Rail Aventure

Mathieu Ract

10 Mars 2024

1 Introduction

Dans le cadre de la gestion efficace d'un réseau ferroviaire, un problème crucial émerge : celui de l'optimisation des trajets pour les voyageurs. En effet, la fluidité et l'efficacité des déplacements dépendent de plusieurs paramètres clés tels que le temps de trajet, le coût financier et l'impact environnemental. Pour ce faire, il est essentiel de développer une heuristique appropriée qui tienne compte de ces différentes dimensions. Ce défi complexe implique la coordination des gares, des rails et des trains, ainsi que la prise en compte des besoins et des contraintes des voyageurs. Dans cette étude, nous explorerons les différentes composantes de ce problème et proposerons des solutions visant à minimiser cette heuristique.

2 Formalisation

2.1 Réseau ferroviaire

Le réseau ferroviaire est caractérisé par :

- $G = \{G_i \mid i = 1, 2, \ldots, g\}$: ensemble des gares.
- $R = \{R_{ij} \mid i \neq j,\ i, j = 1, 2, \ldots, g\}$: ensemble des rails ralliant les gares G_i et G_j, où g est le nombre de gares.

2.2 Trains

Chaque train t est caractérisé par :

- Une gare de départ GD_t et d'arrivée GA_t avec $t = 1, 2, \ldots, T$: nombre de trains.
- Une heure de départ HD_t et d'arrivée HA_t avec $t = 1, 2, \ldots, T$: nombre de trains.
- Un prix P_t
- Le CO2 émis $\mathrm{CO2}_t$

2.3 Voyageur

Un voyageur est caractérisé par :

- Une gare de départ GD_v et d'arrivée GA_v,
- Une date de départ DD_v.

2.4 Voyage

Un voyage est caractérisé par :

- Un parcours $P = \{T_{k1}, T_{k2}, \ldots, T_{kL}\}$ avec L le nombre de trains empruntés.

2.5 Problématisation

Considérons le problème d'optimisation suivant :

$$\begin{aligned} \text{Minimiser} \quad & \sum_{t \in P} f_t(P_t, \mathrm{CO2}_t, \mathrm{HA}_t, \mathrm{HD}_t) \\ \text{sous contraintes} \quad & GD_{Tk1} = GDv \quad (i) \\ & GA_{TkL} = GAv \quad (ii) \\ & GA_{Tki} = GD_{Tki+1}, \quad i = 1, 2, \ldots, L-1 \quad (iii) \\ & HA_{Tki} \leq HD_{Tki+1}, \quad i = 1, 2, \ldots, L-1 \quad (iv) \end{aligned}$$

(i) La première gare du parcours correspond à la gare de départ du voyageur
(ii) La dernière gare du parcours correspond à la gare d'arrivée du voyageur
(iii) Pour toute liaison, deux trains successifs doivent être reliés par une même gare
(iv) Pour toute liaison, l'heure d'arrivée du train amont doit être inférieur à l'heure de départ du train aval

3 Résolution

3.1 Structure des données

Cette fonction prend en entrée les données des trains ainsi que des informations sur la gare actuelle, l'heure actuelle, le prix accumulé et le CO2 accumulé.
Elle utilise une approche récursive pour créer un arbre représentant les différentes combinaisons de trajets possibles à partir de la gare actuelle.

```python
def transform_data_tree(trains, gare, heure, prix, co2):
    arbre = {'gare': gare, 'heure': heure, 'prix': prix, 'co2': co2, 'trains': []}
    for train_id, train_info in trains.items():
        if (gare == "" or train_info['depart'] == gare) and train_info['depart_heure'] > heure:
            prochaine_gare = train_info['arrivee']
            prochaine_heure = train_info['arrivee_heure']
            prochain_prix = prix + train_info['prix']
            prochain_co2 = co2 + train_info['co2']
            prochain_noeud = transform_data_tree(trains, prochaine_gare, prochaine_heure, prochain_prix, prochain_co2)
            arbre['trains'].append({train_id: prochain_noeud})
    return arbre
```

Prenons un exemple de données de trains circulant sur un réseau ferroviaire fictif :

```python
trains = {
    'train_0': {"depart": "Gare1", "arrivee": "Gare2", "prix": 50, "co2": 10, "depart_heure": 8, "arrivee_heure": 10},
    'train_1': {"depart": "Gare2", "arrivee": "Gare3", "prix": 40, "co2": 8, "depart_heure": 11, "arrivee_heure": 13},
    'train_2': {"depart": "Gare3", "arrivee": "Gare1", "prix": 10, "co2": 4, "depart_heure": 15, "arrivee_heure": 18},
    'train_3': {"depart": "Gare1", "arrivee": "Gare2", "prix": 30, "co2": 11, "depart_heure": 1, "arrivee_heure": 5},
    'train_4': {"depart": "Gare2", "arrivee": "Gare1", "prix": 30, "co2": 11, "depart_heure": 2, "arrivee_heure": 4},
    'train_5': {"depart": "Gare2", "arrivee": "Gare3", "prix": 30, "co2": 11, "depart_heure": 7, "arrivee_heure": 15},
    'train_6': {"depart": "Gare1", "arrivee": "Gare5", "prix": 60, "co2": 15, "depart_heure": 9, "arrivee_heure": 12},
    'train_7': {"depart": "Gare2", "arrivee": "Gare4", "prix": 45, "co2": 12, "depart_heure": 12, "arrivee_heure": 16},
    'train_8': {"depart": "Gare3", "arrivee": "Gare5", "prix": 55, "co2": 14, "depart_heure": 18, "arrivee_heure": 25},
    'train_9': {"depart": "Gare4", "arrivee": "Gare1", "prix": 50, "co2": 13, "depart_heure": 16, "arrivee_heure": 19},
    'train_10': {"depart": "Gare5", "arrivee": "Gare2", "prix": 50, "co2": 13, "depart_heure": 18, "arrivee_heure": 20},
}
```

Visualisons leur arbre de sortie :

```
{
    "gare": "Gare1",
    "heure": 0,
    "prix": 0,
    "co2": 0,
    "trains": [
        {
            "train_0": {
                "gare": "Gare2",
                "heure": 10,
                "prix": 50,
                "co2": 10,
                "trains": [
```

```
            {
                "train_1": {
                    "gare": "Gare3",
                    "heure": 13,
                    "prix": 90,
                    "co2": 18,
                    "trains": [
                        {
                            "train_2": {
                                "gare": "Gare1",
                                "heure": 18,
                                "prix": 100,
                                "co2": 22,
                                "trains": []
                            }
                        },
                        {
                            "train_8": {
                                "gare": "Gare5",
                                "heure": 25,
                                "prix": 145,
                                "co2": 32,
                                "trains": []
                            }
                        }
                    ]
                }
            },
            {
                "train_7": {
                    "gare": "Gare4",
                    "heure": 16,
                    "prix": 95,
                    "co2": 22,
                    "trains": []
                }
            }
        ]
    }
},
{
    "train_3": {
        "gare": "Gare2",
        "heure": 5,
        "prix": 30,
        "co2": 11,
        "trains": [
            {
                "train_1": {
                    "gare": "Gare3",
                    "heure": 13,
                    "prix": 70,
                    "co2": 19,
                    "trains": [
                        {
                            "train_2": {
                                "gare": "Gare1",
                                "heure": 18,
                                "prix": 80,
                                "co2": 23,
                                "trains": []
                            }
                        },
                        {
                            "train_8": {
                                "gare": "Gare5",
                                "heure": 25,
                                "prix": 125,
                                "co2": 33,
                                "trains": []
                            }
                        }
                    ]
                }
            },
            {
```

```
                    "train_5": {
                        "gare": "Gare3",
                        "heure": 15,
                        "prix": 60,
                        "co2": 22,
                        "trains": [
                            {
                                "train_8": {
                                    "gare": "Gare5",
                                    "heure": 25,
                                    "prix": 115,
                                    "co2": 36,
                                    "trains": []
                                }
                            }
                        ]
                    }
                },
                {
                    "train_7": {
                        "gare": "Gare4",
                        "heure": 16,
                        "prix": 75,
                        "co2": 23,
                        "trains": []
                    }
                }
            ]
        }
    },
    {
        "train_6": {
            "gare": "Gare5",
            "heure": 12,
            "prix": 60,
            "co2": 15,
            "trains": [
                {
                    "train_10": {
                        "gare": "Gare2",
                        "heure": 20,
                        "prix": 110,
                        "co2": 28,
                        "trains": []
                    }
                }
            ]
        }
    }
]
}
```

3.2 Affichage des chemins

Cette fonction prend l'arbre généré par la fonction précédente et le transforme en un graphe orienté, où les nœuds représentent les états des voyages possibles et les arêtes représentent les transitions entre ces états. Chaque arête est étiquetée avec le numéro de train correspondant.

```python
def transform_tree_graph(arbre, parent=None, graph=None):
    if graph is None:
        graph = nx.DiGraph()
    current_node = arbre['gare'] + '_' + str(arbre['heure']) + '_' + str(arbre['prix']) + '_' + str(arbre['co2'])
    if parent:
        parent_node = parent['gare'] + '_' + str(parent['heure']) + '_' + str(parent['prix']) + '_' + str(parent['co2'])
        for train in parent['trains']:
            if list(train.values())[0] == arbre:
                train_taken = list(train.keys())[0]
                break
        graph.add_edge(parent_node, current_node, train=train_taken)
    for train in arbre['trains']:
        child = list(train.values())[0]
        transform_tree_graph(child, arbre, graph)
    return graph
```

Cette fonction utilise les données des trains ainsi que la gare de départ et l'heure de départ pour afficher graphiquement le réseau ferroviaire sous forme de diagramme.

```python
def display_graph(trains, gare_depart, heure_depart):
    arbre = transform_data_tree(trains, gare_depart, heure_depart, 0, 0)
    graph = transform_tree_graph(arbre)
    pos = nx.spring_layout(graph)

    # Trouver le nœud initial
    initial_node = None
    for node in graph.nodes():
        if graph.in_degree(node) == 0:
            initial_node = node
            break

    # Définition des options
    node_colors = ['red' if node == initial_node else 'lightblue' for node in graph.nodes()]
    node_size = 300
    edge_color = 'gray'
    width = 2.0
    style = 'solid'
    font_size = 10
    font_color = 'black'
    alpha = 0.8

    # Dessiner le graphique avec les options personnalisées
    nx.draw(graph, pos, with_labels=True, arrows=True,
            node_color=node_colors, node_size=node_size,
            edge_color=edge_color, width=width, style=style,
            font_size=font_size, font_color=font_color,
            alpha=alpha)

    # Ajouter les étiquettes des arêtes
    edge_labels = {(n1, n2): d['train'] for n1, n2, d in graph.edges(data=True)}
    nx.draw_networkx_edge_labels(graph, pos, edge_labels=edge_labels)

    # Afficher le graphique
    plt.show()

    return None
```

Avec les mêmes données que précédemment, il est possible de représenter graphiquement les différents chemins envisageables :

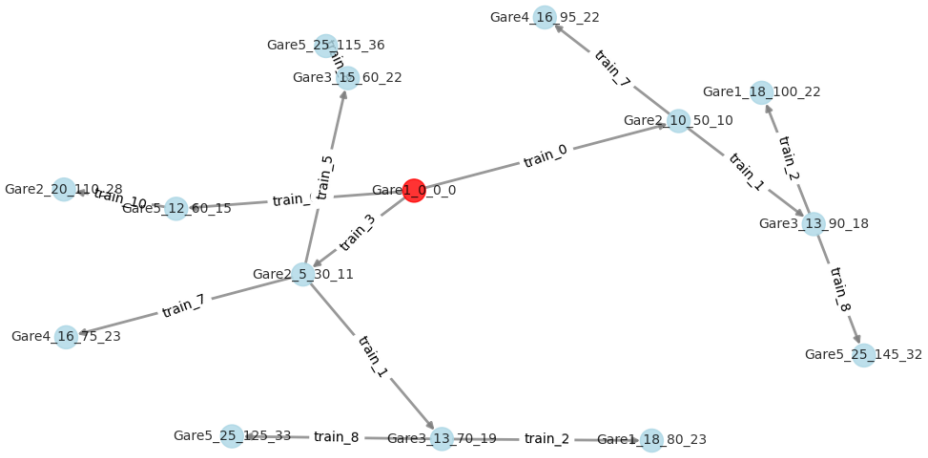

Figure 1: Graph des chemins possibles

3.3 Chemin optimal

Cet algorithme de recherche en profondeur d'abord (DFS) explore récursivement les chemins possibles à partir d'un nœud donné jusqu'à ce qu'il atteigne la destination. Il retourne le chemin optimal en minimisant une heuristique qui dépend du temps, prix et CO2.

```python
def find_optimal_path_DFS(node, destination, current_time=0, current_price=0, current_co2=0):
    if node['gare'] == destination:
```

```
        return {
            'heure': current_time,
            'prix': current_price,
            'co2': current_co2,
            'path': [destination]
        }

    best_path = None

    for train in node['trains']:
        nom_train = list(train.keys())[0]
        next_node = train[nom_train]
        next_time = train[nom_train]['heure']
        next_price = current_price + train[nom_train]['prix']
        next_co2 = current_co2 + train[nom_train]['co2']

        path_result = find_optimal_path_DFS(next_node, destination, next_time, next_price, next_co2)

        if path_result:
            path_result['path'].insert(0, nom_train)
            path_result['path'].insert(0, node['gare'])
            if not best_path:
                best_path = path_result
            else:
                heuristique_path = heuristique(path_result['heure'], path_result['prix'], path_result['co2'])
                heuristique_best_path = heuristique(best_path['heure'], best_path['prix'], best_path['co2'])
                if heuristique_path < heuristique_best_path:
                    best_path = path_result

    return best_path
```

La fonction heuristique calcule une valeur heuristique en fonction des paramètres d'heure, de prix et de CO2. Elle est utilisée dans l'algorithme de recherche pour estimer la qualité des chemins.

```
def heuristique(heure, prix, co2):
    poids_heure = 1
    poids_prix = 2
    poids_co2 = 3

    valeur_heuristique = poids_heure * heure + poids_prix * prix + poids_co2 * co2

    return valeur_heuristique
```

Cette fonction est l'interface principale de l'utilisateur. Elle prend les données des trains et les détails du voyageur (gare de départ, destination, heure de départ) et utilise les autres fonctions pour trouver et afficher le chemin optimal.

```
def find_optimal_journey(trains_data, voyageur):
    arbre = transform_data_tree(trains_data, voyageur['depart'], 0, 0, 0)

    display_graph(trains_data, voyageur['depart'], 0)

    resultat = find_optimal_path_DFS(arbre, voyageur['arrivee'])
    if resultat:
        print("Chemin optimal vers", voyageur['arrivee'], ":")
        print("Heure:", resultat['heure'])
        print("Prix:", resultat['prix'])
        print("CO2:", resultat['co2'])
        print("Chemin:", ' -> '.join(resultat['path']))
    else:
        print("Aucun chemin trouvé vers", voyageur['arrivee'])
```

Voici en exemple d'inférence du modèle :

```
voyageur = {"depart": "Gare1", "arrivee": "Gare4", "date_depart": 7}
find_optimal_journey(trains, voyageur)

Chemin optimal vers Gare4 :
Heure: 16
Prix: 105
CO2: 34
Chemin: Gare1 -> train_3 -> Gare2 -> train_7 -> Gare4
```