles manières de modifier les critères d'affectation proposez-vous ? Pour chaque type de modification donner le graphe modèle correspondant à l'exemple, montrer comment ce graphe diffère du graphe **G**, donner le résultat de l'algorithme d'affectation sur ce graphe, ainsi que les couples tuteur–tutoré qui en résultent.

Éviter d'affecter des candidats Dans l'exemple, fixer quelques candidats tuteurs ou tutorés qu'on voudrait exclure de l'affectation. Puis, donner le graphe modèle correspondant, montrer comment ce graphe diffère du graphe **G**, donner le résultat de l'algorithme d'affectation, ainsi que les couples tuteur–tutoré qui en résultent.

Finalement, votre rapport doit contenir les informations nécessaires pour identifier le code source implémentant ces fonctionnalités. Vous donnerez un numéro de commit git (voir ci-dessous) et le nom d'une classe de test. La classe de test doit contenir :

— La construction de l'exemple d'illustration du rapport.

— Une méthode de test pour le calcul d'affectation. Cette méthode doit construire le graphe modèle **G**, exécuter l'algorithme de calcul d'affectation à coût minimal à l'aide de la bibliothèque externe fournie et exploiter le résultat pour en déduire les couples tuteur-tutoré. Les assertions de cette méthode de test doivent montrer que les couples tuteur-tutoré sont bien ceux présents dans le rapport. **Cette méthode peut faire des appels à du code se trouvant ailleurs dans votre application.**

— Une méthode de test similaire pour chacune des fonctionnalités supplémentaires qui ont été implémentées.

Note sur les fonctionnalités supplémentaires : Vous pouvez inclure dans votre rapport des fonctionnalités supplémentaires que vous n'avez pas eu le temps d'implémenter, du moment que le travail de modélisation est fait.

IMPORTANT à propos du numéro de commit git. Il est possible que votre code concernant le calcul d'affectation évolue après la date de rendu du rapport. Dans ce cas le rapport et le code source seraient incohérents. C'est pourquoi vous devez donner le numéro de version de votre code qui correspond au moment où le rapport est écrit.

2.1.2 Ressources et calendrier

Voici les séances de la ressource R2.07 consacrées à la SAÉ :

— Semaine 12 : présentation du problème d'affectation dans les graphes (cours d'amphi).
— Semaine 13 : TP SAÉ en programmation java consacré à l'utilisation de graphes et au calcul d'une affectation.
— Semaine 14 : TD de la ressource R2.07 partiellement consacré au problème d'affectation.
— Semaine 17 : TP SAÉ encadré par les enseignantes et enseignants de la ressource R2.07.
— Semaine 18 : **Rendu du rapport**.

Nous vous fournirons également une librairie java permettant de représenter des graphes et de calculer une affectation à coût minimal, voir la section 3.

2.2 Partie II : modélisation UML et implémentation

De nombreux éléments et fonctionnalités ont été décrits en préambule. Il convient donc de réfléchir à la manière la plus efficace d'organiser le code, afin de limiter la redondance, de faciliter la maintenance ou l'évolution des fonctionnalités, etc. Pour cela, un diagramme UML prenant l'ensemble des éléments en compte doit être établi a priori, afin de ne pas devoir modifier les classes déjà écrites à chaque nouvelle fonctionnalité. Vous êtes libres de proposer de nouvelles fonctionnalités qui vous semblent pertinentes dès lors que celles requises sont implémentées.

2.2.1 Résultats attendus

Lors de l'évaluation de votre projet, différents aspects seront pris en compte comme la qualité du code produit, la couverture de votre code par les tests que vous aurez fournis, le respect du cahier des charges et des fonctionnalités demandées, la qualité des tests (les scénarios fournis, couvrent-ils bien l'ensemble des fonctionnalités demandées), la qualité du rapport et de la réflexion qui y est détaillée, l'utilisation des outils de gestion de projet (git via le nombre, la qualité et la régularité des *commit*s)...

—

Vous devez rendre l'archive `zip` d'un répertoire contenant :

— une archive `jar` de votre application fonctionnant sans interface graphique
— un rapport au format `pdf` décrivant successivement :
  — la manière de lancer et utiliser votre application (position des ressources nécessaires, commande de lancement et ses options éventuelles...)
  — un diagramme UML de vos classes, accompagné d'une réflexion sur les mécanismes objets vus en cours que vous avez mis en œuvre et leurs intérêts le cas échéant
  — une analyse quantitative/qualitative des tests que vous avez réalisés, en rapport aux notions vues en cours
— une archive `zip` du code de l'application et des tests

2.2.2 Planning

— Semaine 13 : élicitation des entités à représenter et des fonctionnalités liées, proposition d'une première version d'un diagramme UML, mise en place des outils.
— Semaine 14 : réflexion sur l'organisation à partir d'un UML pour tirer profit des mécanismes objets de Java, *refactoring* de l'application.
— Semaine 17 : développement et intégration des fonctionnalités nécessaires, développement des classes de tests.
— Semaine 18 : TP spécifique dédié à la sérialisation, pour intégration au projet :
— Semaine 19 : réflexion sur l'indépendance entre le code de l'application et du code de l'interface graphique et *refactoring* par anticipation (si nécessaire). Réflexion et analyse de la qualité du code, *refactoring* (si nécessaire).
— Semaine 20 : développement et intégration des fonctionnalités nécessaires, développement des classes de tests.

2.3 Partie III : IHM

L'interface de l'application utilisera un cahier des charges plus restreint que celui présenté au début du sujet. Elle se concentrera sur une seule matière et elle devra permettre de réaliser les opérations suivantes :

1. visualiser l'ensemble des affectations,
2. visualiser les détails sur une affectation (noms, prénoms des étudiants et moyennes),
3. visualiser les contraintes qui ne sont pas respectées (étudiant non affecté),
4. ajuster les pondérations des différents critères,
5. fixer au préalable des affectations,
6. éviter d'affecter des étudiants.

Il doit par exemple être possible de gérer la situation suivante : un étudiant de deuxième année s'est désisté et il faut trouver un nouvel étudiant qui réponde aux mieux aux critères, en conservant les étudiants déjà affectés.

2.3.1 Planning

— Semaine 19 : prototypage basse fidélité
— Semaine 20 : prototypage haute fidélité. À l'issue de la seconde semaine, vous devez avoir itéré sur les prototypes et savoir exactement comment l'interface va se présenter.
— Semaines 21 à 23 : développement en JavaFX

2.3.2 Résultats attendus

L'application sera développée en utilisant JavaFX.
Vous devrez rendre une archive **ZIP** contenant :

— un jar exécutable. Suivez les instructions disponibles dans le TP6 pour créer votre jar exécutable et utilisez JavaFX version 17.0.2 pour la compilation de votre code. Il n'est pas nécessaire d'ajouter les librairies propres au système à côté de votre jar exécutable.
— un export au format `A` de vos *mockups* placés dans un répertoire `mockups`,
— un compte rendu **au format** `pdf` contenant :
  — vos noms et prénoms,
  — une capture d'écran de l'application finale,
  — une partie sur la justification de vos choix de conception au regard des critères ergonomiques, guide de conception, etc. (cela servira notamment à évaluer la compétence 5 "identifier les besoins métiers des clients et des utilisateurs" de R2.02),
  — une partie qui détaille les contributions de chaque membre du groupe et comment vous avez réussi à exploiter au mieux les compétences de chacun (cela servira notamment à évaluer la compétence 6 "identifier ses aptitudes pour travailler dans une équipe" de R2.02),
  — toute autre information utile permettant de mettre en valeur votre travail.
— Une vidéo de présentation de votre projet, conforme aux exigences disponibles ici.

3 Matériel à disposition

Les ressources suivantes sont à votre disposition sur pour mener à bien la SAÉ :

— une liste fictive d'étudiants sous forme d'un tableau défini en extension
— les bibliothèques nécessaires à la partie "*graphe*" sous la forme d'archives `jar`

Elles sont accessibles sur `https://gitlab.univ-lille.fr/sae2.01-2.02/common-tools`.

4 Portfolio

Ces documents vous seront utiles pour la constitution de votre portfolio, vous devez les conserver :

— le sujet de la SAÉ,
— les rapports rendus pour les parties I, II et III
— la vidéo de présentation du projet.