IUT de Lille - Département informatique - N2
Université de Lille
2023-2024

SAÉ 2.01 & 2.02

Comparaison d'itinéraires de transport

1 Cadre applicatif

1.1 Description du contexte

Vous devez créer un assistant personnel de voyage, logiciel permettant de calculer, visualiser et comparer des itinéraires pour un utilisateur donné. Nous allons considérer un ensemble de villes reliées par un réseau de transport multimodal (routier, ferroviaire, aérien). Dans ce cadre, un itinéraire est une suite d'étapes reliées par un moyen de transport. Chaque étape est caractérisée par un coût exprimé selon différents critères : en euros, en temps ou en émissions polluantes. Le projet se focalise sur la comparaison des coûts des différents itinéraires en fonction des préférences de l'utilisateur et du choix des modes de transport.

Le logiciel doit permettre de répondre à des questions de ce type :

— Par quels moyens de transport peut-on aller de Toulouse à Valenciennes en minimisant les émissions de gaz à effet de serre ?

— Quels sont les 3 meilleurs itinéraires de Bordeaux à Strasbourg qui représentent un bon compromis entre le prix et la durée du voyage ?

— Quels itinéraires entre Lille et Brest minimisent l'impact environnemental tout en ne dépassant pas les 8h de trajets ?

Il n'est pas demandé du logiciel de permettre de planifier un voyage pour une date et une heure de départ données.

1.2 Données disponibles

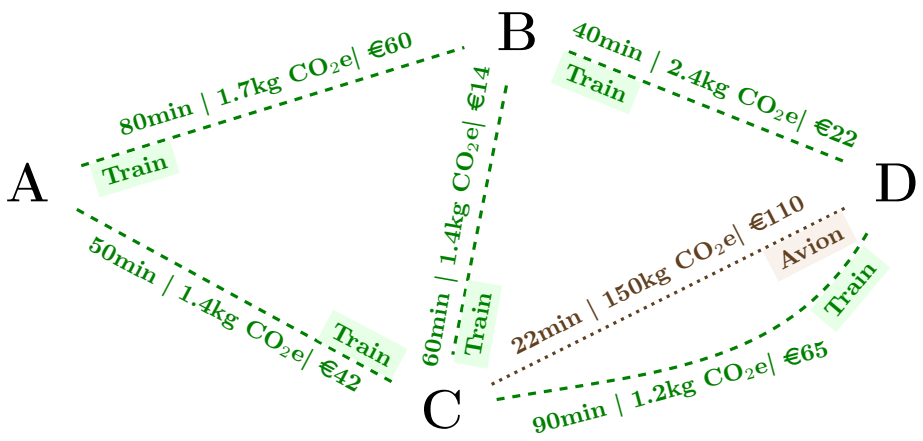
Un réseau est composé de villes et de connexions (également appelées tronçons) entre elles. À chaque connexion, nous associons un coût ayant trois composantes : la durée en minutes, le prix en euro (€) et l'émission de gaz à effet de serre en kilogrammes équivalent dioxyde de carbone (kg CO_2e). L'exemple 1 illustre un réseau composé de quatre villes et quatre lignes de différents types.

D'après le réseau illustré par l'exemple 1, il existe 6 chemins différents permettant de relier les villes `A` et `D`, chacun associé à un coût exprimé selon différents facteurs comme le montre le tableau 1.

Voyage	Coût		
	minutes	kg CO_2e	euros
A-B-D	120	4.1	82
A-C-D (par train)	140	2.6	107
A-C-D (par avion)	72	151.4	152
A-B-C-D (par train)	230	4.3	139
A-B-C-D (par avion)	162	153.1	184
A-C-B-D	150	5.2	78

TABLE 1 – Calcul des coûts des différents itinéraires possibles reliant les villes `A` et `D` selon l'aspect pris en compte à partir du réseau décrit dans l'exemple 1.

Les préférences de l'utilisateur sont exprimées au travers de son profil. La prise en compte de ses préférences est indispensable dans l'identification des voyages à lui proposer. Différentes alternatives doivent être proposées, ordonnées selon le critère qui lui tient le plus à cœur.



EXEMPLE 1 – Réseau de 4 villes (`A`,`B`,`C`,`D`) et de 6 tronçons entre ces villes. Chaque tronçon est caractérisé par une modalité de transport et un coût exprimé en durée, en impact environnemental et en euros. Par exemple, le tronçon `A-C` permet un déplacement en train durant 50 minutes, coûtant 42€ et générant des émissions polluantes à hauteur de 1.4 Kg CO_2e.

2 Versions demandées

Votre travail est décomposé en différentes fonctionnalités qu'il vous faut prendre en compte. Vous devez fournir différentes versions de votre application qui y répondent. La difficulté s'accroît avec les versions.

2.1 Version 1

Dans cette première version, vous devez calculer un itinéraire optimal entre deux villes en n'empruntant qu'un seul type de lignes. On ne prend en compte qu'une seule modalité de transport : tous les tronçons doivent être basés sur le même moyen de transport. Par exemple, sur le réseau de l'exemple 1, si on souhaite se rendre de `A` à `D` et qu'on ne s'intéresse qu'au train comme unique mode de transport, les voyages intégrant un tronçon en avion doivent être exclus : seuls 4 voyages restent éligibles. Parmi eux, on devrait choisir :

— le voyage `A-C-B-D` si l'on souhaite économiser (€78)

— le voyage `A-C-D` (par train) si on a une forte conscience écologique (2.6kg CO_2e)

— le voyage `A-B-D` si on est pressé (120 minutes)

L'utilisateur choisit au préalable quel est le critère le plus important pour lui. Il ne peut en sélectionner qu'un seul à la fois.

Les données vous sont fournies sous forme d'un tableau de chaînes de caractères (`String[]`). Ces données comporteront les informations liées à différentes modalités de transport, qu'il vous faudra filtrer le cas échéant. Les données respectent le format suivant :

`villeDépart;villeArrivée;modalitéTransport;prix(€);pollution(kgCO2e);durée(min)`

Par exemple, le tableau de chaînes représentant l'exemple 1 serait :

```
data = new String[]{
   "villeA;villeB;Train;60;1.7;80",
   "villeB;villeD;Train;22;2.4;40",
   "villeA;villeC;Train;42;1.4;50",
   "villeB;villeC;Train;14;1.4;60",
   "villeC;villeD;Avion;110;150;22",
   "villeC;villeD;Train;65;1.2;90"
};
```

Dans cette première version, l'application doit être capable de :

— vérifier la validité des données fournies : elles doivent être complètes (coûts de tous les tronçons) ;

— récupérer les informations pour reconstituer le réseau ;

— filtrer les données sur un moyen de transport spécifique ;

— déterminer si, selon la modalité choisie, il existe un voyage possible entre les villes sélectionnées par l'utilisateur ;

— afficher les meilleurs voyages possibles, ordonnés selon le critère le plus important pour l'utilisateur ;

— exclure des alternatives qui excèdent les bornes définies par l'utilisateur. Par exemple, le critère le plus important est l'impact environnemental (qu'il convient alors de minimiser), à condition que la durée du voyage n'excède pas 180 minutes. Tout voyage possible dépassant cette limite doit alors être écarté.

La recherche des meilleurs itinéraires doit se faire par un calcul de plus courts chemins dans un graphe.

2.2 Version 2

Dans cette seconde version, la multi-modalité devient possible. Les voyages peuvent inclure des changements de moyens de transport : l'utilisateur accepte de faire une première partie en train, puis la suite en bus par exemple.

Chaque changement de type de transport induit un coût supplémentaire, qui devra être intégré au calcul d'itinéraire. Ce coût de correspondance dépend de la ville où a lieu le changement de moyen de transport, ainsi que des deux types de transport concernés. Il sera donné par une liste de coûts de correspondance similaires à la Table 2.

Ville	Correspondance entre		minutes	kg CO_2e	euro
Lille	Train	Avion	130	0.1	20
Lille	Train	Bus	20	0	0
Lille	Avion	Bus	120	0.1	20
Valenciennes	Train	Bus	10	0	0

TABLE 2 – Exemple de tableau de coûts de correspondance. Par exemple, 130 est le temps en minutes nécessaire pour faire une correspondance entre le train et l'avion dans le ville de Lille.

Le critère important aux yeux de l'utilisateur reste toujours unique. Il cherchera toujours à optimiser soit la durée, soit les émissions polluantes, soit le prix, toujours dans des limites à respecter comme pour la première version. Ainsi, vous devez adapter le graphe qui modélise le problème pour permettre la prise en compte du coût de correspondance dans le calcul des meilleurs itinéraires.

Les données vous sont fournies dans un fichier CSV suivant un format similaire au Tableau 2, dont voici la représentation en CSV pour les deux premières lignes :

```
Lille;Train;Avion;130;0.1;20
Lille;Train;Bus;20;0;0
```

Dans cette version, en plus des fonctionnalités fournies par les versions précédentes (qu'il est parfois nécessaire d'adapter), l'application doit être capable de :

— signaler les problèmes (de validité de données, d'existence de chemin, d'acceptabilité de propositions...) via un mécanisme d'exception ;

— filtrer l'affichage d'une solution pour n'afficher que les points d'intérêt. En effet, inutile de citer tous les arrêts que va faire le train si l'utilisateur reste dedans. Il conviendra de n'afficher que le départ, l'arrivée et les lieux où il y a un changement dans la modalité de transport.

2.3 Version 3

Jusqu'à maintenant, l'utilisateur de votre application ne pouvait exprimer d'intérêt que pour un seul critère. Par exemple, il pouvait optimiser soit la durée, soit le prix mais pas les deux. Avec cette nouvelle version, l'utilisateur doit pouvoir exprimer plus finement ses préférences en spécifiant, de manière relative, l'importance des critères les uns par rapport aux autres. La manière d'exprimer cela est laissée libre : à vous de faciliter cela via une IHM ergonomique !

Il nous paraît également important d'avoir l'historique des voyages effectués par l'utilisateur, afin de pouvoir l'exploiter et lui rendre des informations sur sa manière de voyager (selon les centres d'intérêt qu'il a exprimé). Par exemple, pour un écologiste convaincu, on pourrait lui montrer l'évolution des émissions polluantes liées à ses voyages ; pour un utilisateur qui gère un budget, on montrerait l'évolution du prix de ses voyages.

Dans cette dernière version, l'application doit, en plus des fonctionnalités fournies par les versions précédentes, être capable de :

— adapter le modèle de calcul afin de prendre en compte non pas un critère mais une combinaison des différents critères ;

— proposer une IHM ergonomique et efficace ;

— enregistrer les voyages de l'utilisateur via un mécanisme de sérialisation ;

— exploiter l'historique pour montrer l'évolution de la façon de voyager de l'utilisateur selon les différents coûts (minutes, kg CO_2e et €).

2.4 Interface graphique

Vous devez développer une interface graphique pour la version la plus avancée de votre application ; les autres versions seront manipulées via la console et les scénarios / classes de tests que vous fournirez.

Notez qu'il est également possible de prévoir dans l'interface graphique des éléments pour les fonctionnalités non encore développées. Par exemple, votre interface pourrait permettre à l'utilisateur de spécifier l'importance relative des différents critères de coût alors que vous n'avez pas eu le temps de développer la Version 3. Dans ce cas, en cas de demande de coût composé, l'affichage des meilleurs itinéraires pourrait être toujours le même itinéraire exemple qui illustre comment vous auriez fait l'interface graphique si la Version 3 avait été terminée.

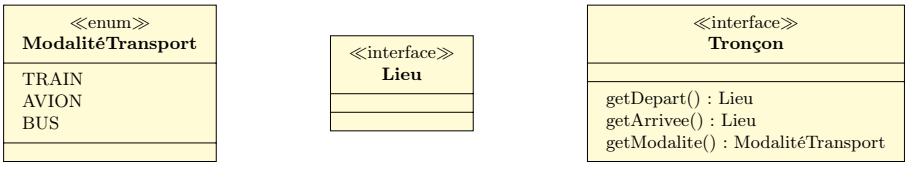
3 Détails techniques liés aux consignes

3.1 [POO]Programmation orientée objet (R2.01/R2.03)

L'utilisateur de l'application est caractérisé au minimum par un nom et un critère à optimiser dans la comparaison de voyages. Le réseau de transport est constitué de villes et de tronçons entre ces villes (via des interfaces à respecter). Chaque tronçon est caractérisé par une ville de départ, une ville d'arrivée, une modalité de transport et un coût. Un voyage sera donc constitué d'un ensemble de tronçons.

Les modalités de transport sont représentées par une énumération `ModalitéTransport` qui peut prendre les valeurs : `TRAIN`, `AVION`, `BUS`, `VÉLO`... De même, les différents types de coût sont définis via une énumération `TypeCout` qui peut prendre les valeurs `CO2`, `TEMPS`, et `PRIX`. Cette énumération représente les différentes manières d'évaluer un tronçon lors d'un voyage et peut être aussi utilisée pour représenter le critère à optimiser dans le profil utilisateur. Le coût d'un tronçon est un objet contenant une table associative (`Map`), associant à chaque type de coût (`TypeCout`) une valeur réelle (`Double`).

La modélisation de base doit **impérativement** se baser sur les deux interfaces `Lieu` et `Trançon` et l'enumération `ModalitéTransport` qui sont fournies.
Bien entendu, il est nécessaire de compléter et d'étendre cette base de conception pour y intégrer intelligemment les fonctionnalités plus avancées.



Exemple 2 – Conception partielle de base à respecter

La `Plateforme` regroupe un ensemble de `Tronçon` et un ensemble de `Lieu`. Le but de cette plateforme est de comparer des voyages entre deux villes données, optimisant le critère spécifié dans le profil de l'utilisateur. Un certain nombre de fonctionnalités seront nécessaires afin que la plateforme puisse faire face aux différentes situations pouvant survenir. La liste (non exhaustive) de ces fonctionnalités est reprise

dans le planning proposé Section 4 pour respecter la progression pédagogique des ressources et faciliter leur implémentation.

3.2 [GRAPHES]Graphes (R2.07)

Vous devez utiliser une modélisation par les graphes. Autrement dit, à partir d'une plateforme et des différentes préférences exprimées par l'utilisateur, vous construirez un graphe tel que les itinéraires préférés de l'utilisateur correspondent aux plus courts chemins dans ce graphe.

Pour faire les calculs de plus courts chemins, vous utiliserez une bibliothèque fournie sur Moodle qui permet de :

— construire un graphe valué dont les sommets sont des lieux et les arêtes représentent des tronçons,

— calculer les k plus courts chemins entre deux sommets dans ce graphe, pour un nombre entier $k \geq 0$ donné.

La modélisation choisie sera décrite dans un rapport, qui constituera la base de l'évaluation pour R2.07. Le rapport doit respecter une structure qui sera fournie sur Moodle. En particulier, vous devrez décrire le modèle utilisé pour la Version 1, et comment vous avez enrichi ce modèle pour prendre en compte les Version 2 et Version 3. Un barème avec les critères d'évaluation vous sera communiqué.

3.3 [IHM]IHM (R2.02)

L'interface de l'application a pour objectifs de permettre aux utilisateurs d'entrer les données nécessaires à l'algorithme d'optimisation, et d'en visualiser les solutions, sous une forme permettant de les rendre compréhensibles aux utilisateurs. Elle doit permettre les opérations suivantes :

1. entrer les villes de départ et d'arrivée en évitant les erreurs,
2. entrer des préférences sur les critères à privilégier (émissions carbone, temps, prix),
3. représenter les résultats fournis par l'algorithme d'optimisation sous une forme intelligible par l'utilisateur, de manière à lui permettre de choisir le voyage qui lui convient le mieux,
4. visualiser l'historique en proposant de bonnes représentations à l'utilisateur

Dans la conception de l'interface, vous proposerez des moyens d'inciter ("nudging" en anglais) les utilisateurs à adopter des comportements plus écoresponsables.

L'application sera développée en utilisant JavaFX.

4 Organisation du travail

4.1 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 intergroupe 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 dépôt (*repository*) doit rester privée pour éviter les problèmes de plagiat.

4.2 Évaluations et calendrier

Le travail est décomposé en parties distinctes selon les ressources impliquées. Des rendus sont attendus pour chacune des parties selon un calendrier précis. Tout manquement au respect du calendrier ou des consignes sera sanctionné lors de l'évaluation. Des *commits* étiquetés (munis de *tags* spécifiques) sont attendus.

La durée totale du projet est de 8 semaines. Pour vous aider dans la planification de votre travail, voici une liste des tâches à effectuer par période de deux semaines. Il est fortement recommandé de respecter ce planning qui prend en compte la progression pédagogique des ressources impliquées !

Semaines 1 (du 15/04), 2 (du 6/05) et 3 (du 13/05)

- [POO] développement des classes de base (`Voyageur`, `Plateforme`, `Lieu`...)
- [POO] vérification de la validité des données récupérées
- [POO] filtrage des données selon la modalité de transport choisie par l'utilisateur
- [GRAPHES] modélisation du problème pour la Version 1
- [GRAPHES] utilisation de la bibliothèque fournie pour calculer les voyages optimaux (Version 1)
- [POO] développement des classes de tests correspondantes, illustrant le bon fonctionnement des principales méthodes de ces classes
- [POO] réalisation d'un diagramme UML

⇒ commits étiquetés `POO-v1` et `GRAPHE-v1` à faire **avant** le 18/05.

Semaines 4 (du 20/05), 5 (du 27/05) et 6 (du 3/06)

- [POO] gestion de l'existence d'un voyage possible via un mécanisme d'exception
- [POO] import des données issues de fichiers CSV
- [GRAPHES] prise en compte des coûts de correspondance (Version 2)
- [GRAPHES] implémentation d'un scénario illustrant le bon fonctionnement du système
- [POO] affichage d'un voyage par ses points d'intérêt uniquement (lors des changements de modalités de transport)
- [POO] réalisation d'un diagramme UML
- [IHM] maquettage et prototype basse fidélité

⇒ commits étiquetés `POO-v2 GRAPHE-v2 IHM-V1` à faire **avant** le 8/06.

Semaines 7 (du 10/06) et 8 (du 17/06)

- [POO] gestion de l'historique par sérialisation binaire
- [POO] exploitation de l'historique pour afficher des informations pertinentes
- [GRAPHES][POO] expression des préférences multi-critères
- [GRAPHES][POO] implémentation d'un jeu de données et d'un scénario pour illustrer le bon fonctionnement de votre application
- [IHM] prototype haute fidélité
- [IHM] développement en JavaFX

⇒ commits étiquetés `POO-v3 GRAPHE-v3 IHM-v2`, à faire **avant** le 21/06...

5 Rendus attendus

5.1 Partie I : 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.

Vous devez fournir **un diagramme UML correspondant à chacune des versions**. Les différentes versions amèneront nécessairement des modifications : il convient donc de conserver sur votre dépôt Gitlab un diagramme pour chacune de ces versions. De plus, on doit voir cohabiter les "versions" dans le diagramme UML : il ne s'agit pas de recommencer, mais de faire évoluer !

Lors de l'évaluation de votre projet, différents aspects seront pris en compte comme la qualité du code écrit, 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 sous Moodle `un rapport au format PDF`. Ce rapport doit être cumulatif sur les différentes versions que vous rendez. Vous rédigez la section sur la première version de votre application, puis ajoutez une nouvelle section pour chaque nouvelle version (sans ne plus toucher aux sections précédentes). Chaque section doit décrire 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. Si jamais une restructuration a été nécessaire pour passer à la version suivante, vous devez justifier pourquoi (et ce qui n'avait pas été pris en compte) ;
- une analyse technique et critique de l'implémentation des fonctionnalités demandées (cf Section 3.1)
- une analyse quantitative/qualitative des tests que vous avez réalisés, en rapport aux notions vues en cours

Une archive `JAR` de votre application fonctionnant **sans interface graphique** doit être disponible à la racine de votre dépôt Gitlab.

5.2 Partie II : modélisation par les graphes

L'évaluation va se baser sur un rapport écrit et sur certaines parties du code. Le rapport doit être au format pdf et sera rendu sur Moodle.

Nous fournissons sur Moodle une trame qui contient le plan du rapport avec les parties qu'il doit contenir et le contenu de chaque partie. Ce document décrit également ce qui est attendu pour les classes de test.

5.3 Partie III : IHM

Pour la partie [IHM], 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 `HTML` ou PDF de vos *mockups* placés dans un répertoire `mockups`,
- un compte rendu **au format `pdf`** contenant :
  - vos noms et prénoms et numéro du groupe (par exemple B1),
  - lien vers le repo GitLab et référence du commit correspondant au rendu,
  - 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...(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. 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, de 2-3 minutes, conforme aux exigences disponibles ici. L'objectif est de présenter l'ensemble de votre réalisation, à destination de personnes qui n'ont aucune connaissance du projet (par exemple des recruteurs).

6 Matériel à disposition

Les ressources suivantes sont à votre disposition sur pour mener à bien la SAÉ : [inline]mettre l'url à jour. Donne-t-on ici aussi les fichiers de données prévus pour les étudiants ?

- 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`.

7 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 différentes ressources,

— la vidéo de présentation du projet.