

Rapport sur la modélisation du problème et la conception d'une interface graphique

Table des matières

- [Rapport sur la modélisation du problème et la conception d'une interface graphique](#)
 - [Table des matières](#)
 - [Modélisation mathématique du problème](#)
 - [Modélisation CPLEX du problème](#)
 - [Conception de l'interface graphique](#)
 - [Choix d'une architecture MVC](#)
 - [Librairie utilisée](#)
 - [Implémentation de l'API de CPLEX](#)
 - [Définition des Contraintes et des Variables](#)
 - [Importation des Données](#)
 - [Résolution du Problème avec CPLEX](#)
 - [Génération des Solutions et du Graphe](#)
 - [Problème rencontré](#)

Modélisation mathématique du problème

Modélisation CPLEX du problème

Conception de l'interface graphique

Choix d'une architecture MVC

Afin de garantir une conception claire et un code lisible, nous avons opté pour l'implémentation d'une architecture MVC (Modèle-Vue-Contrôleur) pour le développement de ce projet. Ce choix s'est avéré pertinent pour plusieurs raisons :

- **Séparation des préoccupations:** L'architecture MVC permet de séparer les différentes couches du code, à savoir la logique métier (Modèle), l'interface utilisateur (Vue) et la gestion des interactions (Contrôleur). Cette séparation facilite la compréhension et la maintenance du code, en favorisant une organisation logique et modulaire.
- **Clarté et lisibilité:** Le découpage en couches distinctes permet d'améliorer la clarté et la lisibilité du code. Chaque partie du code est dédiée à une responsabilité spécifique, ce qui rend le code plus facile à comprendre et à modifier.
- **Facilité de maintenance:** L'architecture MVC facilite la maintenance du code en permettant de modifier et d'étendre chaque couche indépendamment. Cela permet de corriger des bugs, d'ajouter de nouvelles fonctionnalités ou de modifier l'interface utilisateur sans affecter les autres couches du code.

En résumé, l'utilisation d'une architecture MVC a contribué à la création d'un code clair, lisible et facile à maintenir, tout en garantissant une conception logicielle robuste et flexible.

Librairie utilisée

'sys'

Le module sys est une bibliothèque standard de Python qui fournit un accès à certaines fonctionnalités spécifiques du système. En particulier, l'importation de sys est couramment utilisée pour manipuler des paramètres liés à l'exécution du script, tels que les arguments de la ligne de commande. Il offre également des fonctionnalités pour interagir avec l'environnement d'exécution, comme la gestion des chemins d'accès et la sortie standard.

'PyQt5.QtWidgets'

Le module PyQt5.QtWidgets fait partie de la bibliothèque PyQt5, qui est une liaison Python pour la bibliothèque graphique Qt. Ce module fournit des composants d'interface utilisateur (UI) et des éléments graphiques essentiels pour le développement d'applications GUI. En important PyQt5.QtWidgets, vous avez accès à des classes telles que les fenêtres, les boutons, les boîtes de dialogue, etc., facilitant ainsi la création d'interfaces utilisateur interactives et conviviales.

'PyQt5.QtGui'

Le module PyQt5.QtGui offre des fonctionnalités graphiques supplémentaires pour le développement d'interfaces utilisateur. Il inclut des éléments tels que les polices, les couleurs, les images, et d'autres outils graphiques. En important ce module, vous enrichissez les capacités graphiques de votre application PyQt5, permettant la manipulation avancée des composants visuels et des éléments esthétiques.

'PyQt5.QtCore'

Le module PyQt5.QtCore est un composant fondamental de PyQt5, fournissant des fonctionnalités de base pour le développement d'applications. Il offre des classes liées à la gestion des événements, aux signaux et aux slots, ainsi qu'à la manipulation du temps et des threads. Importer PyQt5.QtCore est crucial pour assurer le bon fonctionnement de l'architecture de l'application PyQt5, en facilitant la gestion des interactions utilisateur et des tâches concurrentes.

Implémentation de l'API de CPLEX

Pour résoudre le problème d'optimisation à l'aide de CPLEX, nous avons intégré l'API CPLEX dans notre programme Python. L'utilisation de l'API a été réalisée en respectant les étapes suivantes :

Définition des Contraintes et des Variables

Les contraintes et les variables du modèle d'optimisation ont été spécifiées en utilisant les fonctionnalités fournies par l'API DOCPlex. Les contraintes et variables sont déclarées directement dans le code Python, en utilisant les méthodes et les classes fournies par l'API pour construire le modèle de manière intuitive et efficace.

```
from docplex.mp.model import Model

# Création du modèle
model = Model('NomDuModele')

# Déclaration des variables
x = model.integer_var_dict( ... )
# Déclaration des contraintes
model.add_constraint( ... )
```

Importation des Données

Les données nécessaires pour résoudre le problème sont importées à partir d'un fichier généré par une application externe. Ces données sont ensuite traitées à l'aide d'un programme spécifique qui convertit une matrice de coordonnées en une matrice de distances. Les données résultantes sont ensuite intégrées dans le modèle CPLEX pour être utilisées lors de la résolution du problème.

```
# Importation des données à partir d'un fichier externe
donnees = importer_donnees('chemin/vers/fichier.csv')

# Traitement des données pour obtenir la matrice de distances
matrice_distances = convertir_en_matrice_distances(donnees)

# Utilisation des données dans le modèle
model.add_constraints( ... )
```

Résolution du Problème avec CPLEX

Une fois que le modèle a été construit et que les données ont été importées, nous utilisons l'API CPLEX pour résoudre le problème d'optimisation. La résolution se fait en appelant la méthode `solve()` du modèle.

```
# Résolution du modèle
solution = model.solve()

# Obtention des résultats
if solution:
    # Traitement et affichage des résultats
    afficher_resultats(solution)
else:
    print("Aucune solution trouvée.")
```

Génération des Solutions et du Graphe

Les solutions obtenues après la résolution du modèle sont exploitées pour générer les résultats finaux. L'API DOCPlex facilite la récupération des valeurs des variables, ce qui permet de créer le graphe de manière dynamique.

```
# Récupération des valeurs des variables
variables_solution = solution.get_all_values()

# Génération du graphe
generer_graphe(variables_solution)
```

En suivant cette approche, nous avons pu intégrer l'API CPLEX de manière transparente dans notre programme Python, simplifiant ainsi le processus de modélisation, de résolution et de visualisation des résultats.

Problème rencontré

- **Problème de compatibilité:** Si nous voulons lire un fichier avec l'API CPLEX en Python, nous sommes limité à une fourchette d'extensions de fichier. Qui sont le MPS, le LP et le SAV. Hors, le fichier de données que nous devons lire est un fichier .dat, qui n'est pas compatible avec l'API CPLEX.