Rapport problème ouvert : Échange de colis

Enzo Slamnia - Eliot Revol

Le problème qui nous est posé est celui de l'échange de colis. On veut trouver un point de rencontre pour deux individus ayant chacun une origine et une destination. Ce point doit faire partie des deux chemins des individus tel que la somme des temps de ces deux chemins est minimale.
On suppose que les graphes sur lesquels on pose le problème sont non-orientés et sont connexes. De ce fait, chaque point du graphe peut potentiellement être un point de rencontre.

Nous avons en premier lieu approché le problème en cherchant des solutions "triviales", ou qui semblaient logiques d'un point de vue schématique.
La première à été de réaliser chaque plus court chemin entre les quatre points d'origine et de destination. Cela nous donne six chemins pour lesquels nous faisons l'hypothèse qu'il existe forcément un point d'intersection entre deux d'entre eux et qui serait notre point de rencontre. Cette hypothèse est fausse et plusieurs cas montrent que ce point d'intersection peut ne pas exister. Le schéma ci-dessous le prouve.

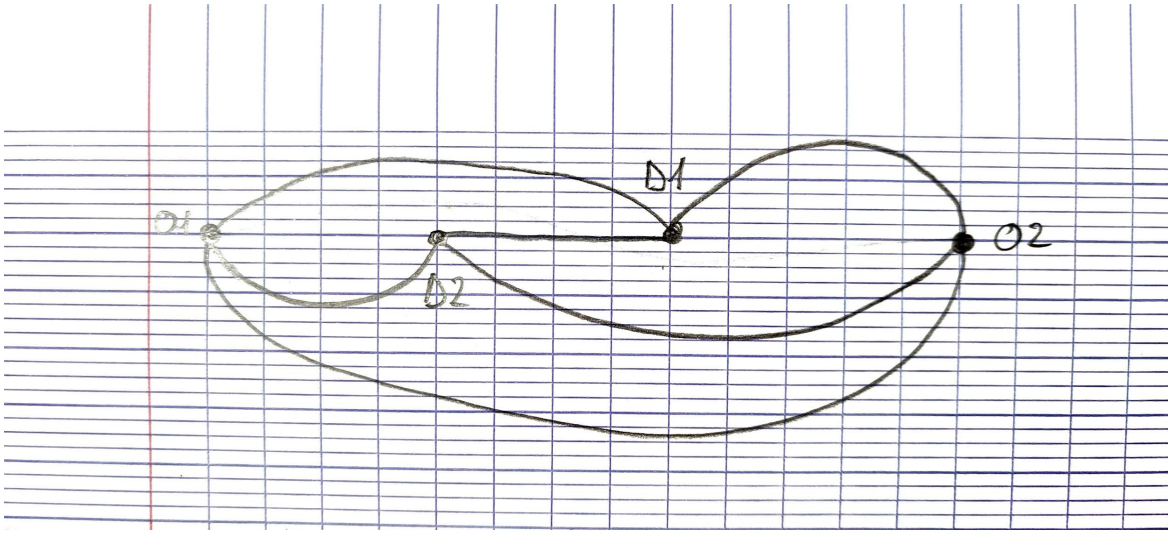

Schéma d'un graphe admettant des plus courts chemins sans point d'intersection

Dans un deuxième temps, nous avons décidé d'utiliser le concept de l'isochrone. Deux points sont isochrones par rapport à un point d'origine s'ils sont atteignable depuis ce dernier avec un temps ou une distance inférieurs à une borne fixée. Cette méthode permet de délimiter des zones de nœuds. L'idée que nous avons eu est de prendre un chemin trivial qui permette aux deux individus de se rencontrer, puis de tracer des isochrones inférieurs à la longueur de ce chemin depuis les quatre points d'origine et de destination. Ceci est réalisé avec l'algorithme de Dijkstra et la longueur maximale en tant que condition d'arrêt. Une fois que l'on obtient un groupe

de points situés dans les quatre zones en même temps, on réalise une nouvelle fois les isochrones avec une condition d'arrêt plus petite. On réalise cette boucle jusqu'à n'obtenir aucun point appartenant aux quatre isochrones, puis on revient à l'itération précédente pour trouver le point d'intersection à une distance minimale des quatre points de base. Le problème avec le point que l'on trouve est qu'il ne répond pas à l'objectif initial. Ce point sera seulement à équidistance minimale des quatre autres et ne garantit en rien que cette distance est la plus basse.

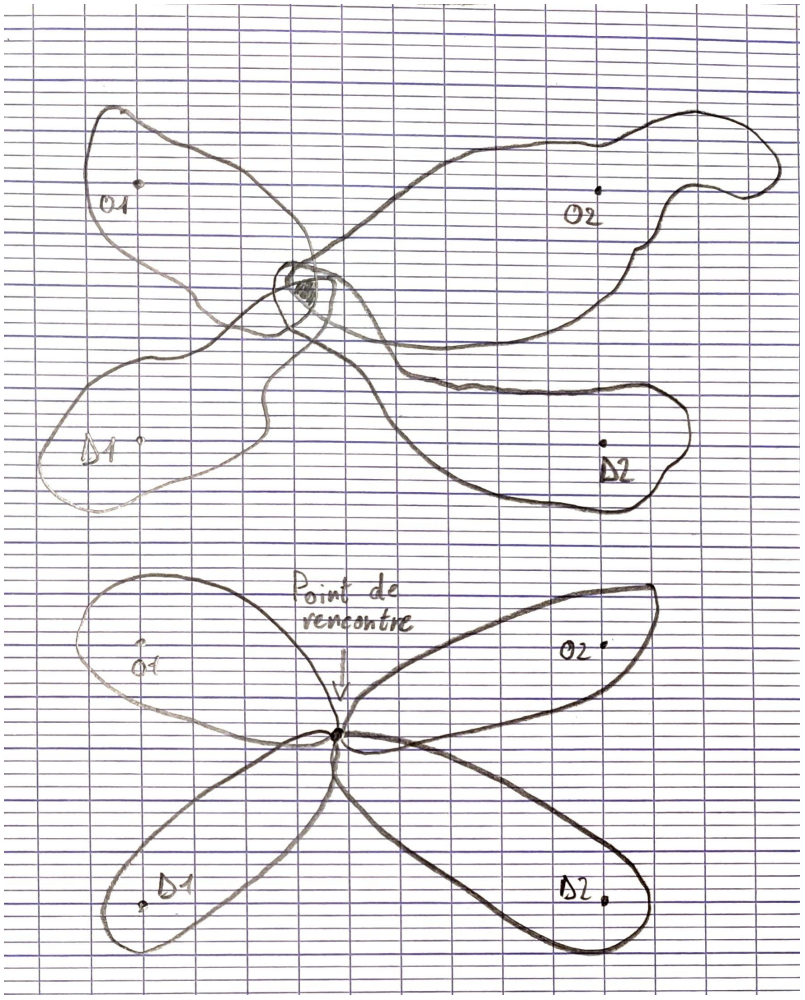

Schéma des étapes de notre algorithme utilisant les isochrones

Enfin, nous avons trouvé une méthode qui permet de trouver la solution optimale. En réalisant l'algorithme de Dijkstra depuis les quatre points vers l'ensemble des points du graphe, on obtient tous les plus courts chemins pour tous les points d'intersections candidats. Ensuite, on réalise un parcours des résultats pour chaque point du graphe, en additionnant les longueurs des quatre chemins vers les origines et destinations. Enfin on compare ces sommes entre elles pour trouver quel point d'intersection est à une durée minimale des quatre autres.

Comme cette méthode est une recherche exhaustive du point d'intersection, on est certain de l'optimalité du résultat.

La complexité de l'algorithme est en 4nlog(n) + n pour les 4 instances de l'algorithme de Dijkstra lancées et pour la recherche de la solution dans les résultats obtenus.

Cette complexité peut être réduite en reprenant la méthode 2 et en choisissant un chemin trivial (par exemple, O1->D1 et O1->D1->D2 en utilisant aStar) comme durée maximale. A chaque fois que l'algorithme atteindra un nœud dépassant cette durée, on marquera le reste des nœuds en bleu et on n'effectura pas le reste des algorithmes sur ces nœuds marqués qui, forcément, ne seront pas dans la solution finale. On pourrait encore optimiser en choisissant le chemin de plus cours durée parmis tous les chemins possibles (par exemple, entre O1->D1, O1->D1->D2 et O2->D2 et O1->D2->D1)

L'algorithme en question est décrit ci-dessous :

```
Données : origine O1, O2; destinations D1, D2

//Inititalisation

DureeMin <- duree(O1,D1) + duree(O2,D1) + duree(D1,D2)//duree a ne
pas dépasser dans l'algorithme (heuristique), calculer avec astar
(plus rapide)

xmin <- D1 //point d'intersection minimal

tas_binaire[4] <- new tasBinaire<label> // on fait quatre tas
binaire pour les 4 dijkstra à effectué

nombredeNoeud <- graphe.nombreDeNoeud

tableau_label[4][nombreDeNoeud] <- {0} //matrice contenant tous
les labels de chaque algo associé à l'id de noeud

tableau_couleur[nombreDeNoeud] <- {rouge}

tableau_point <- [O1,O2,D1,D2]

//Algorithme de dijkstra

pour j allant de 0 à 4
pour i allant de 0 à nombreDeNoeud
      si tableau_point[i] != origine
            tableau_label[j][i] = new label(score = infini)
      sinon
            tableau_label[j][i] = new label(score = 0)
      si tableau_couleur[i] != bleu //on ne refait pas les sommets
marqués en bleu (inutile)
```

```
          tas_binaire.add(tableau_label[j][i])

     tant que tas_binaire[j] non vide
          on récupère le prochain du tas binaire et on l'enlève
(indice k)
          si score > DureeMin

               do
                    tableau_couleur[k] <- bleu
                    on récupère le prochain du tas binaire et on
l'enlève (indice k)
               while le tas_binaire[j] est non vide //on marque
les sommet inutiles en bleu

               sortir du tant que (break)

          sinon
               marquer le label sorti (dans le tableau de label)
               pour tous les arcs qui suivent
                         si score label_destination < score du
label_origne + distance(arc)
                         score label_destination <- score du
label_origne + distance(arc)
     fin tant que
fin pour
fin pour


pour i allant de 0 à nombreDeNoeud
     si tableau_couleur[j] != bleu
          d1 <- 0
          pour j allant de 0 à 4
               d1 <- d1 + tableau_label[j][i].score() //d1 <-
dist(O1,noeud[i]) + dist(noeud[i],D1) + dist(O2,noeud[i]) +
dist(noeud[i],O2)
          fin pour
          si d1 < DureeMin
               DureeMin <- d;
               xmin <- noeud[i];
          fin si
     fin si
fin pour

retourner chemin((O1,xmin),(xmin,D1),(O2,xmin),(xmin,D2))
```

On pourrait également effectuer le même type d'algorithme pour n chemins, mais le temps d'exécution serait à chaque fois plus grand car on effectuera cette version de dijkstra 2*n fois.