RAPPORT PROJET RECHERCHE OPERATIONNELLE

01-05-2024

PRESENTATION DES MEMBRES

MOHAMED HABIB TRIKI

AYMEN KOCHED

IDRIS SADDI

ALA EDDINE ACHACH

1.DESCRIPTION DES PROBLEMES :

- ## Probleme de mélange

"Blending Problem Solver," enables users to solve linear optimization problems involving ingredient blending. It minimizes the total cost while satisfying user-defined constraints. Users can dynamically add ingredients and constraints, input relevant data, and obtain optimal blending solutions. The interface includes fields for ingredient names and costs, and constraints specifying minimum or maximum values. The application simplifies complex blending problems, providing clear, real-time results for decision-making.

- ## Probléme de routage des vehicules

Le VRP est un problème d'optimisation où l'objectif est de déterminer les itinéraires les plus efficaces pour une flotte de véhicules devant livrer des produits à plusieurs clients. Chaque véhicule part d'un dépôt, visite un ensemble de clients, et retourne au dépôt, tout en respectant les contraintes de capacité de charge.

2.MODELISATION MATHEMATIQUE :

- **Probleme de mélange**

Variables

- x_i : quantité de l'ingrédient i utilisée dans le mélange.
- c_i : coût unitaire de l'ingrédient i.
- p_i : quantité de protéine par unité de l'ingrédient i.
- f_i : quantité de gras par unité de l'ingrédient i.
- P_{min} : exigence minimale de protéine dans le mélange.
- F_{max} : limite maximale de gras dans le mélange.

Fonction Objectif

Minimiser le coût total du mélange :

$$\text{Minimiser } Z = \sum_{i \in \text{Ingrédients}} c_i x_i$$

Contraintes

1. **Contrainte de protéines** :
   - Le mélange final doit contenir au moins une quantité minimale de protéines.

     $$\sum_{i \in \text{Ingrédients}} p_i x_i \geq P_{min}$$
2. **Contrainte de gras** :
   - Le mélange final ne doit pas dépasser une quantité maximale de gras.

     $$\sum_{i \in \text{Ingrédients}} f_i x_i \leq F_{max}$$
3. **Contraintes de non-négativité** :
   - Les quantités de chaque ingrédient utilisées doivent être non-négatives.

     $$x_i \geq 0 \text{ pour tout } i \in \text{Ingrédients}$$

ILLUSTRATION EN PYTHON :

```python
def solve_problem(self):
    cost = {}
    for ingredient_input in self.ingredients:
        values = ingredient_input.text().split(',')
        if len(values) == 2:
            try:
                name = values[0].strip()
                cost[name] = float(values[1].strip())
            except ValueError:
                self.result_label.setText("Invalid input for one or more ingredients. Please enter numeric values.")
                return
        else:
            self.result_label.setText("Each ingredient must have a name and a cost.")
            return

    constraints = []
    for constraint_input in self.constraints:
        values = constraint_input.text().split(',')
        if len(values) == 3:
            try:
                name = values[0].strip()
                value = float(values[1].strip())
                constraint_type = values[2].strip().lower()
                if constraint_type not in ['min', 'max']:
                    raise ValueError
                constraints.append((name, value, constraint_type))
            except ValueError:
                self.result_label.setText("Invalid input for one or more constraints. Please enter valid values.")
                return
        else:
            self.result_label.setText("Each constraint must have a name, a value, and a type.")
            return
    try:
        model = Model("Blending Problem")

        amount = model.addVars(cost.keys(), name="amount")

        model.setObjective(sum(cost[i] * amount[i] for i in cost.keys()), GRB.MINIMIZE)

        for name, value, constraint_type in constraints:
            if constraint_type == 'min':
                model.addConstr(sum(amount[i] for i in cost.keys()) >= value, name)
            elif constraint_type == 'max':
                model.addConstr(sum(amount[i] for i in cost.keys()) <= value, name)

        model.optimize()

        if model.status == GRB.OPTIMAL:
            result_text = "Optimal solution found:\n"
            for ingredient in cost.keys():
                result_text += f"{ingredient}: {amount[ingredient].x:.2f} units\n"
            self.result_label.setText(result_text)
        else:
            self.result_label.setText("No optimal solution found.")
    except GurobiError as e:
        self.result_label.setText(f"Gurobi error: {e.message}")
    except AttributeError:
        self.result_label.setText("Attribute error: Please check the inputs again.")
```

• Probleme de routage des vehicules

Variables de décision

- x_{ij} : variable binaire qui prend la valeur 1 si le véhicule parcourt le trajet de i à j, et 0 sinon.
- q_i : quantité de produit transportée par le véhicule après avoir visité le client i.

Paramètres

- C_{ij} : coût ou distance entre le client i et le client j.
- Q_i : demande du client i (quantité de produits à livrer).
- V : nombre de véhicules disponibles.
- W : capacité maximale de chaque véhicule.
- N : nombre total de points de livraison, incluant le dépôt.

Fonction objectif

Minimiser le coût total de transport :

$\min Z = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C_{ij} x_{ij}$

où N inclut le dépôt et tous les clients.

Contraintes

1. **Visite chaque client exactement une fois** (chaque client doit être visité par un seul véhicule) :

$\sum_{i=0, i \neq j}^{N-1} x_{ij} = 1 \quad \text{pour tout } j = 1, \ldots, N-1$

2. **Départ du dépôt** (le nombre total de départs du dépôt est égal au nombre de véhicules) :

$\sum_{j=1}^{N-1} x_{0j} = V$

3. **Arrivée au dépôt** (assurer que tous les véhicules reviennent au dépôt) :

$\sum_{i=1}^{N-1} x_{i0} = V$

4. **Contrainte de capacité** (la capacité du véhicule ne doit pas être dépassée) :

$\sum_{j=1}^{N-1} Q_j x_{ij} \leq W \quad \text{pour tout } i = 0, \ldots, N-1$

5. **Continuité de la charge** (la charge du véhicule doit être correctement gérée) :

$q_i + Q_j \leq W + (1 - x_{ij}) \times W \quad \text{pour tous } i, j = 1, \ldots, N-1, i \neq j$

6. **Non-négativité** (assurer que les quantités transportées et les décisions de trajet sont positives ou nulles) :

$x_{ij} \in \{0, 1\} \quad \text{et} \quad q_i \geq 0 \quad \text{pour tous } i, j$

ILLUSTRATION EN PYTHON :

```python
N = len(Q) + 1  # Adding 1 for the depot
m = Model("VRP")

# Decision variables
x = m.addVars(N, N, vtype=GRB.BINARY, name="x")
q = m.addVars(N, vtype=GRB.CONTINUOUS, name="q")

# Objective: Minimize distance
m.setObjective(sum(C[i][j] * x[i, j] for i in range(N) for j in range(N)), GRB.MINIMIZE)

# Constraints
m.addConstrs((x.sum('*', j) == 1 for j in range(1, N)), "visit_once")
m.addConstr(x.sum(0, '*') == V, "leave_depot")
m.addConstrs((q[i] <= W for i in range(N)), "capacity")
m.addConstrs((q[i] + Q[j-1] <= W + (1 - x[i, j]) * W for i in range(N) for j in range(1, N)), "load_continuity")

m.optimize()
```

PREPARATION DE LA MATRICE :

```python
def setupMatrix(self):
    try:
        n = int(self.numCustomersInput.text()) + 1  # Including depot
        self.matrixTable.setRowCount(n)
        self.matrixTable.setColumnCount(n)
        self.matrixTable.setHorizontalHeaderLabels([f"Node {i}" for i in range(n)])
        self.matrixTable.setVerticalHeaderLabels([f"Node {i}" for i in range(n)])
        for i in range(n):
            for j in range(n):
                if self.matrixTable.item(i, j) is None:
                    self.matrixTable.setItem(i, j, QTableWidgetItem("0"))
    except ValueError:
        QMessageBox.warning(self, "Error", "Invalid number of customers")
```

ILLUSTRATION GRAPHIQUE :

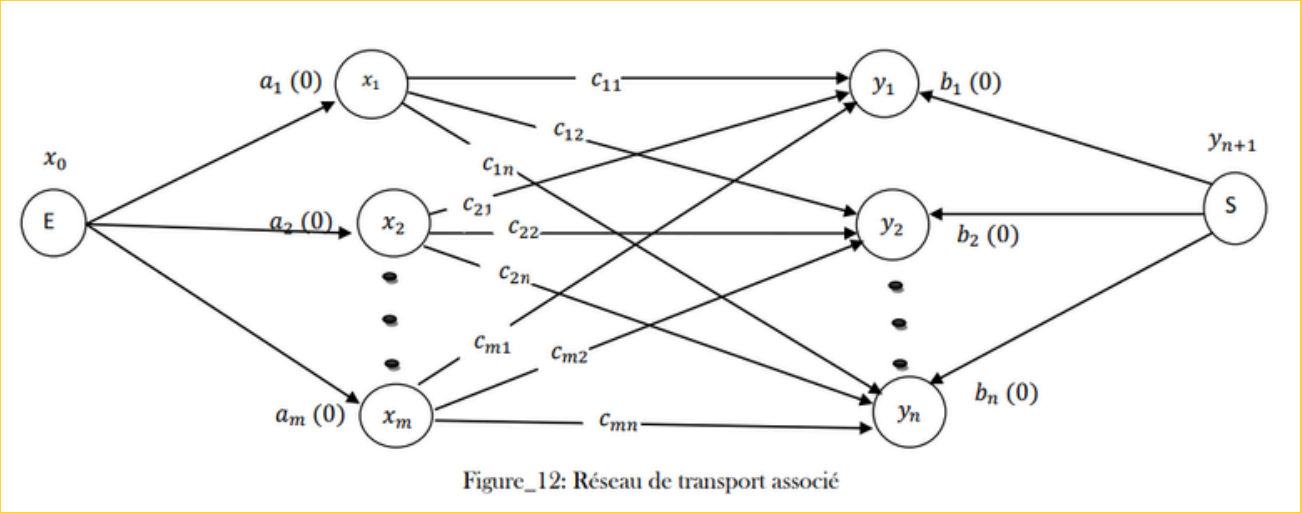


Figure_12: Réseau de transport associé

3.DESCRIPTION IHM DEVELOPPEE :

• Probleme de mélange

L'IHM est construite à l'aide de PyQt5, une bibliothèque populaire pour le développement d'applications graphiques en Python. L'interface permet aux utilisateurs de saisir des données, de lancer des calculs, et de visualiser les résultats. Voici les éléments clés de l'IHM :

Fenêtre Principale

- Taille et Titre : La fenêtre est configurée avec une dimension spécifique et un titre "Blending Problem Solver".
- Layout : Utilisation de QVBoxLayout pour organiser les widgets verticalement.

Entrée des Données

- Ingrédients : Des champs de texte (QLineEdit) pour chaque ingrédient où l'utilisateur peut entrer le nom et le coût.
- Contraintes : Des champs supplémentaires pour saisir les contraintes du mélange, incluant les quantités minimales et maximales.

Boutons d'Action

- Bouton d'Ajout d'Ingrédient : Un bouton pour ajouter dynamiquement des champs pour de nouveaux ingrédients.
- Bouton d'Ajout de Contrainte : Un bouton pour ajouter dynamiquement des champs pour de nouvelles contraintes.
- Bouton de Soumission : Un bouton pour soumettre les données saisies et lancer le calcul de la solution optimale.
- Bouton de Retour : Permet de fermer l'application.

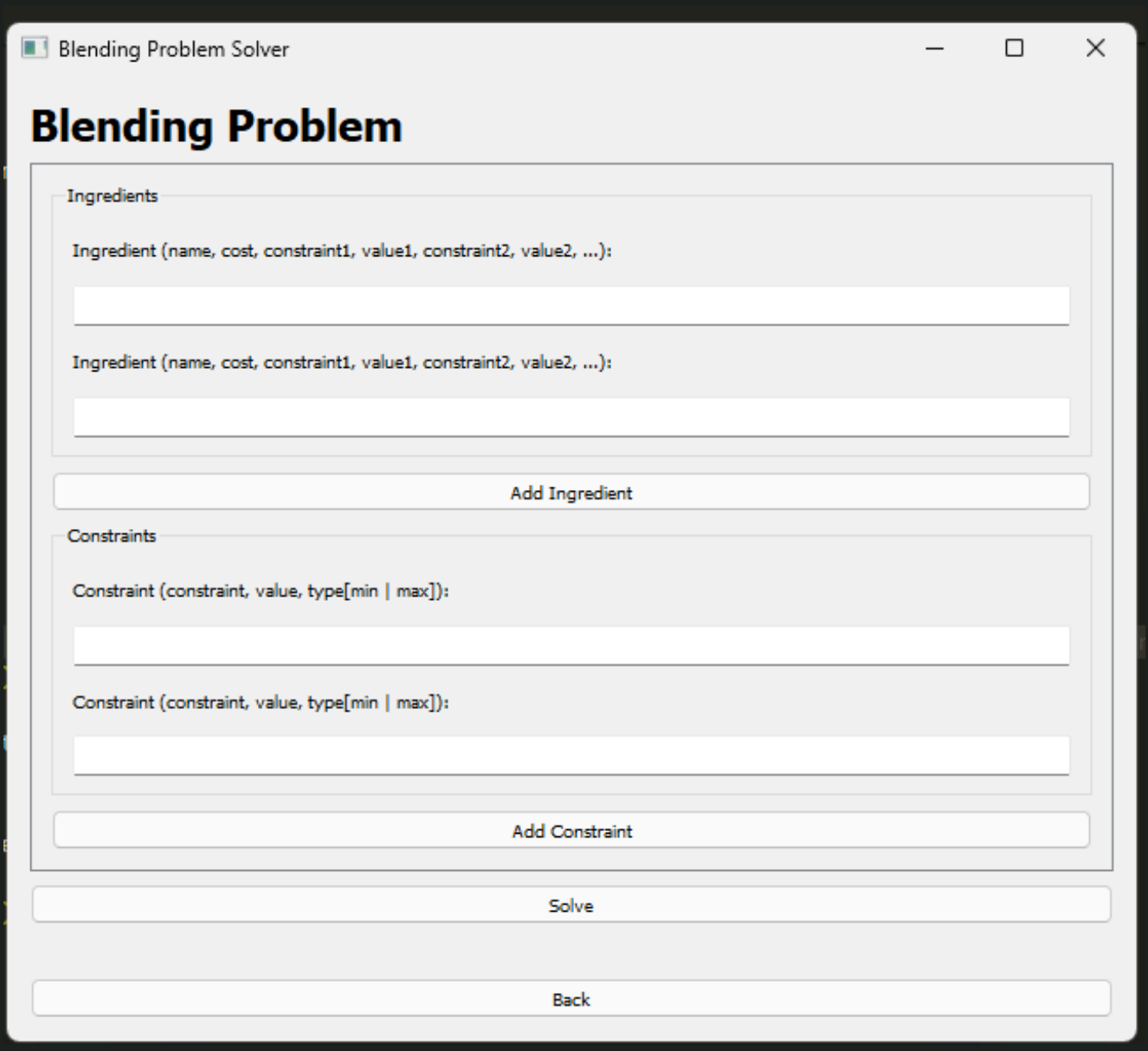
Interaction Utilisateur

- Saisie des Données : L'utilisateur remplit les champs relatifs aux ingrédients et aux contraintes. Des validations de format peuvent être effectuées pour s'assurer que les entrées sont numériques et correctes.
- Calcul de la Solution : En cliquant sur le bouton de soumission, les données sont recueillies et utilisées pour configurer et résoudre le problème de mélange à l'aide du solveur Gurobi. Si les entrées sont invalides, un message d'erreur est affiché.
- Visualisation des Résultats : Après le calcul, les résultats sont affichés dans le label de résultats. Si une solution optimale est trouvée, elle est présentée avec les quantités de chaque ingrédient ; sinon, un message indique qu'aucune solution optimale n'a été trouvée.
- Fermeture de l'Application : Le bouton de retour permet à l'utilisateur de fermer l'application à tout moment.

ILLUSTRATION EN PYTHON :

```python
def initUI(self):
    self.setGeometry(100, 100, 700, 500)  # Adjust size as needed
    self.setWindowTitle('Blending Problem Solver')

    # Main layout
    main_layout = QVBoxLayout()
    self.setLayout(main_layout)

    # Title label (Heading 1)
    title_label = QLabel('Blending Problem')
    title_label.setStyleSheet('font-size: 24px; font-weight: bold;')
    main_layout.addWidget(title_label)

    # Ingredient fields
    self.inputs = {}
    for ingredient in ['Corn', 'Wheat', 'Soy']:
        label = QLabel(f'{ingredient} (cost, protein, fat):')
        line_edit = QLineEdit()
        main_layout.addWidget(label)
        main_layout.addWidget(line_edit)
        self.inputs[ingredient] = line_edit

    # Protein and fat requirements
    self.min_protein_input = QLineEdit()
    self.max_fat_input = QLineEdit()
    main_layout.addWidget(QLabel('Minimum Protein:'))
    main_layout.addWidget(self.min_protein_input)
    main_layout.addWidget(QLabel('Maximum Fat:'))
    main_layout.addWidget(self.max_fat_input)

    # Submit button
    submit_btn = QPushButton('Solve')
    submit_btn.clicked.connect(self.solve_problem)
    main_layout.addWidget(submit_btn)

    # Result display label
    self.result_label = QLabel('')
    main_layout.addWidget(self.result_label)

    # Back button
    back_btn = QPushButton('Back')
    back_btn.clicked.connect(self.close)  # Close the current window
    main_layout.addWidget(back_btn)
```

ILLUSTRATION :



• Probleme de routage des vehicules

L'IHM est conçue avec PyQt5, qui offre un cadre robuste pour développer des applications de bureau interactives. L'application implémente une interface graphique pour saisir les données du problème VRP, exécuter le modèle d'optimisation, et afficher les résultats.

Fenêtre Principale

- Titre : "Vehicle Routing Problem Solver" qui identifie clairement la fonction de l'application.
- Configuration de la Fenêtre : La fenêtre est dimensionnée pour accueillir tous les widgets (éléments d'interface) nécessaires à l'interaction utilisateur.

Layout et Widgets

- Disposition Verticale (QVBoxLayout) : Utilisée pour organiser les widgets verticalement dans la fenêtre principale.
- En-tête (QLabel) : Affiche "Vehicle Routing Problem" avec une mise en forme pour attirer l'attention sur le problème traité.
- Formulaire d'Entrée (QFormLayout) : Permet de saisir le nombre de véhicules, la capacité de chaque véhicule, le nombre de clients, et les demandes spécifiques de chaque client. Il y a aussi un espace pour entrer ou modifier une matrice de distances ou de coûts.
- Table de Matrice (QTableWidget) : Pour visualiser et modifier les distances ou les coûts entre les clients et le dépôt.

Boutons d'Action

- Bouton de Résolution (QPushButton) : Permet de lancer le calcul du routage basé sur les données saisies.
- Bouton de Retour (QPushButton) : Permet de fermer l'application.

Interaction Utilisateur

Saisie des Données

- Entrées Numériques : L'utilisateur remplit les champs pour les véhicules, la capacité, les clients, leurs demandes, et la matrice de coûts. Des validations sont intégrées pour s'assurer que les entrées sont correctes et complètes.

Calcul de la Solution

- Exécution du Modèle : En cliquant sur le bouton "Solve VRP", les données sont collectées et le modèle d'optimisation est exécuté. Le solveur Gurobi est utilisé pour trouver une solution au problème de VRP.
- Gestion des Erreurs : Si les données sont incomplètes ou incorrectes, un message d'erreur est affiché pour guider l'utilisateur vers la correction nécessaire.

Fermeture de l'Application

- Fermeture Facile : Le bouton de retour permet de fermer l'application, offrant un moyen simple et intuitif de terminer l'interaction.

ILLUSTRATION EN PYTHON :

```python
def initUI(self):
    self.setWindowTitle('Vehicle Routing Problem Solver')
    layout = QVBoxLayout()


    # H1 styled heading
    heading = QLabel('Vehicle Routing Problem')
    heading.setFont(QFont('Arial', 24, QFont.Bold))  # Setting font to Arial, 24pt, bold
    layout.addWidget(heading)

    # Form layout for inputs
    formLayout = QFormLayout()
    self.numVehiclesInput = QLineEdit()
    self.capacityInput = QLineEdit()
    self.numCustomersInput = QLineEdit()
    self.demandsInput = QLineEdit()

    self.numCustomersInput.editingFinished.connect(self.setupMatrix)

    formLayout.addRow(QLabel("Number of Vehicles:"), self.numVehiclesInput)
    formLayout.addRow(QLabel("Vehicle Capacity:"), self.capacityInput)
    formLayout.addRow(QLabel("Number of Customers:"), self.numCustomersInput)
    formLayout.addRow(QLabel("Demands (comma-separated):"), self.demandsInput)

    # Table for distance matrix
    self.matrixTable = QTableWidget()
    self.matrixTable.setRowCount(1)  # Default size
    self.matrixTable.setColumnCount(1)
    layout.addLayout(formLayout)
    layout.addWidget(self.matrixTable)

    # Buttons
    self.solveButton = QPushButton('Solve VRP')
    self.solveButton.clicked.connect(self.solveProblem)
    layout.addWidget(self.solveButton)

    #Back button
    back_btn = QPushButton('Back')
    back_btn.clicked.connect(self.close)  # Close the current window
    layout.addWidget(back_btn)

    self.setLayout(layout)
    self.setGeometry(200, 200, 800, 600)
```

ILLUSTRATION :

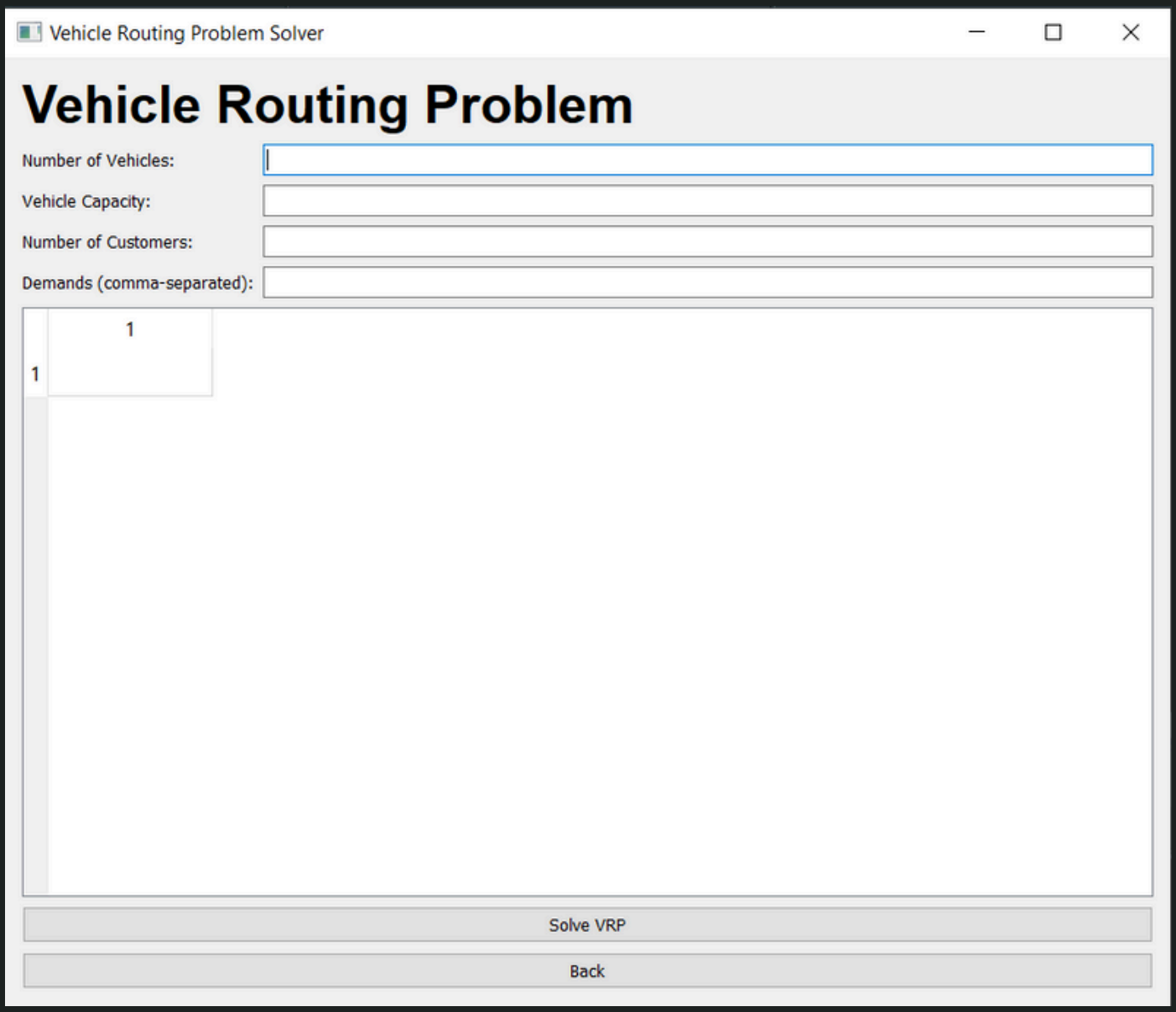
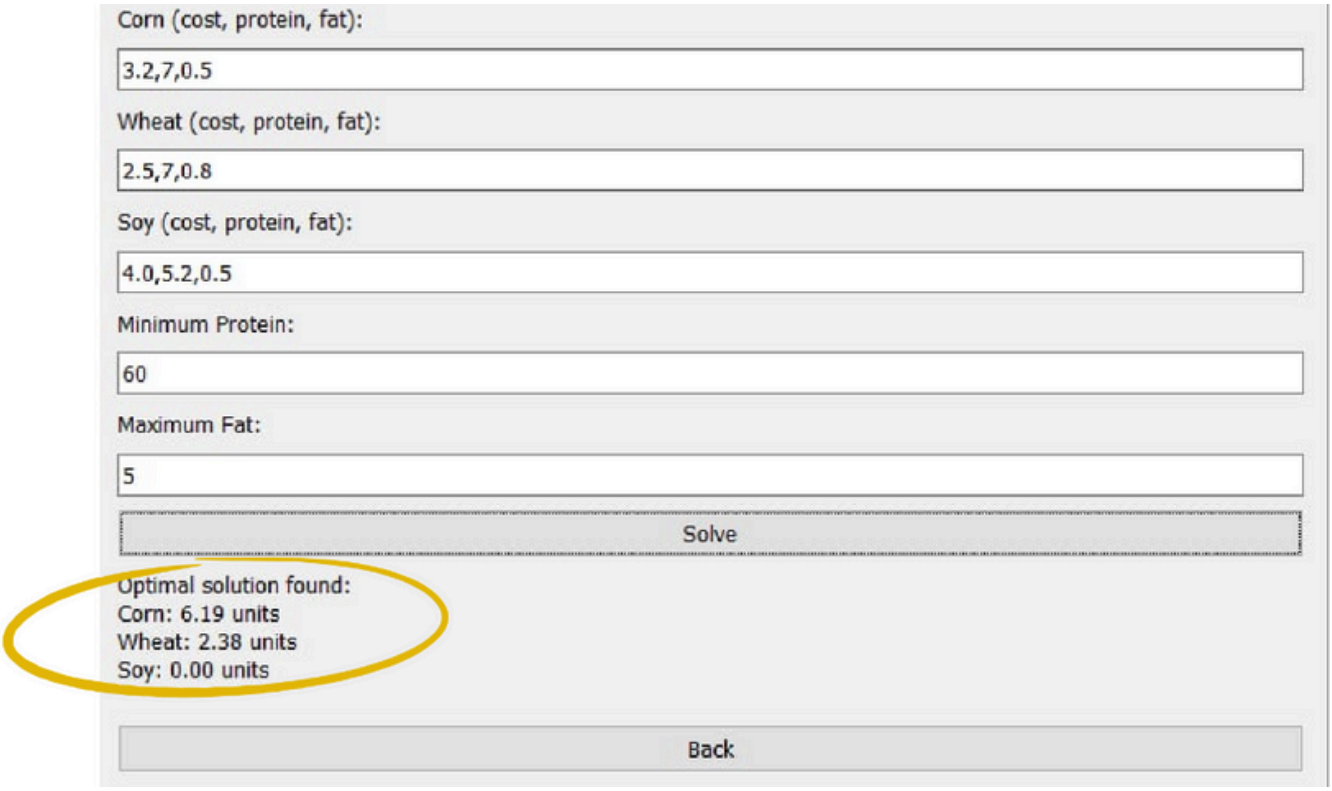


ILLUSTRATION EN PYTHON SUR LA PAGE MAIN :

4.RESULTATS & ANALYSE :

- **Probleme de mélange**

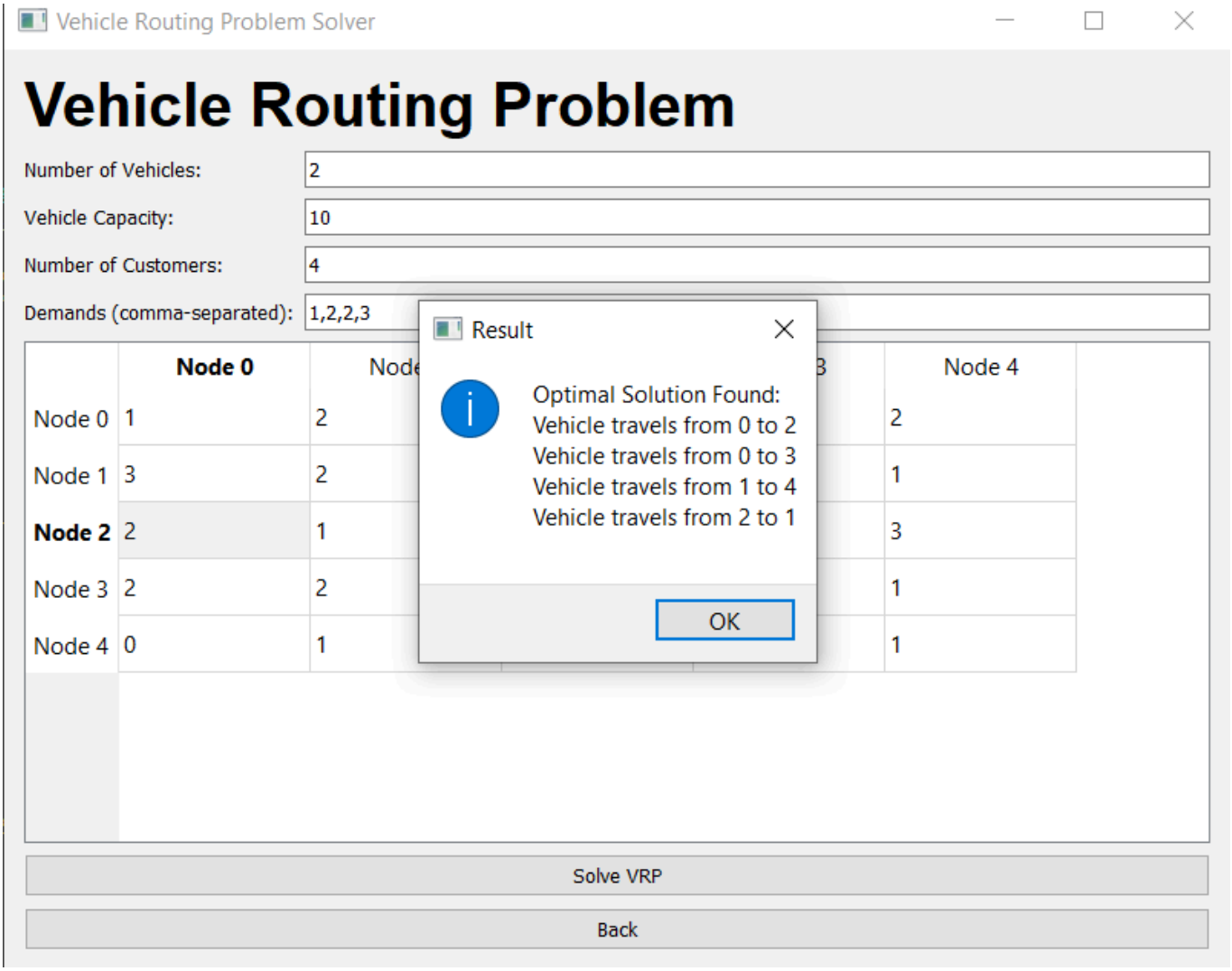


Utilisation des Ingrédients :
Observations : Le maïs et le blé sont utilisés en quantités significatives, tandis que le soja n'est pas utilisé du tout. Cela peut suggérer que le coût et/ou la composition nutritionnelle du soja (en termes de protéines et de gras) le rendent moins favorable pour atteindre les objectifs fixés sous les contraintes données.

Respect des Contraintes Nutritionnelles :
Protéine : Avec les quantités données, la teneur en protéines totale doit être vérifiée pour s'assurer qu'elle atteint au moins le minimum requis de 60. Puisque le problème indique qu'une solution optimale a été trouvée, on peut supposer que cette contrainte a été satisfaite.
Gras : De même, la quantité totale de gras doit être inférieure ou égale à 5 unités. L'absence de soja dans la solution pourrait être due à sa contribution relative au gras qui pourrait potentiellement pousser le total au-delà de la limite autorisée.

Coût :
Analyse : La solution optimale a probablement minimisé le coût total tout en respectant les contraintes. Les coûts unitaires de chaque ingrédient (maïs à 3.2, blé à 2.5, et soja à 4.0) indiquent que le blé, bien qu'utilisé en moindre quantité que le maïs, contribue efficacement à la réduction du coût total grâce à son coût unitaire inférieur.

- **Probleme de routage des vehicules**



- "0 to 2": Cela signifie qu'un véhicule part du dépôt (client 0) et se rend au client 2.
- "0 to 3": Un autre véhicule part du dépôt et se rend au client 3.
- "1 to 4": Un véhicule part du client 1 et se rend au client 4.
- "2 to 1": Un autre véhicule part du client 2 et retourne au dépôt (client 1).

Ces résultats montrent que plusieurs véhicules ont été utilisés pour servir les clients. Chaque véhicule commence sa tournée au dépôt, puis se déplace vers les clients pour effectuer les livraisons. Une fois que les livraisons sont effectuées, les véhicules peuvent revenir au dépôt pour terminer leur tournée.

5.CONCLUSION :

Contributions principales :

1. Implémentation de solutions algorithmiques : Le projet a permis d'implémenter des algorithmes efficaces pour résoudre les problèmes de routage des véhicules et de mélange, en utilisant des outils tels que Gurobi pour la programmation linéaire.
2. Interface utilisateur conviviale : Une interface graphique conviviale a été développée à l'aide de PyQt5, offrant aux utilisateurs la possibilité d'entrer facilement les données du problème et de visualiser les résultats.
3. Résolution optimale : Les solutions obtenues ont été optimisées pour minimiser les coûts associés au routage des véhicules et au mélange des matériaux, contribuant ainsi à l'efficacité opérationnelle des entreprises.

Évaluation des objectifs initiaux :

Les objectifs initiaux du projet étaient probablement de fournir des solutions efficaces et optimales pour les problèmes de routage des véhicules et de mélange, ainsi que de créer une interface utilisateur intuitive pour faciliter leur utilisation. Ces objectifs semblent avoir été atteints avec succès, comme en témoignent les résultats obtenus et la convivialité de l'interface utilisateur.

Limites et perspectives :

Malgré les réussites du projet, quelques limites peuvent être identifiées :

1. Complexité des problèmes : Les problèmes de routage des véhicules et de mélange peuvent devenir très complexes avec un grand nombre de variables et de contraintes, ce qui peut limiter la capacité des méthodes algorithmiques à trouver des solutions optimales dans un temps raisonnable.
2. Optimisation supplémentaire : Il pourrait être intéressant d'explorer des techniques d'optimisation supplémentaires ou des approches heuristiques pour résoudre ces problèmes de manière plus efficace, en particulier pour les instances de grande taille.
3. Gestion de scénarios multiples : Les entreprises peuvent être confrontées à des scénarios variés et dynamiques. Une extension du projet pourrait impliquer la gestion de multiples scénarios et la prise de décision en temps réel pour s'adapter aux changements.
4. Amélioration de l'interface utilisateur : Une amélioration continue de l'interface utilisateur en ajoutant des fonctionnalités supplémentaires et en améliorant l'expérience utilisateur pourrait rendre l'outil encore plus attrayant et facile à utiliser.

Vous pouvez acceder au projet d'ici

https://github.com/HabibTriki/PL-PLNE-Solver