TIPE 2023

Parcourir la ville efficacement - offrir aux visiteurs de Paris des itinéraires optimisés

Marilou Bernard de Courville

Nº SCEI 41188

2 octobre 2023

Introduction

Problématique et pertinence au regard du thème de l'année

- **Objectif :** parcourir efficacement la ville dans un but touristique.
- **Recherche opérationnelle :** permet d'atteindre l'objectif en réalisant des optimisations sous contraintes [1].
- **Deux cas** envisagés :
  1. optimiser un trajet en métro via l'algorithme de Dijkstra et différentes structures de données codées en OCaml [3] ;
  2. optimiser un trajet soit en métro ou à pied en passant par plusieurs types de chemins en utilisant un solveur linéaire (CPLEX d'IBM) et un code en Python.
- Ces deux cas sont modélisés à l'aide de **graphes orientés**.

Trajet le plus court en métro - problème

Données du problème

- Système étudié : métro parisien
- Problématique : plus court trajet entre 2 stations
- Graphe conséquent : 302 sommets, 347 arêtes
- Contribution : structures de données adaptées à une résolution efficace en temps
- Rapport thème : qualifier distance maximale acceptable entre hôtel et lieux de visite.

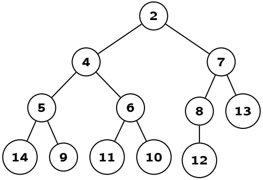
plan-metro.png

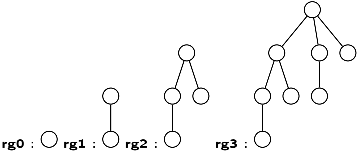
graphe-metro-eps-converte

Trajet le plus court en métro - modélisation

Modèles de données utilisés (types)

- Un tas est une représentation organisée en arbre des stations en fonction d'une priorité (distance du chemin).
- Représentation en tas (heap) efficace en complexité pour accéder au nœud de priorité minimum et mettre à jour les priorités.
- Trois structures étudiées : tas non modifiables (*immutable heaps*), tas modifiables, et tarbres (*treap*).
- Programmes réalisés en OCaml utilisent seulement le module `Hashtbl` de la bibliothèque standard pour manipuler les tas.





2	4	7	5	6	8	13	14	9	11	10	12

Élément	2	4	5	6	7	8	9	10	11	12	13	14
Position	0	1	3	4	2	5	8	10	9	11	6	7

Trajet le plus court en métro - algorithme

Algorithme de Dijkstra et influence des structures de données sur la complexité

Require: Un graphe $G = (V, A)$, V sommets, A arêtes
Require: Un noeud source s
Ensure: d tableau des plus court chemins de s vers $v \in V$

```
for all v ∈ V[G] do
    d[v] ← +∞, père[v] ← None
end for
d[s] ← 0, S ← ∅, Q ← V[G]
while Q ≠ ∅ do
    u ← Extrait_Min(Q)
    S ← S ∪ {u}
    for all arête (u, v) d'origine u do
        if d[u] + w(u, v) < d[v] then
            d[v] ← d[u] + w(u, v),
père[v] := u
        end if
    end for
end while
```

Table – Dijkstra : complexité (opérations)

Implantation	Complexity
Naif	$\mathcal{O}\left(V^2 + A\right)$
Tas	$\mathcal{O}\left((V + A)\log V\right)$

Table – Simulations : temps exécution pour toutes les stations du métro

Type	Temps exécution
Naif	960ms
Tas non mutable v1	312ms
Tas mutable	191ms
Tarbre	1018ms
Tas non mutable v2	145ms

Trajet le plus court en métro - résultats

Quelles sont les stations les plus éloignées ?

solution-dijkstra.png

Plus compliqué : plus petit chemin passant par toutes les lignes du métro

Étude de la résolution à l'aide d'un solveur linéaire

- Sujet traité par Florian Sikora [7] par étude de graphe coloré pour le réseau du métro ;
- Sommet : station, arête : trajet entre deux stations connexes, arête colorée par la couleur de la ligne reliant les stations
- Problème du "Generalised directed rural postman" [2] ;
- Problème NP-difficile : pas de solution en temps polynomial [6] ;
- Résolution requiert l'utilisation d'un solveur linéaire (CPLEX d'IBM) pour "integer linear programming" (ILP) [9] ;
- Fait intervenir une matrice de contraintes (MIP) de `1270x1847, 6999 coeffs non nuls.`

Solution : un plus petit chemin passant par toutes les lignes du métro

Résolution à l'aide d'un solveur linéaire

solution-sikora.png

Adaptation dans un but touristique à Paris

Problématique : trouver le plus court trajet qui passe par différents "types" de chemins et des monuments

20 monuments considérés :
1 Tour Eiffel, 2 Musée du Louvre,
3 Palais du Luxembourg, 4 Centre Pompidou,
5 Sainte-Chapelle, 6 Musée d'Orsay,
7 Fondation Louis Vuitton, 8 Panthéon,
9 Arc de triomphe, 10 Musée quai Branly,
11 Institut monde arabe, 12 Musée Rodin,
13 Musée de Cluny, 14 Grand Palais,
15 Notre-Dame, 16 Sacré-Coeur,
17 Hotel de Ville, 18 Place de la Concorde,
19 Palais Garnier, 20 Père Lachaise.

chemins-small.jpg

graphe.png

5 types de chemins (activités) passant :

- par des espaces verts
- par les quais de la Seine
- par des quartiers historiques
- proche d'architectures parisiennes typiques
- par des rues commerçantes, grands magasins

Formalisation du problème : variables

- Ensemble V des sommets, A des arêtes, C des couleurs. $(u, v) \in V^2$, $u \xrightarrow[l \in C]{} v \in A$.
- $x_{u,v,l} \in \{0, 1\}$: variable binaire pour chaque arc $u \xrightarrow[l]{} v$ (sur ligne l), avec $x = 1$ si l'arc est considéré, $x = 0$ sinon.
- $w_{u,v,l} \in \mathbb{N}$: est le temps pour parcourir l'arête $u \xrightarrow[l]{} v$
- $(u, v) \in V^2, f_{u,v,l}, y_v \in \mathbb{N}$ sont les flots des arcs/sommets : positifs si l'arc/sommet est sur le chemin considéré.
- s, t sont les points de départ/arrivée fictifs (temps nul pour rejoindre tout sommet).
- $\forall((u, v, l_1), (v, w, l_2)) \in A^2, z_{u,v,w,l_1,l_2} \in \{0, 1\}$ indique si deux arêtes sont utilisées consécutivement $u \xrightarrow[l_1]{} v \xrightarrow[l_2]{} w$.

Formalisation du problème : optimisation

Objectif : minimiser distance $\sum_{(u,v,l)\in A} w_{u,v,l} \times x_{u,v,l}$ sous contraintes.

Autant de chemins qui entrent sur un sommet et qui en sortent :	$\forall v \in V\backslash\{s,t\}, \sum_{(u,v,l)\in A} x_{u,v,l} = \sum_{(v,w,l)\in A} x_{v,w,l}$	(1)
Unique chemin de la source et vers la cible :	$\sum_{(s,v,l)\in A} x_{s,v,l} = \sum_{(u,t,l)\in A} x_{u,t,l} = 1$	(2)
Pour chaque ligne au moins 1 arc de cette ligne sélectionné :	$\forall l \in C, \sum_{(u,v,l)\in A} x_{u,v,l} \geq 1$	(3)
Evite solutions disjointes : le flot est décroissant pour solution connectée	$\forall (u,v,l) \in A, \mid V \mid x_{u,v,l} \geq f_{u,v,l}$	(4)
Tous les sommets perdent du flot sauf la source :	$\forall v \in V\backslash\{s\}, \sum_{(u,v,l)\in A} f_{u,v,l} - \sum_{(v,w,l)\in A} f_{v,w,l} \geq y_v$	(5)
Le flot au niveau du sommet est positif s'il est dans la solution :	$\forall v \in V, y_v - \sum_{(u,v,l)\in A} x_{u,v,l} - \sum_{(v,w,l)\in A} f_{v,w,l} \geq 0$	(6)
Evite de prendre deux fois le même sommet :	$y_v \geq 2$	(7)
Chemin consécutif :	$x_{u,v,l_1} + x_{v,w,l_2} \leq z_{u,v,w,l_1,l_2} + 1$	(8)
Evite de reprendre deux fois la même ligne :	$\forall l \in C, \sum_{(u,v,l_1),(v,w,l_2)\ /\ l=l_1 \text{ ou } l=l_2} z_{u,v,w,l_1,l_2} = 2$	(9)

Méthode de résolution

Outils mis en oeuvre

- Utilisation de Google Maps REST API et un programme python pour récupérer automatiquement la matrice des distances entres monuments.
- Représentation du graphe en utilisant le module `Networkx` et une structure de donnée `MultiDiGraph` de python.
- Résolution du problème d'optimisation linéaire à variables entières (ILP) via un programme python s'interfaçant au solveur CPLEX d'IBM.

Résultats notables dans le cadre de la ville :

Meilleur trajet sans contrainte de départ et d'arrivée :

Application possible au tourisme : garantir un maximum d'activités (chemins) en un minimum de temps tout en croisant des monuments.

Distance minimale trouvée par CPLEX pour le parcours : 2149m

sol_chemin.png

1. s $\rightarrow$ Panthéon
2. Panthéon $\rightarrow$ Palais du Luxembourg
3. Palais du Luxembourg $\rightarrow$ Musée de Cluny
4. Musée de Cluny $\rightarrow$ Notre-Dame
5. Notre-Dame $\rightarrow$ Hotel de Ville
6. Hotel de Ville $\rightarrow$ Centre Pompidou
7. Centre Pompidou $\rightarrow$ t

Résultats notables dans le cadre de la ville :

Meilleur trajet cyclique (on sélectionne un point de départ dans le cycle) :

Application possible au tourisme : trouver un hotel localisé sur le cycle qui permet de faire un parcours dans une journée et garantir un maximum d'activités (chemins) en un minimum de temps tout en croisant des monuments.

Distance minimale trouvée par CPLEX pour le cycle : `6268m`

sol_cycle.png

1. `Musée du Louvre` → `Musée d'Orsay`
2. `Musée d'Orsay` → `Place de la Concorde`
3. `Place de la Concorde` → `Grand Palais`
4. `Grand Palais` → `Musée Rodin Paris`
5. `Musée Rodin Paris` → `Musée de Cluny`
6. `Musée de Cluny` → `Sainte-Chapelle`
7. `Sainte-Chapelle` → `Musée du Louvre`

Principe de fonctionnement d'un solveur linéaire

Étude de la résolution à l'aide d'un solveur linéaire - code en python

- Formalisme programmation linéaire entière ILP : minimiser fonction objectif sous contraintes $\min_{Ax \leq b, A'x = b', x_i \in \mathbb{N}} w^\top x$, d'inconnues à variables entières [8]

Example-Integer-Linear-Program-ILP.png

- Solutions de l'ILP ne se déduisent pas par arrondi de celles obtenues par Linear Programming (LP) (variables réelles) par méthode du simplex.
- CPLEX : progiciel de résolution des problèmes ILP [9].

- Procédé itératif de diviser pour régner par l'algorithme "branch-and-cut" : combinaison de méthodes "branch-and-bound" et de "cutting-plane" en relaxant la contrainte de variable entière pour une résolution avec du LP.

Conclusion

- Deux problématiques urbaines traitées :
  - optimisation d'un trajet en métro
  - parcours touristique efficace d'une ville en empruntant différents types de chemins
- Pertinence de la modélisation des problèmes urbains par des graphes pour les résoudre.
- Application de la recherche opérationnelle pour trouver une solution.
  - Optimisation fait intervenir un grand nombre de contraintes résultant en des problèmes combinatoires complexes sans solution analytique.
  - Importance du choix des algorithmes et structures de données pour obtenir des solutions pratiques efficaces.

Illustration méthode branch and bound

Un exemple simple à 2 variables

$$\mathrm{IP}^0 : \quad \operatorname{argmax} \quad 13x_1 + 8x_2$$

$$\begin{cases} x_1 + 2x_2 \leq 10 \\ 5x_1 + 2x_2 \leq 20 \\ (x_1, x_2) \in \mathbb{N}^2 \end{cases}$$

Notations :

- $\underline{z}_{ip}$ désigne la valeur de la meilleure solution entière connue, initialisée à $-\infty$;
- z_i^R est la valeur de la solution LP en relaxant la contrainte pour IP^i.

bnbegpic.jpeg

Dijkstra - algorithme naif - I

Code OCaml

Dijkstra - Tas non mutable v1 - I

Code OCaml

Dijkstra - Tas mutable - I

Code OCaml

Dijkstra - Tas binomiaux - I

Code OCaml

Dijkstra - Tas non mutable v2 - I

Code OCaml

Dijkstra - Calcul du temps d'exécution - I

Code OCaml

Dijkstra - Fonction pour trouver le max. - I

Code OCaml

ILP - Monuments de Paris - I

Code python