Interstices

Découvrir

Le plus court chemin

Par Jean-Michel Hélary

NIVEAU DE LECTURE **Intermédiaire** • • •

Publié le 17/11/2005

Il est courant, lorsque l'on cherche à se rendre d'un point à un autre dans un réseau (routier, par exemple), de chercher le plus court chemin, c'est-à-dire celui dont la distance est la plus petite. Si le nombre de trajets possibles entre le point de départ et le point d'arrivée est faible, il suffira de calculer les longueurs de chacun des trajets - en additionnant la longueur des liens qui le composent - et de comparer directement les longueurs obtenues. Mais une telle solution exhaustive devient rapidement impraticable si le nombre de trajets possibles est grand.
Heureusement, il existe des algorithmes qui évitent d'avoir à calculer tous les trajets possibles. Pour cela, ils mettent en œuvre diverses stratégies.

1. Un problème moins simple qu'il n'y paraît

Un problème vite inextricable

On représente un réseau par un graphe composé de nœuds (les sommets du graphe) et de liens orientés (les arcs du graphe). Imaginez, par exemple, un réseau composé de 20 nœuds - ce qui est très raisonnable - et comportant un lien direct entre chaque nœud (soit 380 liens orientés). Un simple calcul combinatoire montre qu'entre deux nœuds donnés, le nombre de « chemins » possibles est supérieur à 6000 milliards ! Ainsi, à supposer qu'il faille 1 micro-seconde pour calculer la longueur d'un trajet, la solution exhaustive prendrait environ 6 milliards de secondes, c'est-à-dire plus de 740 000 jours... et même si on réduisait le temps de calcul d'un trajet à 1 nano-seconde, il faudrait encore 741 jours. Cet exemple est extrême, mais il montre bien la nécessité de disposer d'algorithmes évitant de calculer tous les trajets possibles, pour éviter l'« explosion combinatoire ». Heureusement, on connaît de tels algorithmes, dont les temps de calculs resteront proportionnels, dans le pire des cas, au cube du nombre de nœuds (et souvent au carré). Ainsi, pour notre exemple, le nombre de chemins calculés sera au pire de 20^3 c'est à dire 8000 ou même de 20^2 c'est-à-dire 400, selon les cas. Ce qui est nettement mieux, chacun en conviendra.

Un cas particulier d'un problème plus général

Le problème sous-jacent est celui du « chemin de valeur optimale (ou de meilleure valeur) dans un graphe valué ». L'exemple le plus connu est sans doute celui du réseau de communication : il est composé de nœuds et de liens, en général orientés, chaque lien reliant directement un nœud origine à un nœud extrémité. On imagine aisément la diversité du type de valeurs pouvant être attribuées aux

liens : distances, temps (de parcours), coûts (de transferts), fiabilités (valeurs comprises entre 0 et 1), capacités (débits maxima, par exemple en kb/s), etc. Le long d'un chemin, les distances, temps, coûts vont en général s'additionner, les fiabilités vont se multiplier, les capacités vont se composer par minimum, et l'on aura tendance à rechercher un minimum dans le cas des distances, temps, coûts, mais un maximum dans le cas des fiabilités ou des capacités.

On imagine aisément toutes les applications d'un tel mécanisme : étude d'un itinéraire routier, organisation d'un réseau de télécommunication, système d'information de trafic d'un réseau ferroviaire, analyse de réseaux sémantiques (par exemple de citations bibliographiques), etc.

L'étude de ce problème générique nécessite une présentation formelle qui fait appel à des notions algébriques avancées. Nous avons choisi de nous limiter ici au problème de recherche du « plus court chemin », que nous préférons appeler « chemin de valeur additive minimale ».

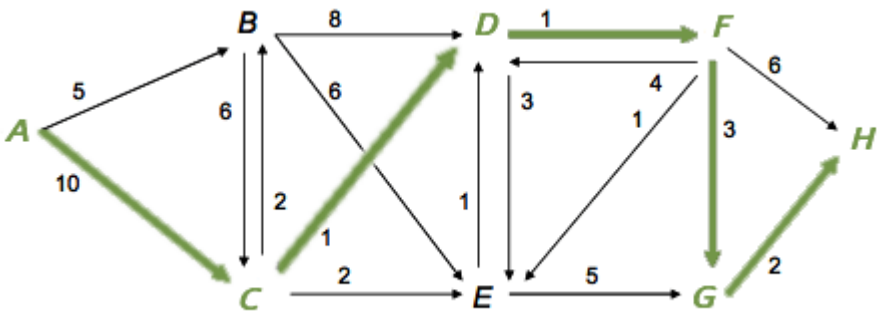
Mieux formaliser le problème pour le résoudre plus efficacement

Précisons maintenant le problème. Nous en distinguons trois types :

1. plus court chemin entre deux sommets donnés;
2. plus courts chemins d'origine fixée mais vers n'importe quelle extrémité ;
3. plus courts chemins de n'importe quelle origine vers n'importe quelle extrémité.

À première vue, il semble que ces problèmes aillent du plus simple au plus complexe. En effet, le nombre de chemins à déterminer augmente. Cependant, et contrairement à l'intuition précédente, la résolution du problème (1) nécessite en général de résoudre le problème (2) ! De plus, dans le cas général, où les valeurs des liens peuvent être négatives, le meilleur algorithme connu pour résoudre le problème (3) ne demande pas plus d'opérations que le meilleur algorithme connu du problème (2). C'est pourquoi nous nous focaliserons sur cet algorithme, celui de Roy-Warshall-Floyd.

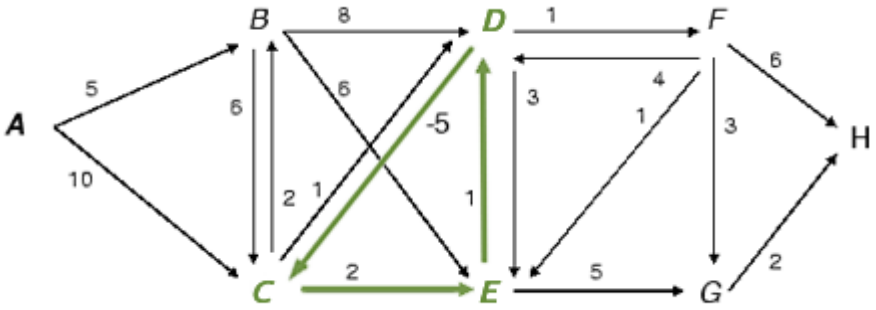
Il convient de préciser aussi ce que l'on cherche à déterminer : le premier résultat cherché est bien sûr la valeur minimale (pour chaque couple de sommets considéré).



Meilleur chemin de A à H.

Par exemple, dans le graphe représenté sur cette figure, le meilleur chemin pour aller de A à H a pour valeur 17. Mais souvent cela ne suffit pas : s'il est bien de connaître la meilleure valeur possible, il est encore mieux de connaître le tracé d'un chemin ayant cette valeur. Ainsi, le meilleur chemin de A à H a pour valeur 17, et son tracé est A;C;D;F;G;H. On parle alors de routage.

Le problème étant ainsi précisé, peut-on affirmer qu'il y a toujours une solution ? La question peut paraître triviale, mais elle ne l'est pas. Supposons que les valeurs des arcs soient de signe quelconque, et que ces valeurs s'additionnent le long des chemins. Si le graphe possède des circuits, c'est-à-dire des chemins permettant de revenir au point de départ, il peut exister des chemins empruntant de tels circuits.



Graphe avec circuit absorbant.

Sur cette figure, par exemple, le chemin A;B;D;C;E;D;F;G;H comprend le circuit D;C;E;D. Dans cet exemple, le circuit D;C;E;D a pour valeur -2, qui est <0. Si l'on cherche les chemins de valeur minimale entre A et H, le problème n'a pas de solution. En effet, le chemin A;B;D;F;G;H vaut 19. Le chemin A;B;D;C;E;D;F;G;H est meilleur (valeur 17). Le chemin A;B;D;C;E;D;C;E;D;F;G;H, faisant deux tours, est encore meilleur (valeur 15). Et ainsi de suite : on peut trouver un chemin de valeur aussi petite que l'on veut, en « tournant » suffisamment de fois sur le circuit ! De tels circuits s'appellent circuits absorbants, et leur présence éventuelle doit être détectée par les algorithmes, qui sinon risqueraient de « boucler » indéfiniment. La question n'est pas seulement théorique, car il existe des applications concrètes où les graphes peuvent avoir des circuits dont la valeur est de signe quelconque. C'est le cas par exemple pour les problèmes d'ordonnancement (ou plannings), qui nécessitent le calcul de chemins de valeur maximale.

Différentes stratégies algorithmiques

Si on s'intéresse uniquement à trouver le meilleur chemin issu d'un sommet donné, il existe des algorithmes assez simples et répandus dans lesquels, par construction, il ne peut pas y avoir de circuits absorbants.

L'algorithme ordinal repose sur le principe très simple d'exploration à partir des prédécesseurs, en supposant qu'il n'y a pas de circuit, et que le sommet de départ est le seul sommet sans prédécesseur.

L'algorithme de Dijkstra quant à lui repose sur le principe d'« exploration à partir du meilleur », c'est à dire du meilleur prédécesseur visité. Il peut être utilisé quand tous les arcs ont une valeur non négative, correspondant à ce qui est consommé sur un trajet donné. Dans ce cas, il ne peut pas y avoir de circuits absorbants, mais il peut y avoir d'autres types de circuits, comme des liens bi-directionnels. En pratique, cet algorithme est très souvent utilisé pour résoudre des problèmes de routage dans les réseaux de télécommunication.

Ces deux algorithmes peuvent se substituer à un algorithme plus général, l'algorithme de Bellman-Kalaba (connu aussi comme Bellman-Ford). Cet algorithme ne présuppose pas l'absence de circuits absorbants sur le graphe, et peut détecter et signaler ces circuits éventuels. Il s'appuie sur un des principaux principes d'optimisation combinatoire : le principe de la programmation dynamique, formulé à l'origine par Bellman et Kalaba.

Ces trois mécanismes peuvent être considérés comme des algorithmes d'exploration de la descendance du sommet, avec des stratégies différentes : tous les prédécesseurs calculés pour l'algorithme ordinal ; le meilleur visité, pour celui de Dijkstra ; et enfin par longueur successive pour celui de Bellman-Kalaba.

Les deux premiers ont une complexité plus faible (de l'ordre de n^2 dans le pire cas) au lieu d'être de l'ordre de n^3 pour le dernier, tout comme pour l'algorithme de Roy-Warshall-Floyd. Ils sont donc utiles dans des situations où il y a un très grand nombre de sommets. C'est le cas par exemple des grands réseaux, des réseaux sémantiques (moteurs de recherche), ou encore des systèmes de recherche

d'horaires. Ainsi, un graphe modélisant le système horaire des chemins de fer allemands comprend de l'ordre de 1 million de sommets et 1 million et demi d'arcs.

De plus, pour l'algorithme ordinal ou celui de Dijkstra, chaque étape nous fournit la valeur définitive d'un sommet, et on le sait. Si on s'intéresse aux chemins de valeur minimale entre le sommet de départ A et un sommet particulier B, on peut arrêter le calcul dès que B est calculé, même si tous les sommets ne sont pas calculés. On dit que la stratégie est gloutonne, ce qui signifie que certains résultats ne seront plus remis en question (un glouton est quelqu'un qui avale et ne peut pas revenir en arrière, c'est-à-dire ne peut pas remettre en cause ce qu'il vient de faire).

Pour l'algorithme de Bellman-Kalaba ou celui de Roy-Warshall-Floyd, en revanche, aucun sommet ne peut être considéré comme calculé avant que tous les sommets soient calculés : jusqu'à la fin de l'exécution (c'est-à-dire jusqu'à ce que la stabilité soit atteinte), toute valeur est susceptible d'être modifiée. Il s'agit là d'une stratégie de point fixe. Ce sont deux stratégies algorithmiques très générales que l'on rencontre ici.

2. Trouver les chemins issus d'un sommet : l'algorithme de Dijkstra

Cet algorithme est dû à Edsger W. Dijkstra (1930-2002), qui est l'un des trois ou quatre informaticiens les plus importants du 20e siècle : il reçut, en 1972, le prestigieux Turing Award, équivalent du Prix Nobel ou de la médaille Fields. Il publia l'algorithme décrit ici en 1959 (« A Note on Two Problems in Connexion with Graphs », Numerische Mathematik, vol.1, PP. 269-271, 1959), l'année où il soutint sa thèse à l'Université d'Amsterdam. Un excellent article (en anglais), téléchargeable en PDF, résume ses nombreuses contributions.

Principe de l'algorithme

Calcul des plus courts chemins issus du sommet a
(sommet, valeur, prédécesseur)
provisoires
(d, 4, b) (e, 12, c)
calculés
(a, 0, /) (b, 1, a) (c, 4, a)

Pour chaque sommet x, on veut calculer la valeur minimum m(x) des chemins allant du sommet de départ A à x. Pour le routage, on veut déterminer le prédécesseur de x, noté p(x), sur un chemin de valeur m(x). Ceci permet de « tracer », en le « remontant », le meilleur chemin du sommet de départ A à n'importe quel sommet x.

L'algorithme de Dijkstra repose sur le principe d'« exploration à partir du meilleur », c'est à dire du meilleur prédécesseur visité.

Un exemple de déroulement de l'algorithme de Dijkstra : visionner l'animation Flash (nécessite un plug-in Adobe Flash).

Un sommet x est dit visité si au moins un chemin de A à x a été évalué ; un sommet visité possède des valeurs m(x), p(x) provisoires : m(x) est la valeur minimum des chemins de A à x déjà évalués, et p(x) le prédécesseur correspondant.

Un sommet est dit calculé s'il est visité et si l'on sait que ses valeurs m(x), p(x) sont définitives (et correctes).

Initialement, bien sûr, le sommet A est calculé (avec m(A) = 0, p(A) non défini puisque A est le point de départ des chemins), et chaque successeur x de A est visité, avec m(x) = v(A, x), p(x) = A.

Le principe d'« exploration à partir du meilleur » consiste à chercher, parmi les sommets visités non encore calculés, un sommet dont la valeur m(x) est minimum. On peut alors démontrer que, pour un tel

sommet, les valeurs « provisoires » m(x), p(x) sont définitives (et correctes). Cette démonstration utilise explicitement le fait que les valeurs sont non négatives.

On marque donc x comme calculé et on « prolonge » l'exploration en examinant chacun des successeurs de x : chaque successeur y non encore visité devient visité, avec m(y) = m(x) + v(y, x), p(y) = x tandis que, pour chaque successeur y déjà visité, on effectue la mise à jour m(y) = min(m(y), m(x) + v(x, y)) (et p(y) = p(x) si m(y) est mis à jour). L'exploration s'arrête lorsque tous les sommets visités sont calculés (les sommets non visités sont inaccessibles, on peut considérer qu'ils ont une valeur m(x) = +∞).

L'algorithme peut s'écrire de cette façon :

```
/* Initialisations */
calculés = {A} ;
m(A) = 0 ;
provisoires = ∅ ;
pour tout sommet y successeur de A
   provisoires = provisoires
```

∪

```
 {y} ;
   m(y) = v(A,y) ;
   p(y) = A ;
fpour
/* Itérations */
tant que provisoires ≠ ∅
   soit x ∈ provisoires tel que m(x) est minimum sur provisoires ;
   calculés = calculés
```

∪

```
 {x} ; provisoires = provisoires
```

-

```
 {x} ;
   pour tout sommet y successeur de x
      cas
         y ∈ calculés : rien
         y ∈ provisoires :
            si m(x) + v(x,y) < m(y)
               m(y) = m(x) + v(x,y) ;
               p(y) = x ;
            fsi
         autrement :
            provisoires = provisoires
```

∪

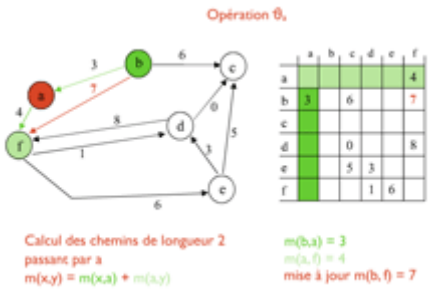
```
 {y} ;
            m(y) = m(x) + v(x,y) ;
            p(y) = x ;
      fcas
   fpour
ftantque
/* les sommets n'appartenant pas à calculés sont inaccessibles */
```

La complexité de l'algorithme est de l'ordre de n^2, mais cette complexité peut être facilement réduite à n+m, où m désigne le nombre d'arcs du graphe, grâce à une structure de données sophistiquée.

3. Le magnifique algorithme général de Roy-Warshall-Floyd

Cet algorithme est dû à R. Floyd, qui l'a publié en 1962 (« Algorithm 97 : Shortest Path », Comm. of the ACM, 5, 1962, p. 345). Il s'agit en fait d'une généralisation au cas des graphes valués d'un algorithme de calcul de fermeture transitive d'un graphe, trouvé à peu près simultanément en France par B. Roy (« Transitivité et connexité », CRAS 249, 1959, pp. 216-218) et aux États-Unis par S.Warshall (« A Theorem on Boolean Matrices », Journal of the ACM, 9(1), 1962, pp. 11-12). Il est bien difficile de savoir si chacun d'eux était au courant des travaux de l'autre, mais, malheureusement, la contribution originale de B. Roy est souvent oubliée dans les références. Nous souhaitons donc contribuer à réparer cette injustice.

Principe de l'algorithme



Il s'agit ici de calculer, pour tout sommet x et tout sommet y, la valeur minimale des chemins de x à y, notée m(x,y). C'est un algorithme applicable dans tous les cas, et dont la complexité est de l'ordre de n^3. De plus, cet algorithme peut gérer le routage, déterminer l'ordre dans lequel les sommets sont visités.

Un exemple de déroulement de l'algorithme de Roy-Warshall-Floyd : visionner l'animation Flash (nécessite un plug-in Adobe Flash).

Supposons que les sommets soient numérotés 1,2, ...,n (ce qui est toujours possible !). Le graphe est composé de l'ensemble fini de ces sommets, et d'un ensemble de couples de sommets, appelé ensemble des arcs. Chaque arc (x,y) est doté d'une valeur notée v(x,y). Initialement, on a, pour tout couple de sommets (x,y), les valeurs m(x,y)=v(x,y) si l'arc (x,y) existe (cet arc correspond à un chemin de longueur 1, c'est-à-dire allant directement de x à y sans passer par un autre sommet - attention de ne pas confondre longueur et valeur), ou m(x,y)=+∞ si l'arc est inexistant.

Une première opération, nommée ϑ_1, va calculer, pour chacun des couples de sommets (x,y), la valeur minimum des chemins de x à y entre celui de longueur 1 (l'arc (x,y)) et celui de longueur 2, sans répétition, passant par le sommet numéro 1, c'est-à-dire le chemin (x,1,y). Cette opération est très simple : elle consiste à calculer, pour tout x et tout y, la valeur m(x,y)=min(m(x,y),m(x,1)+m(1,y)).

Puis une deuxième opération, nommée ϑ_2, va calculer, pour chacun des couples de sommets (x,y), la

valeur minimum des chemins de x à y parmi celui de longueur 1 (l'arc (x,y)), ceux de longueur 2, sans répétition, passant par les sommets numéro 1 ou 2, c'est-à-dire (x,1,y) et (x,2,y), et ceux de longueur 3, sans répétition, passant par les sommets 1 et 2, c'est-à-dire les chemins (x,1,2,y) et (x,2,1,y). Elle consiste à calculer, pour tout x et tout y, la valeur m(x,y)=min(m(x,y),m(x,2)+m(2,y)) où les valeurs m(,) sont celles mises à jour par ϑ_1.

Et ainsi de suite, avec les opérations ϑ_3, ..., ϑ_k, ...,ϑ_n.

On peut voir que, à l'issue de ϑ_k ($1 \leq k \leq n$) chaque valeur m(x,y) est la valeur minimum des chemins de x à y, sans répétition, passant par les sommets intermédiaires de numéro ≤k (ces chemins sont donc de longueur ≤k+1).

Par conséquent, à l'issue de ϑ_n, les valeurs m(x,y) prennent en compte l'ensemble de tous les chemins sans répétition allant de x à y (tous les sommets intermédiaires possibles sont pris en compte).

S'il n'y a pas de circuit absorbant, ces valeurs sont correctes. S'il y a des circuits absorbants (de valeur <0), ils sont détectés, puisque pour chaque sommet x appartenant à un tel circuit, on a m(x,x)<0.

L'algorithme peut s'écrire de la façon suivante :

```
/* initialisation */
pour x depuis 1 jqa n
   pour y depuis 1 jqa n
      si y est successeur de x
         m(x,y)=v(x,y) ;
         sinon m(x,y)=+∞;
      fsi
   fpour
fpour
```

```
/* opérations ϑ */
pour k depuis 1 jqa n
   /* opération ϑ
```

k

```
 */
   pour x depuis 1 jqa n
      v=m(x,k) ;
      pour y depuis 1 jqa n
         w=v+m(k,y) ;
         si w<m(x,y)
            m(x,y)=w ;
         fsi
      fpour
   fpour
fpour
si existe x tel que m(x,x)<0 "il y a des circuits absorbants" fsi
```

Et le tracé des chemins ?

Pour connaître l'ordre dans lequel les sommets sont visités, chaque sommet x est muni d'une table de

routage R(x), qui associe à chaque sommet y la valeur R(x)[y]=successeur de x sur le meilleur chemin menant vers y (ou 0 s'il n'y a pas de chemin de x vers y).

Ces tables sont mises à jour par les opérations ϑ, et restent cohérentes avec les valeurs successives des m(x,y). Initialement, R[x](y) vaut y si y est successeur de x, et vaut 0 sinon (car seuls les arcs, ou chemins de longueur 1, ont été pris en compte à ce stade). Considérons ensuite une opération ϑ_k (avec $1 \leq k \leq n$). Si m(x,y) est modifié par cette opération, cela veut dire que le meilleur chemin de x à y devient un chemin passant par le sommet numéro k. Donc, pour aller de x vers y en suivant ce chemin, il faut se diriger vers le sommet k, et donc suivre le meilleur chemin de x à k. Ceci est indiqué en attribuant R(x)[y]=R(x)[k]. Il suffit d'ajouter ces mises à jour dans le texte de l'algorithme, comme montré ci-après :

```
/* initialisation */
pour x depuis 1 jqa n
   pour y depuis 1 jqa n
      si y est successeur de x
         m(x,y)=v(x,y) ;
         R(x)[y]=y ;
         sinon m(x,y)=+∞; R[x](y)=0 ;
      fsi
   fpour
fpour
```

```
/* opérations ϑ */
pour k depuis 1 jqa n
   /* opération ϑ
```

k

```
 */
   pour x depuis 1 jqa n
      v=m(x,k) ;
      pour y depuis 1 jqa n
         w=v+m(k,y) ;
         si w<m(x,y)
            m(x,y)=w ;
            R(x)[y]=k ;
         fsi
      fpour
   fpour
fpour
si existe x tel que m(x,x)<0 "il y a des circuits absorbants" fsi
```

L'algorithme comporte trois boucles imbriquées, chacune exécutant n pas. Il y a donc de l'ordre de n^3 opérations élémentaires. Il a été prouvé que, sur un graphe quelconque, cette complexité est optimale.

Le texte de l'algorithme montre une possible amélioration : dans l'opération ϑ_k, si on constate que $v(x,k)=+\infty$, alors il est inutile de lancer la boucle sur y, puisque, quel que soit y, $v(x,k)+v(k,y)=+\infty$ qui ne peut pas être inférieur à v(x,y). D'autres améliorations seraient possibles, par exemple, si l'on sait qu'un sommet y est sans successeur, l'opération ϑ_y ne provoquera elle non plus aucune mise à jour, on peut donc l'omettre.

Tags

- Algorithme
- Réseau