Optimisation en informatique – RCP 104

Cédric Bentz

EPN 5 (Informatique), CNAM, Paris

(Contribution aux supports : Agnès Plateau)

Objectif des premières séances de l'UE

Donner un aperçu global des notions importantes du cours à travers des problèmes de base en optimisation dans les réseaux :

- modélisation de réseaux par des graphes,
- illustration pratique de cette modélisation, notamment sur des problèmes de routage (acheminement de flux de données sous contraintes de bandes passantes, tables de routage, etc.),
- problématiques d'optimisation sous-jacentes, catégorisation de leur difficulté, et principales approches de résolution possibles.

Objectif des premières séances de l'UE

Donner un aperçu global des notions importantes du cours à travers des problèmes de base en optimisation dans les réseaux :

- modélisation de réseaux par des graphes,
- illustration pratique de cette modélisation, notamment sur des problèmes de routage (acheminement de flux de données sous contraintes de bandes passantes, tables de routage, etc.),
- problématiques d'optimisation sous-jacentes, catégorisation de leur difficulté, et principales approches de résolution possibles.

Objectif des premières séances de l'UE

Donner un aperçu global des notions importantes du cours à travers des problèmes de base en optimisation dans les réseaux :

- modélisation de réseaux par des graphes,
- illustration pratique de cette modélisation, notamment sur des problèmes de routage (acheminement de flux de données sous contraintes de bandes passantes, tables de routage, etc.),
- problématiques d'optimisation sous-jacentes, catégorisation de leur difficulté, et principales approches de résolution possibles.

Objectif des premières séances de l'UE

Donner un aperçu global des notions importantes du cours à travers des problèmes de base en optimisation dans les réseaux :

- modélisation de réseaux par des graphes,
- illustration pratique de cette modélisation, notamment sur des problèmes de routage (acheminement de flux de données sous contraintes de bandes passantes, tables de routage, etc.),
- problématiques d'optimisation sous-jacentes, catégorisation de leur difficulté, et principales approches de résolution possibles.

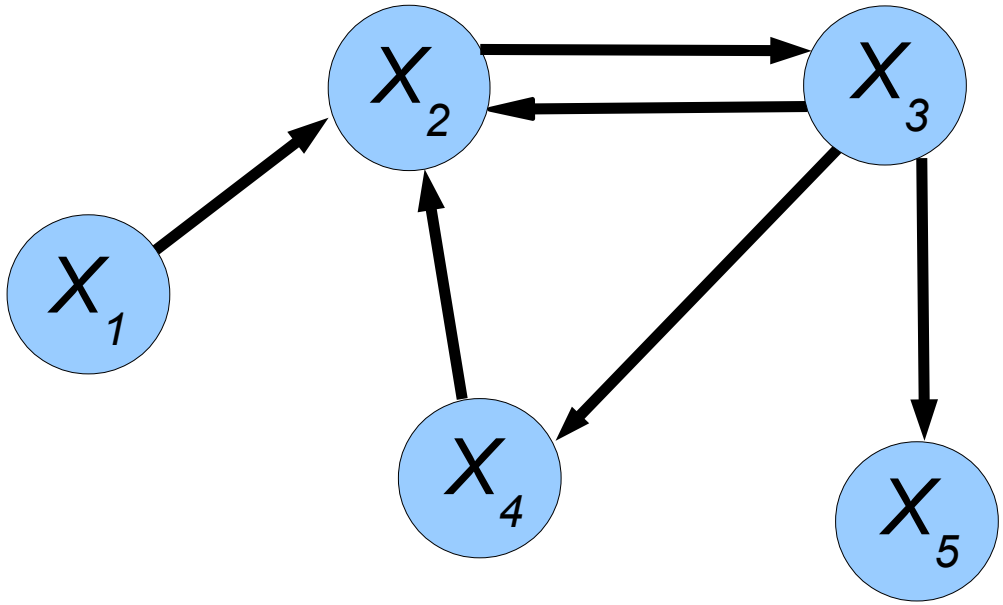
Plan

Graphes (orientés) et réseaux (1/2)

Définition

Un **graphe orienté** *est composé :*

- *de sommets,*
- *d'arcs reliant certaines paires de ses sommets.*

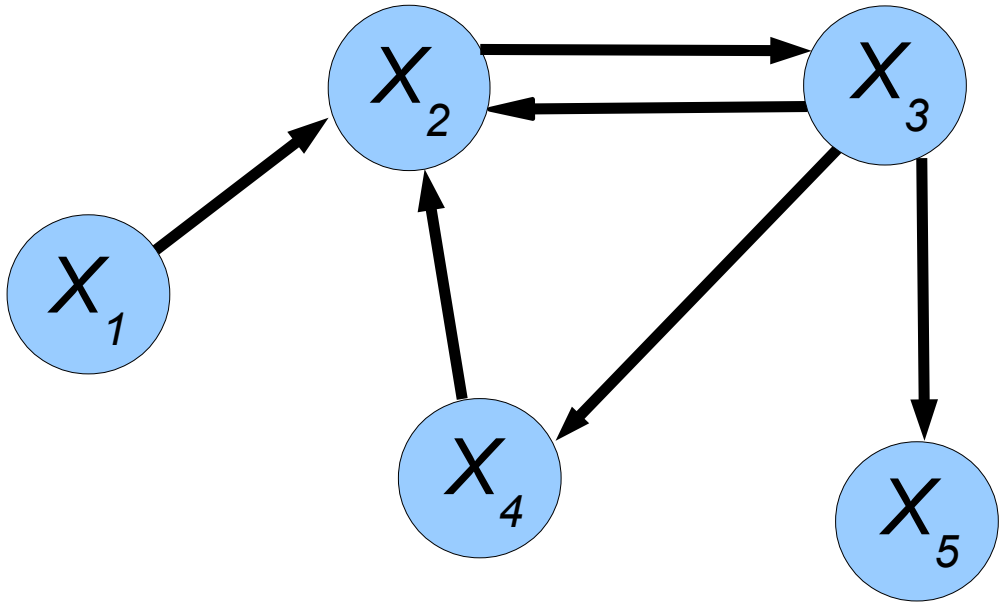


Graphes (orientés) et réseaux (1/2)

Définition

Un **graphe orienté** *est composé :*

- *de sommets,*
- *d'arcs reliant certaines paires de ses sommets.*

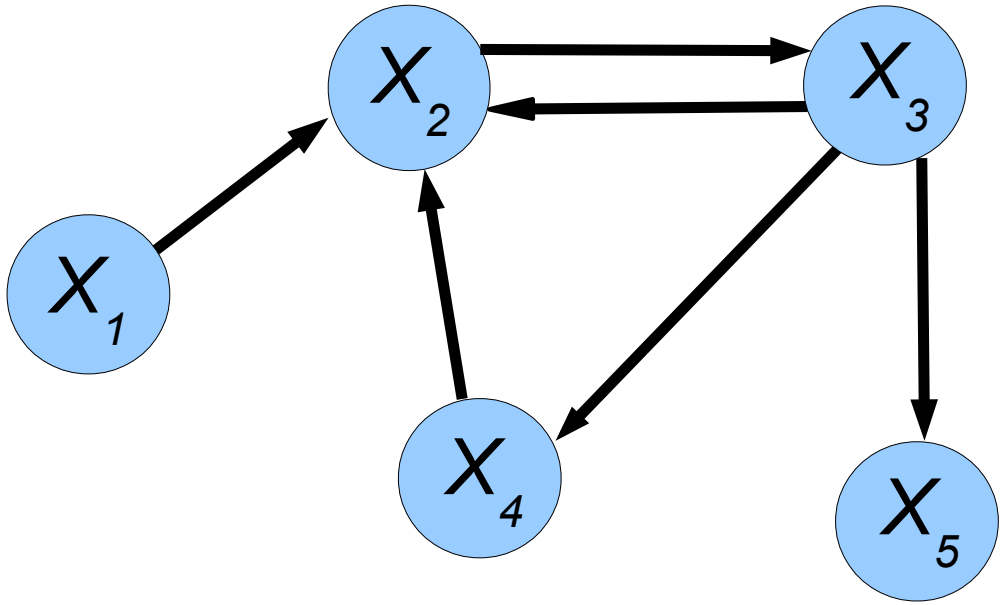


Graphes (orientés) et réseaux (1/2)

Définition

Un **graphe orienté** *est composé :*

- *de sommets,*
- *d'arcs reliant certaines paires de ses sommets.*



Graphes (orientés) et réseaux (2/2)

Remarque

- *Un tel graphe permet naturellement de modéliser un réseau de télécommunication (ou autre) : les sommets sont les nœuds, et les arcs sont les liens/liaisons qui permettent aux noeuds concernés de communiquer entre eux.*
- *Un graphe peut aussi être* **non orienté** *si les sommets/nœuds sont reliés par des liens/liaisons bidirectionnel(le)s (***arêtes***).*
- *Les sommets (ici notés $x_1, \ldots, x_5$) peuvent être représentés indifféremment par des nombres, des lettres, etc.*

Graphes (orientés) et réseaux (2/2)

Remarque

- *Un tel graphe permet naturellement de modéliser un réseau de télécommunication (ou autre) : les sommets sont les nœuds, et les arcs sont les liens/liaisons qui permettent aux noeuds concernés de communiquer entre eux.*
- *Un graphe peut aussi être* **non orienté** *si les sommets/nœuds sont reliés par des liens/liaisons bidirectionnel(le)s (***arêtes***).*
- *Les sommets (ici notés $x_1, \ldots, x_5$) peuvent être représentés indifféremment par des nombres, des lettres, etc.*

Graphes (orientés) et réseaux (2/2)

Remarque

- *Un tel graphe permet naturellement de modéliser un réseau de télécommunication (ou autre) : les sommets sont les nœuds, et les arcs sont les liens/liaisons qui permettent aux noeuds concernés de communiquer entre eux.*
- *Un graphe peut aussi être* **non orienté** *si les sommets/nœuds sont reliés par des liens/liaisons bidirectionnel(le)s (***arêtes***).*
- *Les sommets (ici notés $x_1, \ldots, x_5$) peuvent être représentés indifféremment par des nombres, des lettres, etc.*

Réseau de transport

Définition

Un **réseau de transport** *est un graphe orienté :*

- *contenant deux sommets remarquables, l'un identifié comme* **source** *(s), et l'autre identifié comme* **puits** *(p),*
- *dans lequel à tout arc est associé un entier positif appelé* **capacité de l'arc**.

Remarque

Du point de vue applicatif, la source (nœud émetteur) et le puits (nœud récepteur) sont deux nœuds du réseau qui doivent communiquer, et la capacité d'un arc représente la bande passante du lien/de la liaison associé(e) (par exemple, en Mo/s ou en Go/s).

Réseau de transport

Définition

Un **réseau de transport** *est un graphe orienté :*

- *contenant deux sommets remarquables, l'un identifié comme* **source** *(s), et l'autre identifié comme* **puits** *(p),*
- *dans lequel à tout arc est associé un entier positif appelé* **capacité de l'arc**.

Remarque

Du point de vue applicatif, la source (nœud émetteur) et le puits (nœud récepteur) sont deux nœuds du réseau qui doivent communiquer, et la capacité d'un arc représente la bande passante du lien/de la liaison associé(e) (par exemple, en Mo/s ou en Go/s).

Réseau de transport

Définition

Un **réseau de transport** *est un graphe orienté :*

- *contenant deux sommets remarquables, l'un identifié comme* **source** *(s), et l'autre identifié comme* **puits** *(p),*
- *dans lequel à tout arc est associé un entier positif appelé* **capacité de l'arc**.

Remarque

Du point de vue applicatif, la source (nœud émetteur) et le puits (nœud récepteur) sont deux nœuds du réseau qui doivent communiquer, et la capacité d'un arc représente la bande passante du lien/de la liaison associé(e) (par exemple, en Mo/s ou en Go/s).

Réseau de transport

Définition

Un **réseau de transport** *est un graphe orienté :*

- *contenant deux sommets remarquables, l'un identifié comme* **source** *(s), et l'autre identifié comme* **puits** *(p),*
- *dans lequel à tout arc est associé un entier positif appelé* **capacité de l'arc**.

Remarque

Du point de vue applicatif, la source (nœud émetteur) et le puits (nœud récepteur) sont deux nœuds du réseau qui doivent communiquer, et la capacité d'un arc représente la bande passante du lien/de la liaison associé(e) (par exemple, en Mo/s ou en Go/s).

Flux de données

Définition

La quantité d'information circulant sur un lien/une liaison est exprimée sous la forme d'un **flux** *(de données) entier et positif (représentant par exemple des Mo/s ou des Go/s) sur l'arc associé.*

Définition

Le flux sur tout arc doit vérifier la **contrainte de capacité** *; en d'autres termes, il ne doit pas dépasser la valeur de cette capacité. Un arc est dit* **saturé** *si le flux qui circule dessus est égal à sa capacité (il est alors utilisé au maximum de sa capacité).*

Flux de données

Définition

La quantité d'information circulant sur un lien/une liaison est exprimée sous la forme d'un **flux** *(de données) entier et positif (représentant par exemple des Mo/s ou des Go/s) sur l'arc associé.*

Définition

Le flux sur tout arc doit vérifier la **contrainte de capacité** *; en d'autres termes, il ne doit pas dépasser la valeur de cette capacité. Un arc est dit* **saturé** *si le flux qui circule dessus est égal à sa capacité (il est alors utilisé au maximum de sa capacité).*

Flot dans un réseau de transport

Définition

Un ensemble de flux définis sur les arcs d'un réseau de transport forme un **flot** *sur ce réseau si ces flux vérifient la contrainte de capacité sur chaque arc et la* **loi de conservation des flux**.

Cette loi stipule que, en tout sommet/nœud x du réseau (sauf la source s et le puits p), la somme des flux entrant en x **est égale à** *la somme des flux sortant de x (***pas de perte de données***).*

Flot maximum dans un réseau de transport

Définition

La **valeur** *d'un flot donné est la quantité totale qui circule dans le réseau de transport, c'est-à-dire la somme des flux émis par la source s (qui est égale à la somme des flux reçus par le puits p).*

Problème d'optimisation

Étant donné un réseau de transport, calculer le débit maximum de s vers p (quantité maximum d'information qui peut circuler de s à p, en tenant compte de la bande passante de chaque lien/liaison) revient à **trouver un flot (de valeur) maximum dans ce réseau**.

Flot maximum dans un réseau de transport

Définition

La **valeur** *d'un flot donné est la quantité totale qui circule dans le réseau de transport, c'est-à-dire la somme des flux émis par la source s (qui est égale à la somme des flux reçus par le puits p).*

Problème d'optimisation

Étant donné un réseau de transport, calculer le débit maximum de s vers p (quantité maximum d'information qui peut circuler de s à p, en tenant compte de la bande passante de chaque lien/liaison) revient à **trouver un flot (de valeur) maximum dans ce réseau**.

Flot avec demandes

En pratique, le problème considéré est souvent le suivant :

Définition

Calculer un flot avec demandes = déterminer si/comment une certaine quantité de données peut être acheminée de s à p.

Remarque

Ce problème de flot avec demandes peut se ramener à la détermination d'un flot maximum entre s et p.

Flot avec demandes

En pratique, le problème considéré est souvent le suivant :

Définition

Calculer un flot avec demandes = déterminer si/comment une certaine quantité de données peut être acheminée de s à p.

Remarque

Ce problème de flot avec demandes peut se ramener à la détermination d'un flot maximum entre s et p.

Flot avec demandes

En pratique, le problème considéré est souvent le suivant :

Définition

Calculer un flot avec demandes = déterminer si/comment une certaine quantité de données peut être acheminée de s à p.

Remarque

Ce problème de flot avec demandes peut se ramener à la détermination d'un flot maximum entre s et p.

Illustration sur un exemple

On considère un réseau de transport à 9 nœuds.

La source est notée S, et le puits P.

Les nœuds A à G sont des nœuds intermédiaires.

Les bandes passantes des liens/liaisons (capacités des arcs, exprimées en Mo/s) sont indiquées entre crochets : $[\cdot]$

Les arcs en rouge sont ceux qui partent de la source, et ceux en bleu sont ceux qui arrivent au puits.

Illustration sur un exemple

On considère un réseau de transport à 9 nœuds.

La source est notée S, et le puits P.

Les nœuds A à G sont des nœuds intermédiaires.

Les bandes passantes des liens/liaisons (capacités des arcs, exprimées en Mo/s) sont indiquées entre crochets : $[\cdot]$

Les arcs en rouge sont ceux qui partent de la source, et ceux en bleu sont ceux qui arrivent au puits.

Illustration sur un exemple

On considère un réseau de transport à 9 nœuds.

La source est notée S, et le puits P.

Les nœuds A à G sont des nœuds intermédiaires.

Les bandes passantes des liens/liaisons (capacités des arcs, exprimées en Mo/s) sont indiquées entre crochets : $[\cdot]$

Les arcs en rouge sont ceux qui partent de la source, et ceux en bleu sont ceux qui arrivent au puits.

Illustration sur un exemple

On considère un réseau de transport à 9 nœuds.

La source est notée S, et le puits P.

Les nœuds A à G sont des nœuds intermédiaires.

Les bandes passantes des liens/liaisons (capacités des arcs, exprimées en Mo/s) sont indiquées entre crochets : $[\cdot]$

Les arcs en rouge sont ceux qui partent de la source, et ceux en bleu sont ceux qui arrivent au puits.

Illustration sur un exemple

On considère un réseau de transport à 9 nœuds.

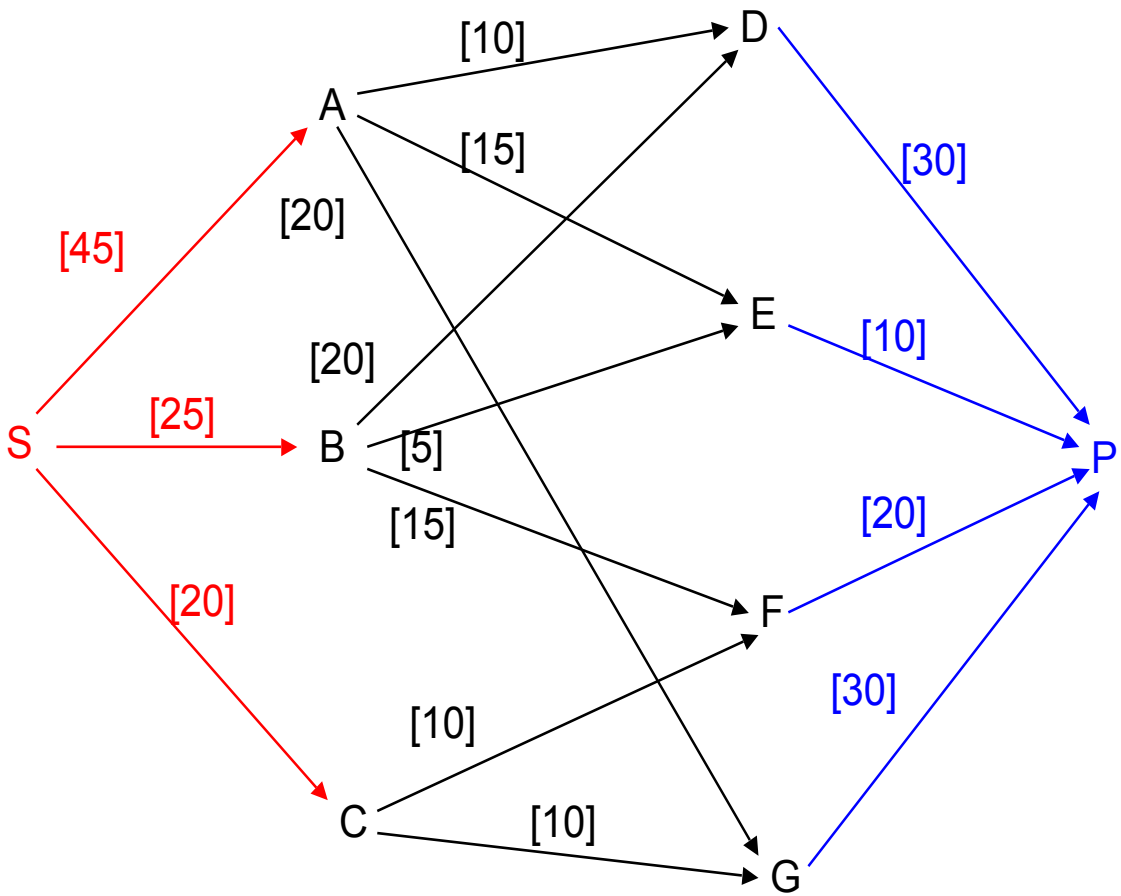
La source est notée S, et le puits P.

Les nœuds A à G sont des nœuds intermédiaires.

Les bandes passantes des liens/liaisons (capacités des arcs, exprimées en Mo/s) sont indiquées entre crochets : $[\cdot]$

Les arcs en rouge sont ceux qui partent de la source, et ceux en bleu sont ceux qui arrivent au puits.

Réseau de transport associé à l'illustration



Construction d'un flot

Définition

Un **chemin** *d'un nœud o vers un nœud d dans un graphe orienté est une suite d'arcs ayant o pour origine et d pour destination, et telle que chaque arc commence au nœud où termine le précédent.*

Remarque

On peut construire un flot d'une source s vers un puits p en sélectionnant itérativement des chemins de s à p, et en acheminant des données sur tous les arcs de chacun de ces chemins.

Construction d'un flot

Définition

Un **chemin** *d'un nœud o vers un nœud d dans un graphe orienté est une suite d'arcs ayant o pour origine et d pour destination, et telle que chaque arc commence au nœud où termine le précédent.*

Remarque

On peut construire un flot d'une source s vers un puits p en sélectionnant itérativement des chemins de s à p, et en acheminant des données sur tous les arcs de chacun de ces chemins.

Flot complet

Définition

Un flot est dit **complet** *s'il existe au moins un arc saturé sur tout chemin allant de la source s au puits p.*

Remarque

Tout flot maximum est nécessairement complet.

L'inverse est-il vrai ?

Flot complet

Définition

Un flot est dit **complet** *s'il existe au moins un arc saturé sur tout chemin allant de la source s au puits p.*

Remarque

Tout flot maximum est nécessairement complet.

L'inverse est-il vrai ?

Comment déterminer un flot complet ?

Procédure gloutonne :

1. Partir du flot NUL (tous les flux sont nuls).
2. Déterminer (par exemple, par un parcours), s'il existe, un chemin allant de la source vers le puits. Si un tel chemin n'existe pas, alors STOP.
3. Faire circuler sur tous les arcs de ce chemin un flux de données jusqu'à saturer un (ou plusieurs) arc(s).
4. Supprimer du réseau (ou interdire lors des parcours suivants) les arcs saturés, et retourner en 2.

Comment déterminer un flot complet ?

Procédure gloutonne :

1. Partir du flot NUL (tous les flux sont nuls).
2. Déterminer (par exemple, par un parcours), s'il existe, un chemin allant de la source vers le puits. Si un tel chemin n'existe pas, alors STOP.
3. Faire circuler sur tous les arcs de ce chemin un flux de données jusqu'à saturer un (ou plusieurs) arc(s).
4. Supprimer du réseau (ou interdire lors des parcours suivants) les arcs saturés, et retourner en 2.

Comment déterminer un flot complet ?

Procédure gloutonne :

1. Partir du flot NUL (tous les flux sont nuls).
2. Déterminer (par exemple, par un parcours), s'il existe, un chemin allant de la source vers le puits. Si un tel chemin n'existe pas, alors STOP.
3. Faire circuler sur tous les arcs de ce chemin un flux de données jusqu'à saturer un (ou plusieurs) arc(s).
4. Supprimer du réseau (ou interdire lors des parcours suivants) les arcs saturés, et retourner en 2.

Comment déterminer un flot complet ?

Procédure gloutonne :

1. Partir du flot NUL (tous les flux sont nuls).
2. Déterminer (par exemple, par un parcours), s'il existe, un chemin allant de la source vers le puits. Si un tel chemin n'existe pas, alors STOP.
3. Faire circuler sur tous les arcs de ce chemin un flux de données jusqu'à saturer un (ou plusieurs) arc(s).
4. Supprimer du réseau (ou interdire lors des parcours suivants) les arcs saturés, et retourner en 2.

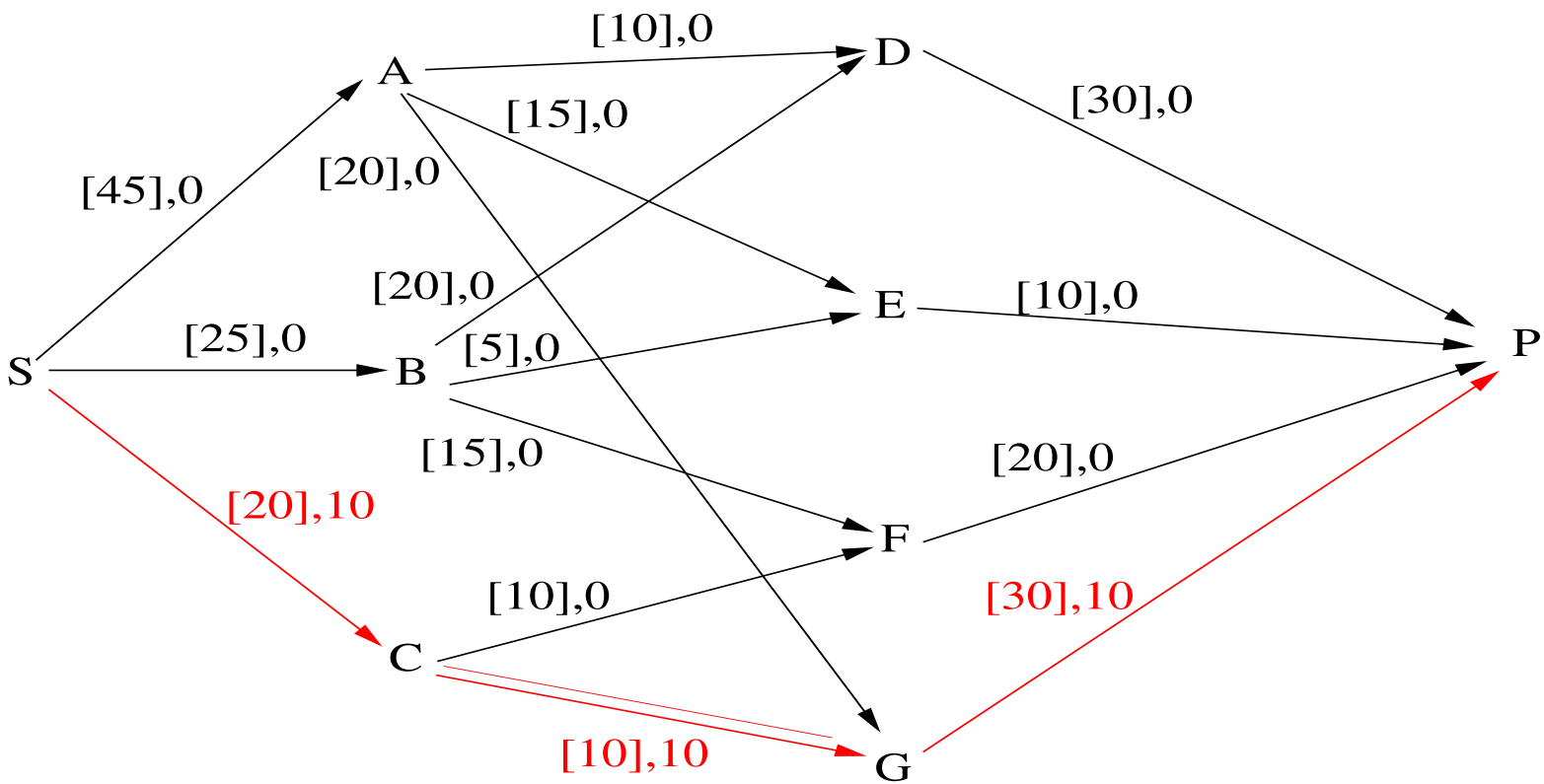
Comment déterminer un flot complet ?

Procédure gloutonne :

1. Partir du flot NUL (tous les flux sont nuls).
2. Déterminer (par exemple, par un parcours), s'il existe, un chemin allant de la source vers le puits. Si un tel chemin n'existe pas, alors STOP.
3. Faire circuler sur tous les arcs de ce chemin un flux de données jusqu'à saturer un (ou plusieurs) arc(s).
4. Supprimer du réseau (ou interdire lors des parcours suivants) les arcs saturés, et retourner en 2.

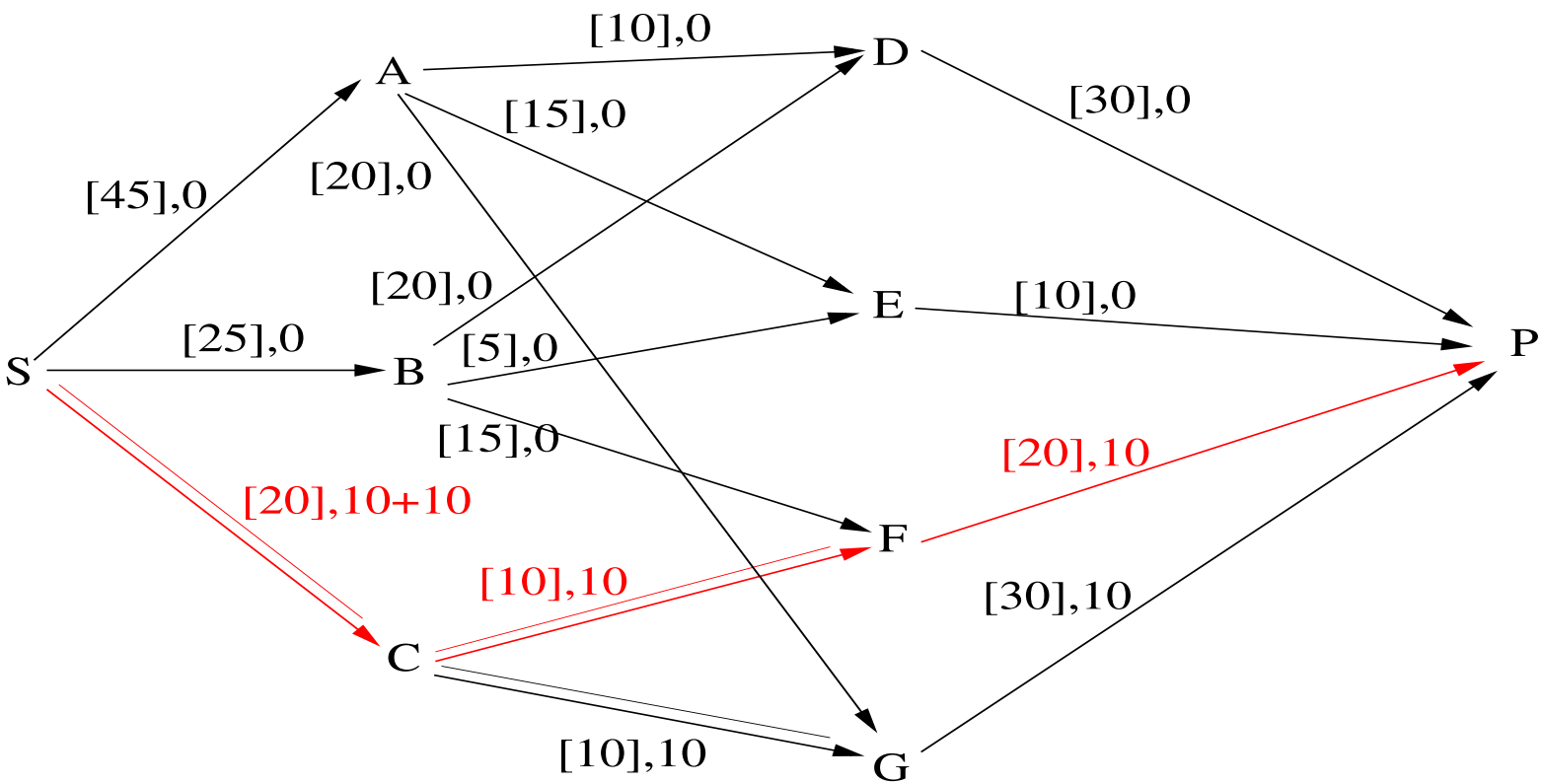
Construction d'un flot complet (1/8)

Sur le chemin (S,C,G,P) : faire circuler un flot de 10 $\Rightarrow$ arc (C, G) saturé (on le double, pour l'interdire).



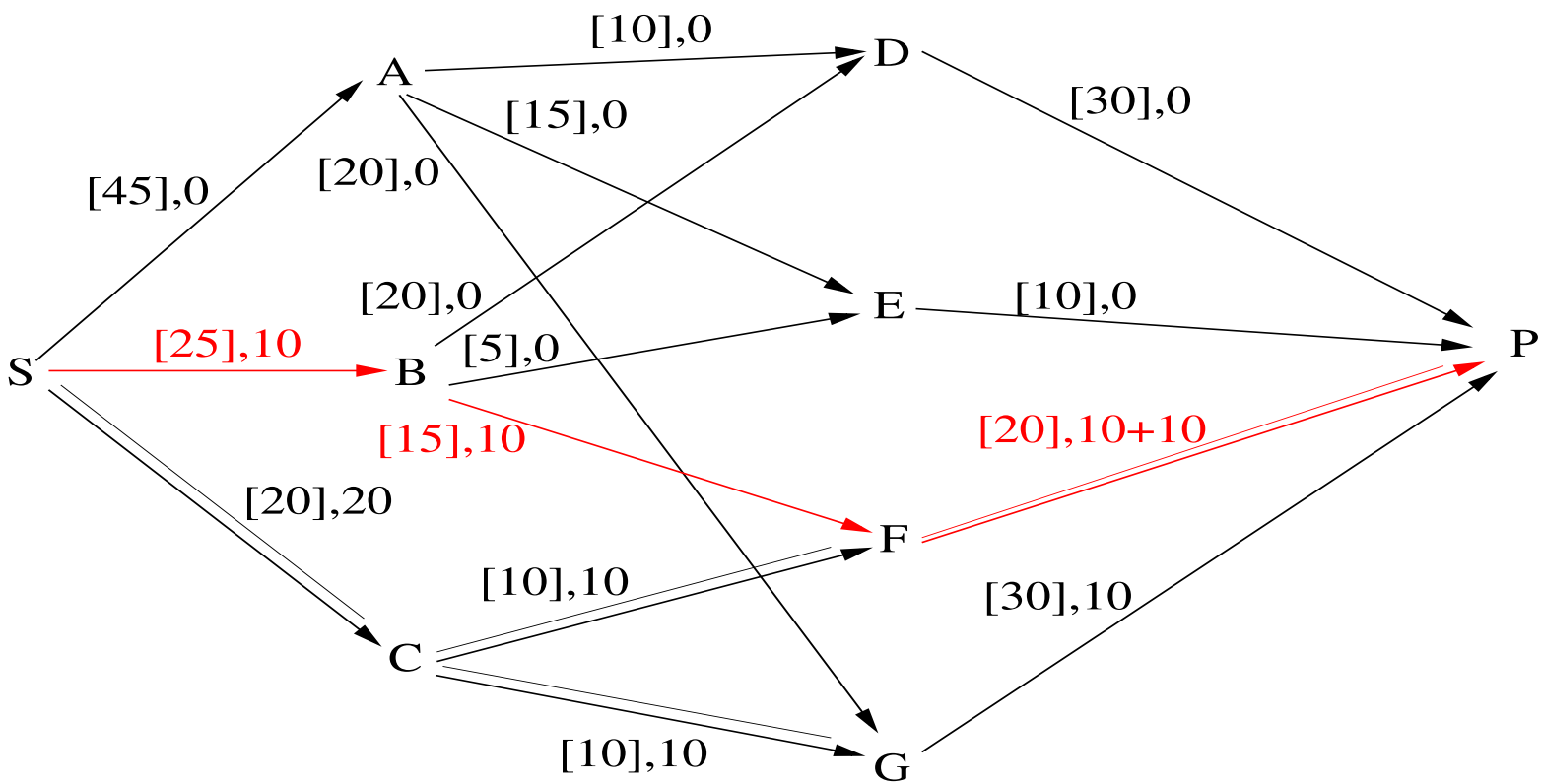
Construction d'un flot complet (2/8)

Sur le chemin (S,C,F,P) : faire circuler un flot de 10 $\Rightarrow$ arcs (S, C) et (C, F) saturés (on les double).



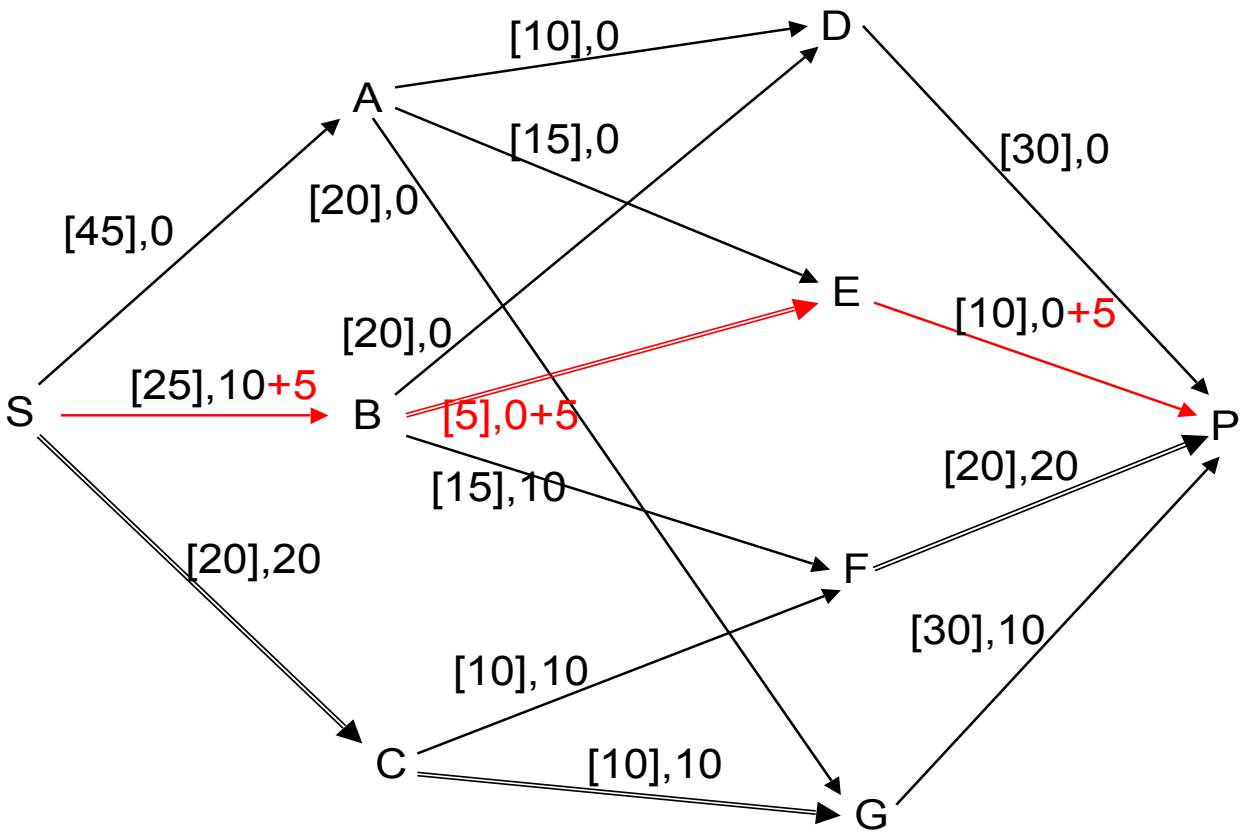
Construction d'un flot complet (3/8)

Sur le chemin (S,B,F,P) : faire circuler un flot de 10 $\Rightarrow$ arc (F, P) saturé (on le double).



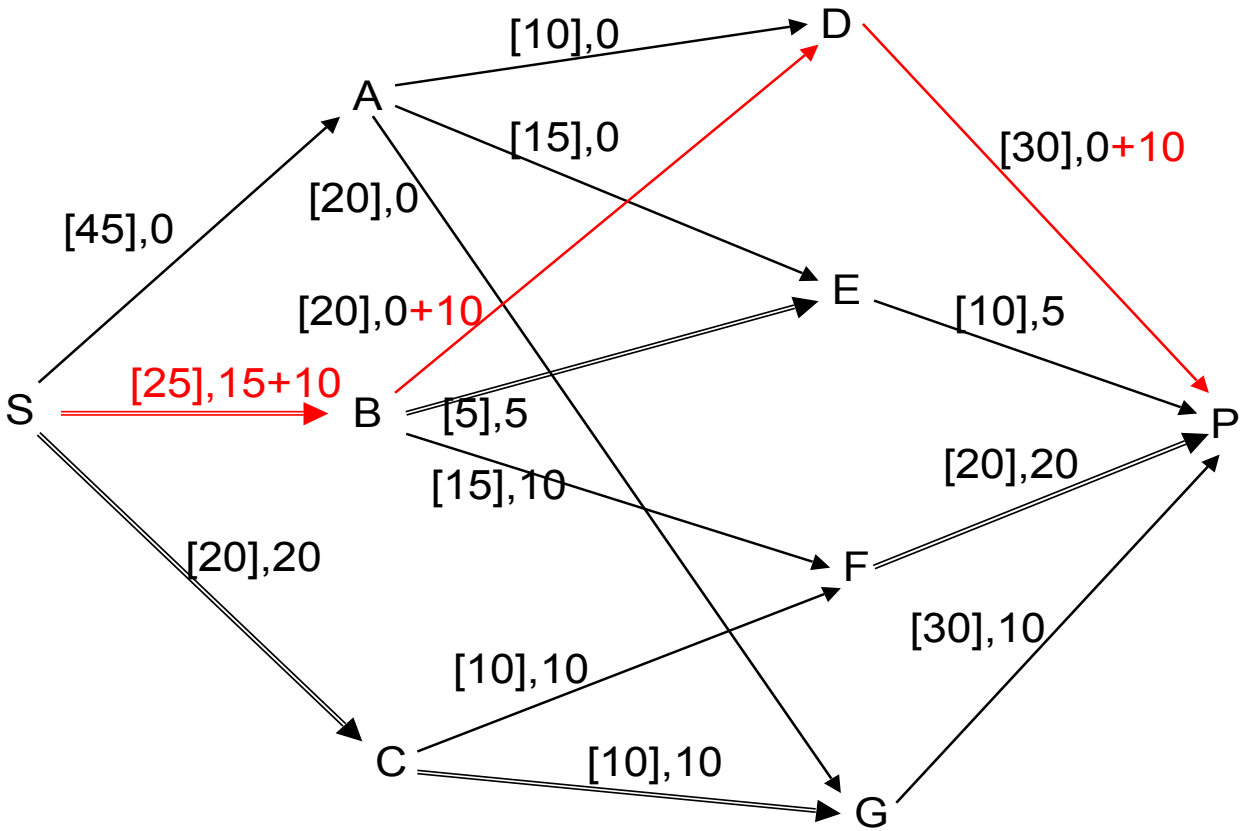
Construction d'un flot complet (4/8)

Sur le chemin (S,B,E,P) : faire circuler un flot de 5 $\Rightarrow$ arc (B, E) saturé (on le double).



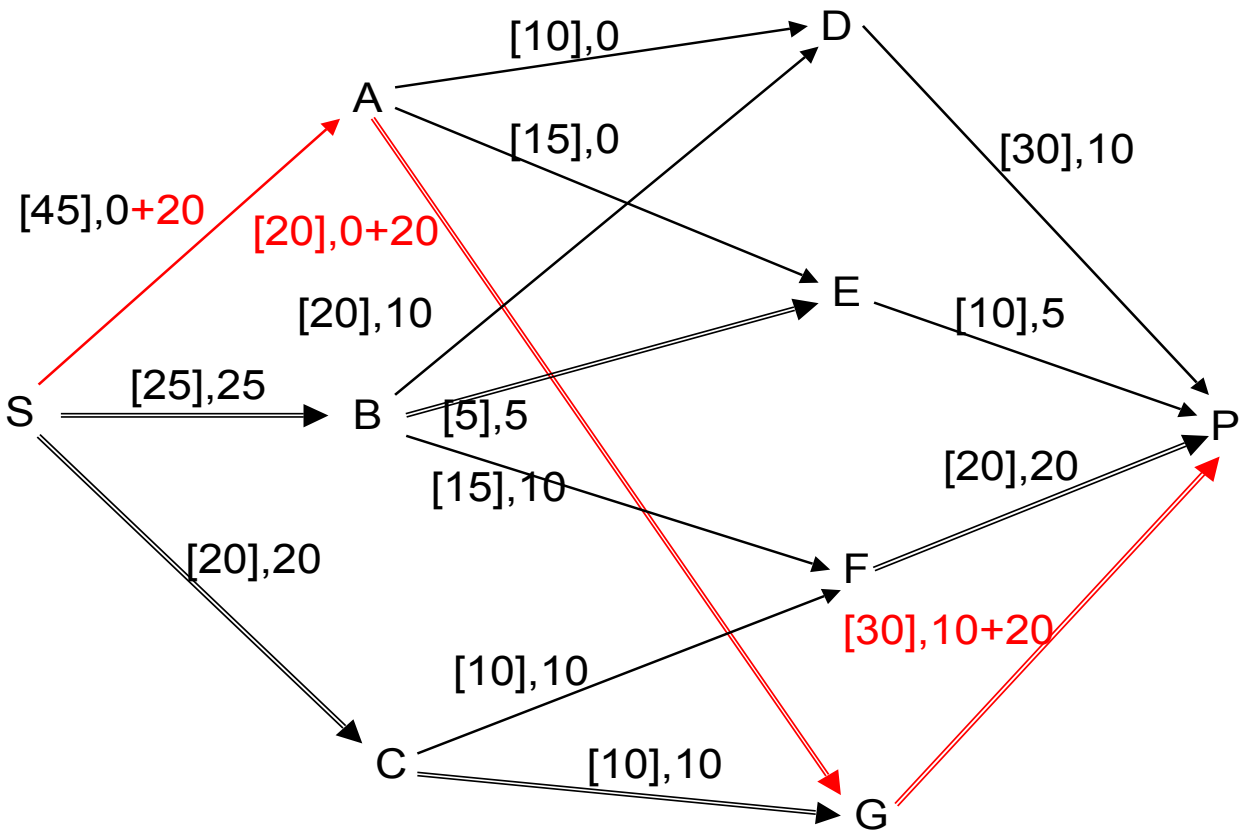
Construction d'un flot complet (5/8)

Sur le chemin (S,B,D,P) : faire circuler un flot de 10 $\Rightarrow$ arc (S, B) saturé (on le double).



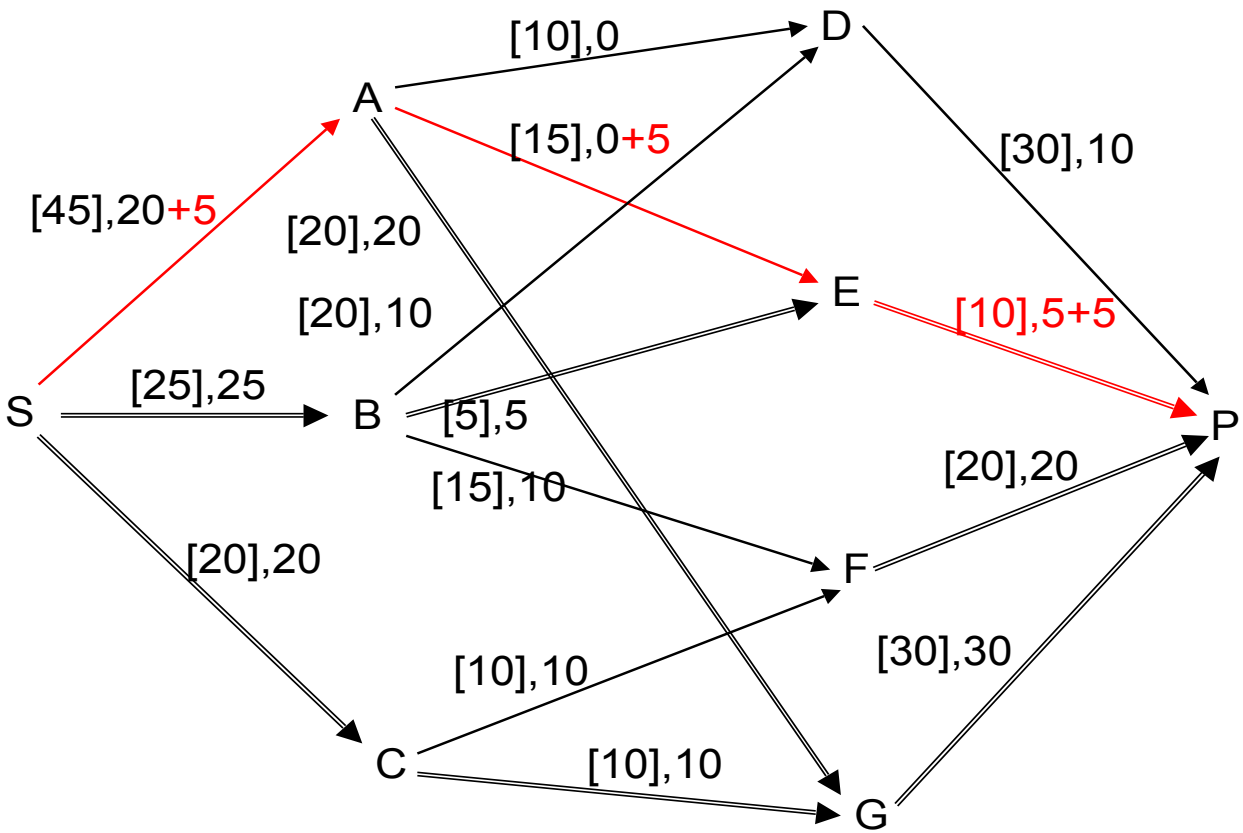
Construction d'un flot complet (6/8)

Sur le chemin (S,A,G,P) : faire circuler un flot de 20 $\Rightarrow$ arcs (A, G) et (G, P) saturés (on les double).



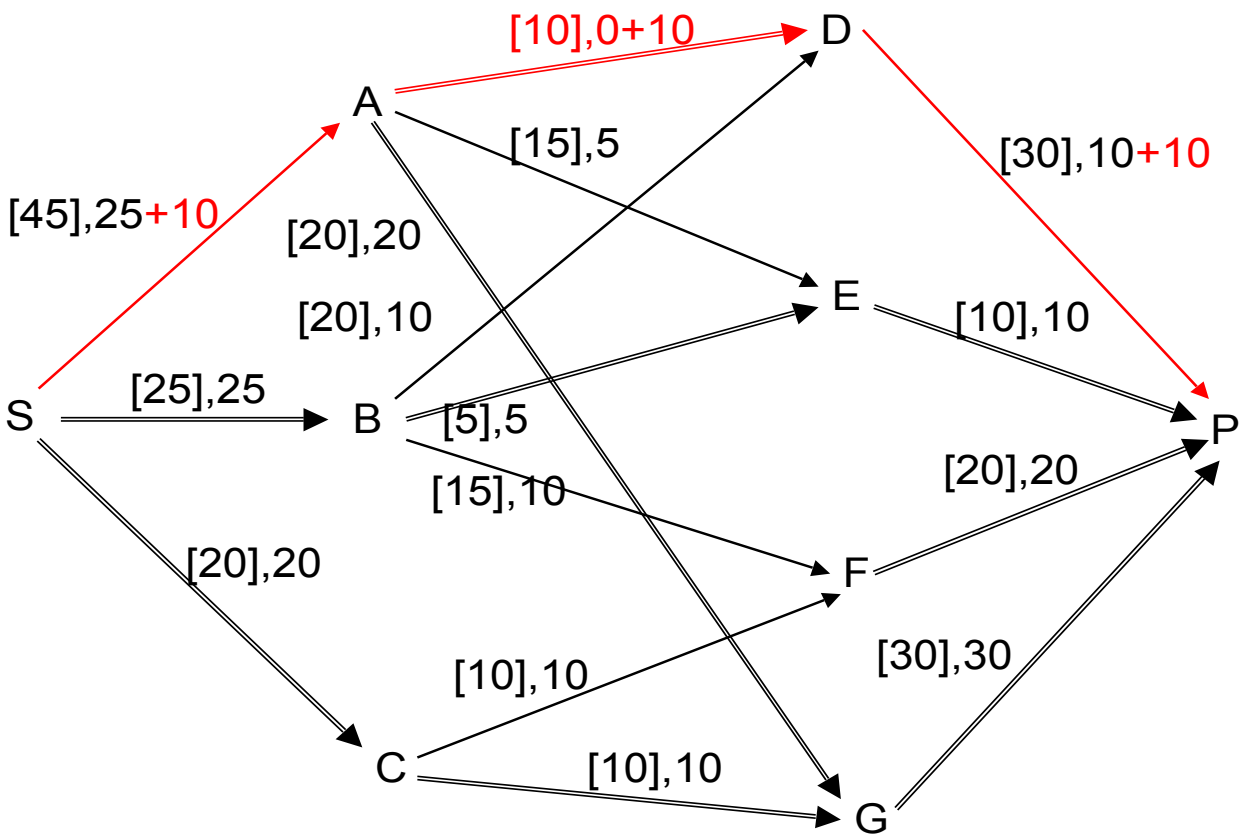
Construction d'un flot complet (7/8)

Sur le chemin (S,A,E,P) : faire circuler un flot de 5 $\Rightarrow$ arc (E, P) saturé (on le double).



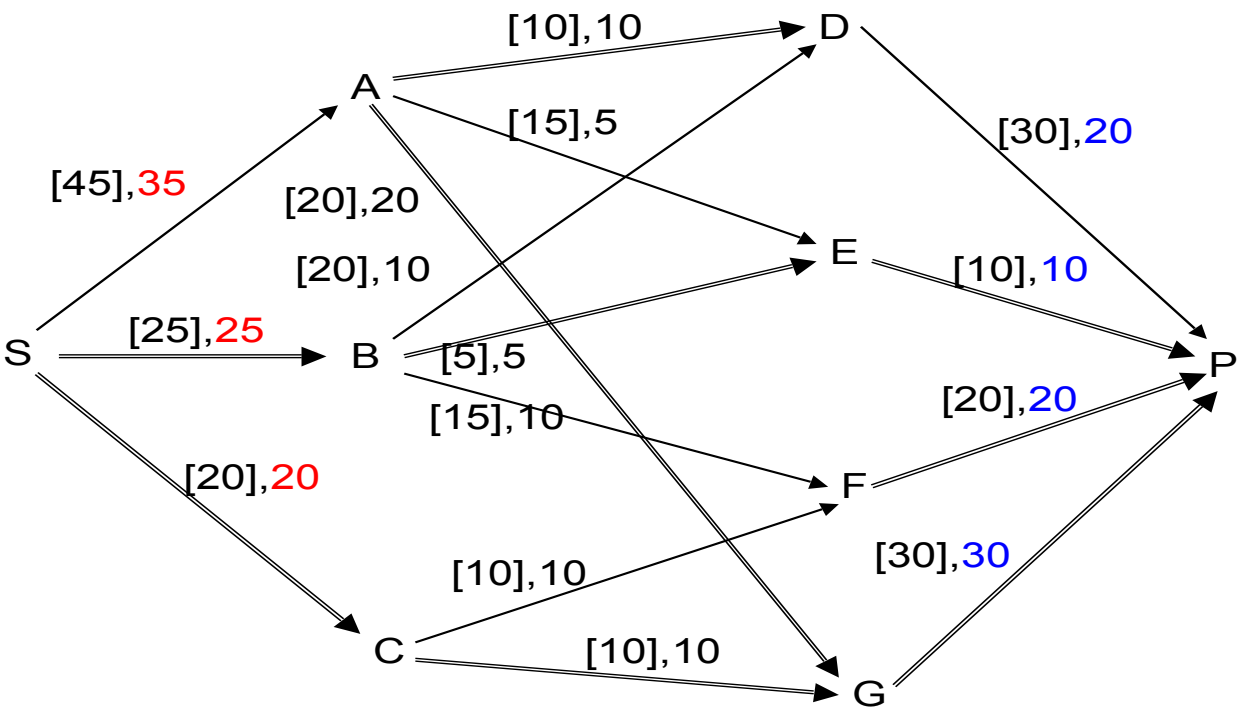
Construction d'un flot complet (8/8)

Sur le chemin (S,A,D,P) : faire circuler un flot de 10 $\Rightarrow$ arc (A, D) saturé (on le double).



Flot complet obtenu

Obtention d'un flot complet, de valeur $35 + 25 + 20$ (somme des flux sortant de la source), soit $20 + 10 + 20 + 30$ (somme des flux arrivant au puits), c'est-à-dire 80. Si on doit acheminer au plus 80 Mo/s de données de S à P, on sait donc comment le faire.



Demande supérieure ?

Si on souhaite acheminer plus de 80 Mo/s, peut-on le faire ?

- Calculer un flot maximum est un problème que l'on peut formuler via la **programmation linéaire** (cf cours suivants).
- Il existe des logiciels (appelés **solveurs**) permettant de résoudre des programmes linéaires $\Rightarrow$ **exemple de GLPK**.
- La valeur d'un flot maximum dans cet exemple est donc 85 : le flot précédent, bien que complet, n'était donc pas maximum !

Demande supérieure ?

Si on souhaite acheminer plus de 80 Mo/s, peut-on le faire ?

- Calculer un flot maximum est un problème que l'on peut formuler via la **programmation linéaire** (cf cours suivants).
- Il existe des logiciels (appelés **solveurs**) permettant de résoudre des programmes linéaires $\Rightarrow$ **exemple de GLPK**.
- La valeur d'un flot maximum dans cet exemple est donc 85 : le flot précédent, bien que complet, n'était donc pas maximum !

Demande supérieure ?

Si on souhaite acheminer plus de 80 Mo/s, peut-on le faire ?

- Calculer un flot maximum est un problème que l'on peut formuler via la **programmation linéaire** (cf cours suivants).
- Il existe des logiciels (appelés **solveurs**) permettant de résoudre des programmes linéaires $\Rightarrow$ **exemple de GLPK**.
- La valeur d'un flot maximum dans cet exemple est donc 85 : le flot précédent, bien que complet, n'était donc pas maximum !

Demande supérieure ?

Si on souhaite acheminer plus de 80 Mo/s, peut-on le faire ?

- Calculer un flot maximum est un problème que l'on peut formuler via la **programmation linéaire** (cf cours suivants).
- Il existe des logiciels (appelés **solveurs**) permettant de résoudre des programmes linéaires $\Rightarrow$ **exemple de GLPK**.
- La valeur d'un flot maximum dans cet exemple est donc 85 : le flot précédent, bien que complet, n'était donc pas maximum !

Flot à coût minimum

En pratique, on souhaite souvent faire circuler des données de s à p en prenant également en compte leur coût total d'acheminement.

Modèle le plus simple = **flot à coût minimum** :

- Coût linéaire/proportionnel à la quantité de données (flux)

$\Rightarrow$ Toute unité d'information (Mo/s) circulant sur chaque arc (x, y) a un coût de $c(x, y)$ euros (**coût unitaire**).

- Peut également se formuler par la programmation linéaire.

Flot à coût minimum

En pratique, on souhaite souvent faire circuler des données de s à p en prenant également en compte leur coût total d'acheminement.

Modèle le plus simple = **flot à coût minimum** :

- Coût linéaire/proportionnel à la quantité de données (flux)

$\Rightarrow$ Toute unité d'information (Mo/s) circulant sur chaque arc (x, y) a un coût de $c(x, y)$ euros (**coût unitaire**).

- Peut également se formuler par la programmation linéaire.

Flot à coût minimum

En pratique, on souhaite souvent faire circuler des données de s à p en prenant également en compte leur coût total d'acheminement.

Modèle le plus simple = **flot à coût minimum** :

- Coût linéaire/proportionnel à la quantité de données (flux)

$\Rightarrow$ Toute unité d'information (Mo/s) circulant sur chaque arc (x, y) a un coût de $c(x, y)$ euros (**coût unitaire**).

- Peut également se formuler par la programmation linéaire.

Flot à coût minimum

En pratique, on souhaite souvent faire circuler des données de s à p en prenant également en compte leur coût total d'acheminement.

Modèle le plus simple = **flot à coût minimum** :

- Coût linéaire/proportionnel à la quantité de données (flux)

$\Rightarrow$ Toute unité d'information (Mo/s) circulant sur chaque arc (x, y) a un coût de $c(x, y)$ euros (**coût unitaire**).

- Peut également se formuler par la programmation linéaire.

Flot à coût minimum

En pratique, on souhaite souvent faire circuler des données de s à p en prenant également en compte leur coût total d'acheminement.

Modèle le plus simple = **flot à coût minimum** :

- Coût linéaire/proportionnel à la quantité de données (flux)

$\Rightarrow$ Toute unité d'information (Mo/s) circulant sur chaque arc (x, y) a un coût de $c(x, y)$ euros (**coût unitaire**).

- Peut également se formuler par la programmation linéaire.

Illustration sur un exemple

On suppose ici que le coût unitaire de tout arc qui part de la source est 10, que celui de tout arc qui arrive au puits est 30, et que celui de tout autre arc est 20.

Le flot complet précédent, de valeur 80, a donc un coût total de :

$$10 \times (35 + 25 + 20) +$$
$$20 \times (10 + 5 + 20 + 10 + 5 + 10 + 10 + 10) +$$
$$30 \times (20 + 10 + 20 + 30)$$
$$= \mathbf{4800}$$

Illustration sur un exemple

On suppose ici que le coût unitaire de tout arc qui part de la source est 10, que celui de tout arc qui arrive au puits est 30, et que celui de tout autre arc est 20.

Le flot complet précédent, de valeur 80, a donc un coût total de :

$$10 \times (35 + 25 + 20) +$$
$$20 \times (10 + 5 + 20 + 10 + 5 + 10 + 10 + 10) +$$
$$30 \times (20 + 10 + 20 + 30)$$
$$= \mathbf{4800}$$

Multiflot

Du point de vue applicatif, un flot impose un unique nœud émetteur (s) et un unique nœud récepteur (p).

En pratique, il est souvent plus réaliste de considérer plusieurs paires de nœuds devant communiquer : (s_1, p_1), (s_2, p_2), ...

Tout flux de données partant de s_i doit donc atteindre p_i, $i \geq 1$.

$\Rightarrow$ **Multiflot** (**biflot** si 2 paires de nœuds) dans un réseau.

Procédure gloutonne (heuristique) : on peut calculer un biflot en calculant d'abord un flot de s_1 à p_1, puis un flot de s_2 à p_2 (après mise à jour des capacités). (Ou l'inverse !)

Multiflot

Du point de vue applicatif, un flot impose un unique nœud émetteur (s) et un unique nœud récepteur (p).

En pratique, il est souvent plus réaliste de considérer plusieurs paires de nœuds devant communiquer : (s_1, p_1), (s_2, p_2), ...

Tout flux de données partant de s_i doit donc atteindre p_i, $i \geq 1$.

$\Rightarrow$ **Multiflot** (**biflot** si 2 paires de nœuds) dans un réseau.

Procédure gloutonne (heuristique) : on peut calculer un biflot en calculant d'abord un flot de s_1 à p_1, puis un flot de s_2 à p_2 (après mise à jour des capacités). (Ou l'inverse !)

Multiflot

Du point de vue applicatif, un flot impose un unique nœud émetteur (s) et un unique nœud récepteur (p).

En pratique, il est souvent plus réaliste de considérer plusieurs paires de nœuds devant communiquer : (s_1, p_1), (s_2, p_2), ...

Tout flux de données partant de s_i doit donc atteindre p_i, $i \geq 1$.

$\Rightarrow$ **Multiflot** (**biflot** si 2 paires de nœuds) dans un réseau.

Procédure gloutonne (heuristique) : on peut calculer un biflot en calculant d'abord un flot de s_1 à p_1, puis un flot de s_2 à p_2 (après mise à jour des capacités). (Ou l'inverse !)

Multiflot

Du point de vue applicatif, un flot impose un unique nœud émetteur (s) et un unique nœud récepteur (p).

En pratique, il est souvent plus réaliste de considérer plusieurs paires de nœuds devant communiquer : (s_1, p_1), (s_2, p_2), ...

Tout flux de données partant de s_i doit donc atteindre p_i, $i \geq 1$.

$\Rightarrow$ **Multiflot** (**biflot** si 2 paires de nœuds) dans un réseau.

Procédure gloutonne (heuristique) : on peut calculer un biflot en calculant d'abord un flot de s_1 à p_1, puis un flot de s_2 à p_2 (après mise à jour des capacités). (Ou l'inverse !)

Multiflot

Du point de vue applicatif, un flot impose un unique nœud émetteur (s) et un unique nœud récepteur (p).

En pratique, il est souvent plus réaliste de considérer plusieurs paires de nœuds devant communiquer : (s_1, p_1), (s_2, p_2), ...

Tout flux de données partant de s_i doit donc atteindre p_i, $i \geq 1$.

$\Rightarrow$ **Multiflot** (**biflot** si 2 paires de nœuds) dans un réseau.

Procédure gloutonne (heuristique) : on peut calculer un biflot en calculant d'abord un flot de s_1 à p_1, puis un flot de s_2 à p_2 (après mise à jour des capacités). (Ou l'inverse !)

Illustration sur un exemple de biflot (1/4)

On considère le réseau (graphe orienté muni de capacités) suivant, avec deux paires (s_1, p_1) et (s_2, p_2) :

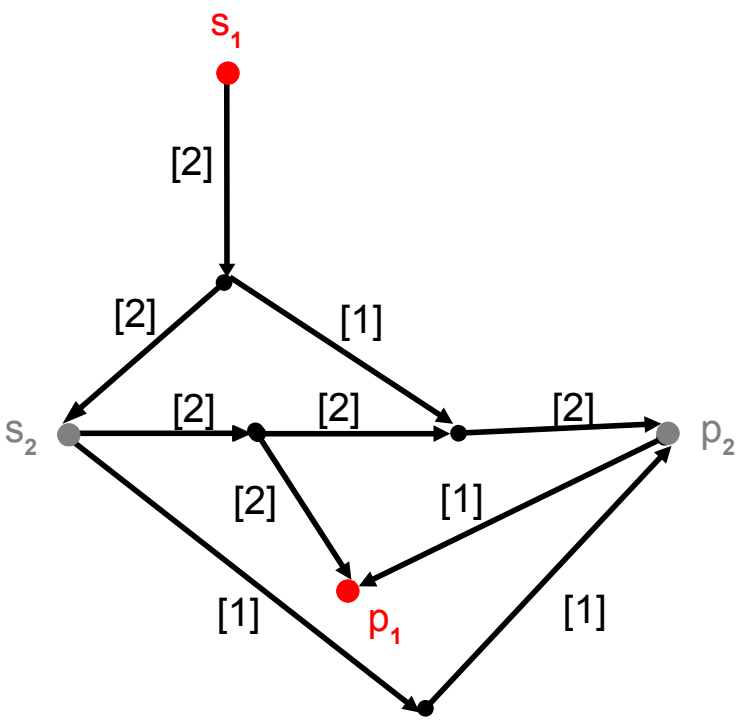


Illustration sur un exemple de biflot (2/4)

Premier biflot (de valeur 3) : flot (de valeur 2) de s_1 à p_1 et mise à jour des capacités, puis flot (de valeur 1) de s_2 à p_2.

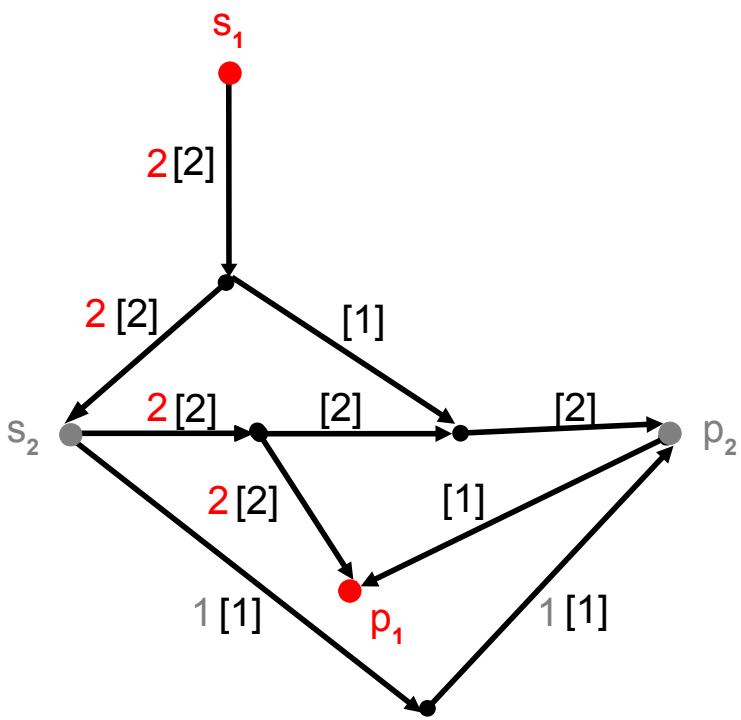


Illustration sur un exemple de biflot (3/4)

Second biflot (de valeur 3) : flot (de valeur 3) de s_2 à p_2 et mise à jour des capacités, puis aucun flot de s_1 à p_1.

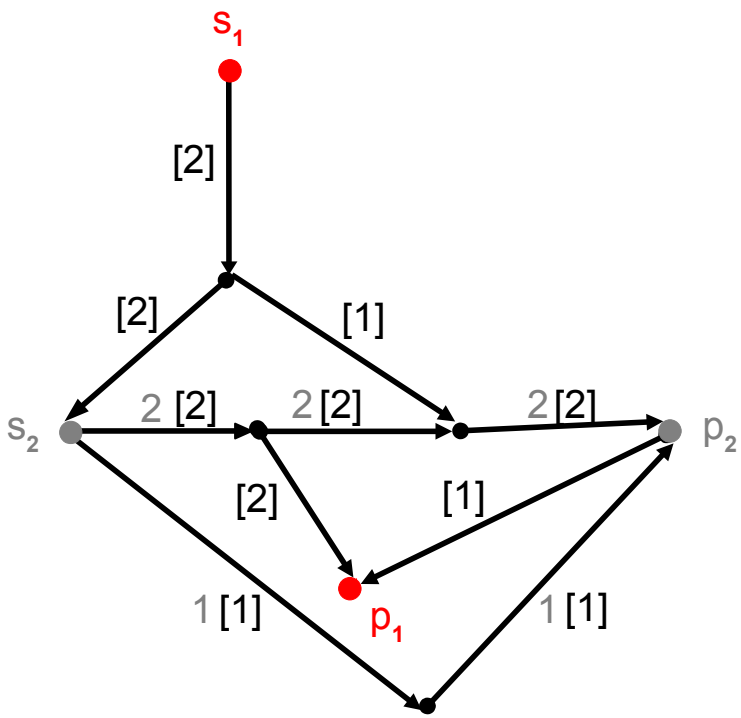
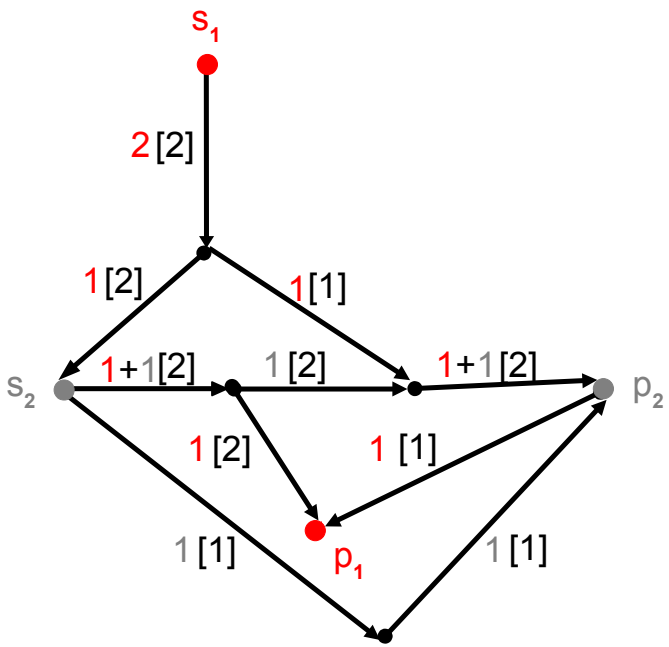


Illustration sur un exemple de biflot (4/4)

Biflot de valeur max. (égale à 4) : de nouveau, on peut l'obtenir à l'aide d'un solveur (cf GLPK). On en déduit que les solutions précédentes, simples à calculer, sont approchées (heuristiques).



Difficultés liées à la résolution de problèmes (1/9)

Concrètement, calculer un flot maximum (ou avec demandes) est un problème **facile**, du point de vue informatique.

Calculer un biflot maximum ne l'est pas, en général.

Cela se traduit par le fait qu'il existe des méthodes de résolution **efficaces** pour le premier problème, mais pas pour le second.

Difficultés liées à la résolution de problèmes (1/9)

Concrètement, calculer un flot maximum (ou avec demandes) est un problème **facile**, du point de vue informatique.

Calculer un biflot maximum ne l'est pas, en général.

Cela se traduit par le fait qu'il existe des méthodes de résolution **efficaces** pour le premier problème, mais pas pour le second.

Difficultés liées à la résolution de problèmes (1/9)

Concrètement, calculer un flot maximum (ou avec demandes) est un problème **facile**, du point de vue informatique.

Calculer un biflot maximum ne l'est pas, en général.

Cela se traduit par le fait qu'il existe des méthodes de résolution **efficaces** pour le premier problème, mais pas pour le second.

Difficultés liées à la résolution de problèmes (2/9)

Qu'est-ce qu'un algorithme "efficace" ?

De façon informelle, c'est un algorithme qui utilise un nombre aussi réduit que possible d'étapes de calcul "standard" (comme, par exemple, additions, affectations, etc.) pour résoudre un problème.

En effet, on ne mesure pas l'efficacité d'un algorithme à son temps d'exécution : ce dernier dépend de la machine sur lequel l'algorithme est implémenté puis exécuté, alors que l'efficacité intrinsèque d'un algorithme ne doit justement pas en dépendre.

Difficultés liées à la résolution de problèmes (2/9)

Qu'est-ce qu'un algorithme "efficace" ?

De façon informelle, c'est un algorithme qui utilise un nombre aussi réduit que possible d'étapes de calcul "standard" (comme, par exemple, additions, affectations, etc.) pour résoudre un problème.

En effet, on ne mesure pas l'efficacité d'un algorithme à son temps d'exécution : ce dernier dépend de la machine sur lequel l'algorithme est implémenté puis exécuté, alors que l'efficacité intrinsèque d'un algorithme ne doit justement pas en dépendre.

Difficultés liées à la résolution de problèmes (2/9)

Qu'est-ce qu'un algorithme "efficace" ?

De façon informelle, c'est un algorithme qui utilise un nombre aussi réduit que possible d'étapes de calcul "standard" (comme, par exemple, additions, affectations, etc.) pour résoudre un problème.

En effet, on ne mesure pas l'efficacité d'un algorithme à son temps d'exécution : ce dernier dépend de la machine sur lequel l'algorithme est implémenté puis exécuté, alors que l'efficacité intrinsèque d'un algorithme ne doit justement pas en dépendre.

Difficultés liées à la résolution de problèmes (3/9)

En général, ce nombre d'étapes de calcul augmente avec la taille des données à traiter.

De façon un peu plus précise, un algorithme efficace est donc un algorithme dont le nombre d'étapes de calcul n'augmente "pas trop" en fonction de la taille des données d'entrée.

Par exemple, pour des données de taille $n = 100$, un nombre d'étapes de calcul égal à n^2 donne 10 000 étapes, alors que 2^n donne $2^{100} \simeq 10^{30}$ étapes.

Sur une machine avec un CPU cadencé à 1 Ghz, qui effectue $\simeq 10^9$ étapes de calcul par seconde, il faudrait environ 30 000 milliards d'années pour effectuer ces 2^{100} étapes.

Difficultés liées à la résolution de problèmes (3/9)

En général, ce nombre d'étapes de calcul augmente avec la taille des données à traiter.

De façon un peu plus précise, un algorithme efficace est donc un algorithme dont le nombre d'étapes de calcul n'augmente "pas trop" en fonction de la taille des données d'entrée.

Par exemple, pour des données de taille $n = 100$, un nombre d'étapes de calcul égal à n^2 donne 10 000 étapes, alors que 2^n donne $2^{100} \simeq 10^{30}$ étapes.

Sur une machine avec un CPU cadencé à 1 Ghz, qui effectue $\simeq 10^9$ étapes de calcul par seconde, il faudrait environ 30 000 milliards d'années pour effectuer ces 2^{100} étapes.

Difficultés liées à la résolution de problèmes (3/9)

En général, ce nombre d'étapes de calcul augmente avec la taille des données à traiter.

De façon un peu plus précise, un algorithme efficace est donc un algorithme dont le nombre d'étapes de calcul n'augmente "pas trop" en fonction de la taille des données d'entrée.

Par exemple, pour des données de taille $n = 100$, un nombre d'étapes de calcul égal à n^2 donne 10 000 étapes, alors que 2^n donne $2^{100} \simeq 10^{30}$ étapes.

Sur une machine avec un CPU cadencé à 1 Ghz, qui effectue $\simeq 10^9$ étapes de calcul par seconde, il faudrait environ 30 000 milliards d'années pour effectuer ces 2^{100} étapes.

Difficultés liées à la résolution de problèmes (3/9)

En général, ce nombre d'étapes de calcul augmente avec la taille des données à traiter.

De façon un peu plus précise, un algorithme efficace est donc un algorithme dont le nombre d'étapes de calcul n'augmente "pas trop" en fonction de la taille des données d'entrée.

Par exemple, pour des données de taille $n = 100$, un nombre d'étapes de calcul égal à n^2 donne 10 000 étapes, alors que 2^n donne $2^{100} \simeq 10^{30}$ étapes.

Sur une machine avec un CPU cadencé à 1 Ghz, qui effectue $\simeq 10^9$ étapes de calcul par seconde, il faudrait environ 30 000 milliards d'années pour effectuer ces 2^{100} étapes.

Difficultés liées à la résolution de problèmes (4/9)

Parmi les principaux problèmes d'optimisation dans les réseaux que l'on étudiera, on en compte essentiellement deux types :

1. ceux tels qu'on peut, de façon efficace, vérifier qu'il existe une solution de valeur plus grande/petite qu'une valeur donnée (une telle solution étant fournie, si nécessaire),
2. ceux tels qu'on peut, de façon efficace, décider s'il existe une solution de valeur plus grande/petite qu'une valeur donnée.

Tout problème de type 2 (donc "facile") est également de type 1.

Le problème du biflot maximum (avec flux de données entiers) est de type 1, mais pas de type 2.

Le problème du flot maximum (avec flux de données entiers ou non) est de type 2, et donc de type 1 également.

Difficultés liées à la résolution de problèmes (4/9)

Parmi les principaux problèmes d'optimisation dans les réseaux que l'on étudiera, on en compte essentiellement deux types :

1. ceux tels qu'on peut, de façon efficace, vérifier qu'il existe une solution de valeur plus grande/petite qu'une valeur donnée (une telle solution étant fournie, si nécessaire),
2. ceux tels qu'on peut, de façon efficace, décider s'il existe une solution de valeur plus grande/petite qu'une valeur donnée.

Tout problème de type 2 (donc "facile") est également de type 1.

Le problème du biflot maximum (avec flux de données entiers) est de type 1, mais pas de type 2.

Le problème du flot maximum (avec flux de données entiers ou non) est de type 2, et donc de type 1 également.

Difficultés liées à la résolution de problèmes (4/9)

Parmi les principaux problèmes d'optimisation dans les réseaux que l'on étudiera, on en compte essentiellement deux types :

1. ceux tels qu'on peut, de façon efficace, vérifier qu'il existe une solution de valeur plus grande/petite qu'une valeur donnée (une telle solution étant fournie, si nécessaire),
2. ceux tels qu'on peut, de façon efficace, décider s'il existe une solution de valeur plus grande/petite qu'une valeur donnée.

Tout problème de type 2 (donc "facile") est également de type 1.

Le problème du biflot maximum (avec flux de données entiers) est de type 1, mais pas de type 2.

Le problème du flot maximum (avec flux de données entiers ou non) est de type 2, et donc de type 1 également.

Difficultés liées à la résolution de problèmes (4/9)

Parmi les principaux problèmes d'optimisation dans les réseaux que l'on étudiera, on en compte essentiellement deux types :

1. ceux tels qu'on peut, de façon efficace, vérifier qu'il existe une solution de valeur plus grande/petite qu'une valeur donnée (une telle solution étant fournie, si nécessaire),
2. ceux tels qu'on peut, de façon efficace, décider s'il existe une solution de valeur plus grande/petite qu'une valeur donnée.

Tout problème de type 2 (donc "facile") est également de type 1.

Le problème du biflot maximum (avec flux de données entiers) est de type 1, mais pas de type 2.

Le problème du flot maximum (avec flux de données entiers ou non) est de type 2, et donc de type 1 également.

Difficultés liées à la résolution de problèmes (4/9)

Parmi les principaux problèmes d'optimisation dans les réseaux que l'on étudiera, on en compte essentiellement deux types :

1. ceux tels qu'on peut, de façon efficace, vérifier qu'il existe une solution de valeur plus grande/petite qu'une valeur donnée (une telle solution étant fournie, si nécessaire),
2. ceux tels qu'on peut, de façon efficace, décider s'il existe une solution de valeur plus grande/petite qu'une valeur donnée.

Tout problème de type 2 (donc "facile") est également de type 1.

Le problème du biflot maximum (avec flux de données entiers) est de type 1, mais pas de type 2.

Le problème du flot maximum (avec flux de données entiers ou non) est de type 2, et donc de type 1 également.

Difficultés liées à la résolution de problèmes (4/9)

Parmi les principaux problèmes d'optimisation dans les réseaux que l'on étudiera, on en compte essentiellement deux types :

1. ceux tels qu'on peut, de façon efficace, vérifier qu'il existe une solution de valeur plus grande/petite qu'une valeur donnée (une telle solution étant fournie, si nécessaire),
2. ceux tels qu'on peut, de façon efficace, décider s'il existe une solution de valeur plus grande/petite qu'une valeur donnée.

Tout problème de type 2 (donc "facile") est également de type 1.

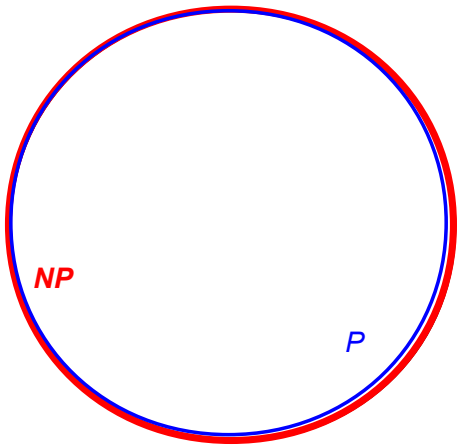
Le problème du biflot maximum (avec flux de données entiers) est de type 1, mais pas de type 2.

Le problème du flot maximum (avec flux de données entiers ou non) est de type 2, et donc de type 1 également.

Difficultés liées à la résolution de problèmes (5/9)

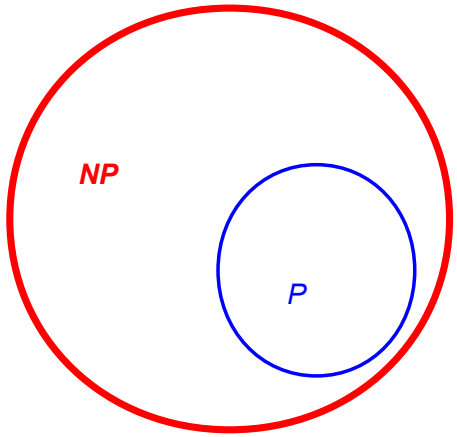
Les problèmes de type 2 (donc “faciles”), qu’ils soient des problèmes d’optimisation dans les réseaux ou non, forment un ensemble noté P (pour “polynomial”).

Ceux de type 1 forment un ensemble noté NP, qui contient P.



$P = NP$?

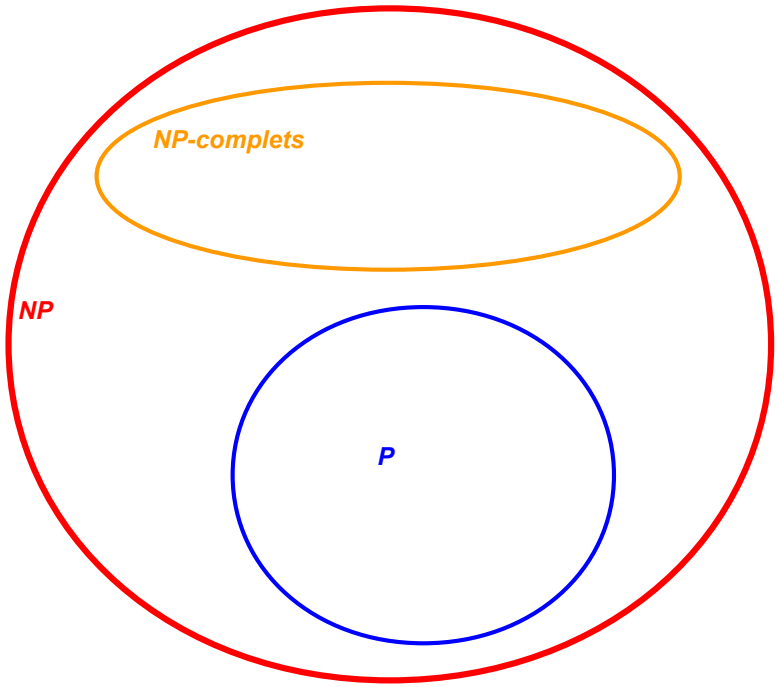
Ou, plus probablement...



$P \neq NP$?

Difficultés liées à la résolution de problèmes (6/9)

NP contient des problèmes dits *NP*-**complets**, tous équivalents : s'il existe un algorithme efficace pour résoudre l'un d'eux, alors il en existe un pour chacun d'entre eux, et pour tous ceux de *NP*.



Difficultés liées à la résolution de problèmes (7/9)

Un exemple de problème qui est dans *NP* **et** *NP*-complet : le SUDOKU (où toutes les solutions ont la même valeur).

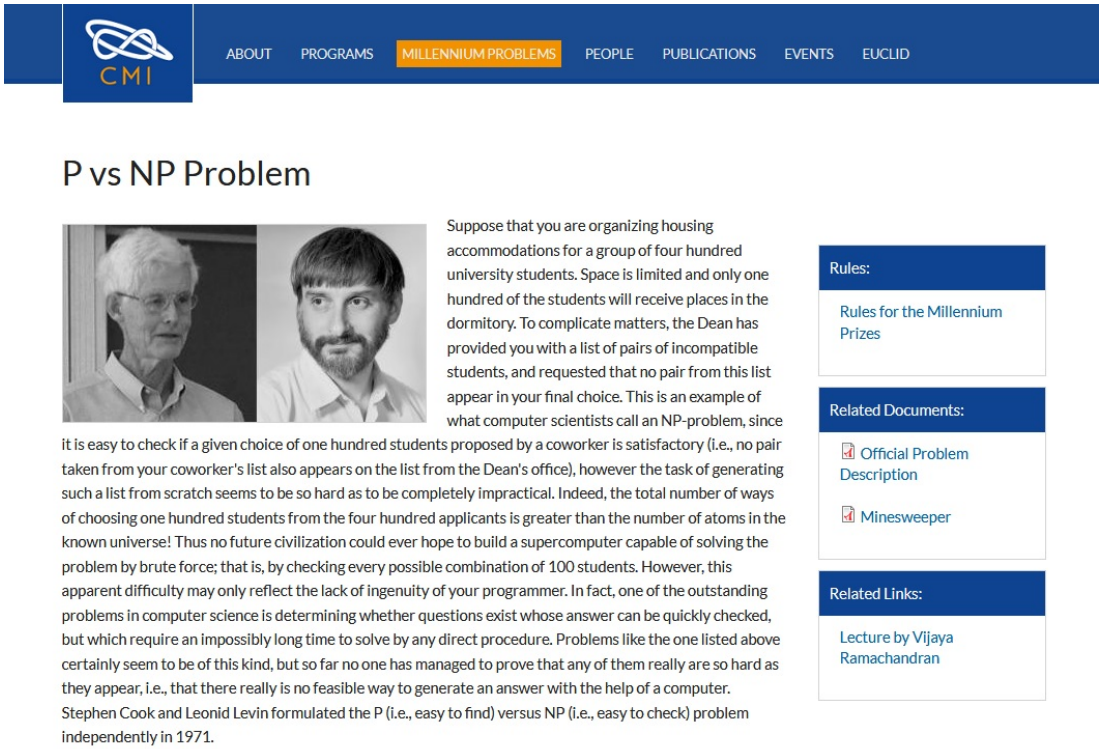
			8					9
	1	9			5	8	3	
	4	3		1				7
4			1	5				3
		2	7		4		1	
	8			9		6		
	7				6	3		
	3			7			8	
9		4	5					1

2	5	6	8	3	7	1	4	9
7	1	9	4	2	5	8	3	6
8	4	3	6	1	9	2	5	7
4	6	7	1	5	8	9	2	3
3	9	2	7	6	4	5	1	8
5	8	1	3	9	2	6	7	4
1	7	8	2	4	6	3	9	5
6	3	5	9	7	1	4	8	2
9	2	4	5	8	3	7	6	1

Difficultés liées à la résolution de problèmes (8/9)

Personne ne sait s'il existe un algorithme efficace pour résoudre le SUDOKU ou le biflot max., ou, en d'autres termes, si $P = NP$.

Plus de détails, et 1 million de dollars à la clé, sur le site du CMI (http://www.claymath.org/millennium-problems/p-vs-np-problem)



Difficultés liées à la résolution de problèmes (9/9)

Et si on est malgré tout contraint de résoudre un problème d'optimisation dans les réseaux qui est NP-complet ? On peut :

- modéliser le problème sous une forme plus générale que la programmation linéaire (PLNE), puis utiliser des méthodes de résolution peu efficaces en théorie, mais efficaces en pratique sur des données pas trop volumineuses, et implémentées dans des solveurs (cf GLPK pour le biflot),
- ou utiliser des méthodes de résolution adaptées au problème, pas toujours efficaces en théorie, mais parfois plus efficaces que les précédentes en pratique (programmation dynamique),
- ou calculer à l'aide d'un algorithme efficace une solution approchée (heuristique/méta-heuristique ; cf encore le biflot).

Difficultés liées à la résolution de problèmes (9/9)

Et si on est malgré tout contraint de résoudre un problème d'optimisation dans les réseaux qui est NP-complet ? On peut :

- modéliser le problème sous une forme plus générale que la programmation linéaire (PLNE), puis utiliser des méthodes de résolution peu efficaces en théorie, mais efficaces en pratique sur des données pas trop volumineuses, et implémentées dans des solveurs (cf GLPK pour le biflot),
- ou utiliser des méthodes de résolution adaptées au problème, pas toujours efficaces en théorie, mais parfois plus efficaces que les précédentes en pratique (programmation dynamique),
- ou calculer à l'aide d'un algorithme efficace une solution approchée (heuristique/méta-heuristique ; cf encore le biflot).

Difficultés liées à la résolution de problèmes (9/9)

Et si on est malgré tout contraint de résoudre un problème d'optimisation dans les réseaux qui est NP-complet ? On peut :

- modéliser le problème sous une forme plus générale que la programmation linéaire (PLNE), puis utiliser des méthodes de résolution peu efficaces en théorie, mais efficaces en pratique sur des données pas trop volumineuses, et implémentées dans des solveurs (cf GLPK pour le biflot),
- ou utiliser des méthodes de résolution adaptées au problème, pas toujours efficaces en théorie, mais parfois plus efficaces que les précédentes en pratique (programmation dynamique),
- ou calculer à l'aide d'un algorithme efficace une solution approchée (heuristique/méta-heuristique ; cf encore le biflot).

Difficultés liées à la résolution de problèmes (9/9)

Et si on est malgré tout contraint de résoudre un problème d'optimisation dans les réseaux qui est NP-complet ? On peut :

- modéliser le problème sous une forme plus générale que la programmation linéaire (PLNE), puis utiliser des méthodes de résolution peu efficaces en théorie, mais efficaces en pratique sur des données pas trop volumineuses, et implémentées dans des solveurs (cf GLPK pour le biflot),
- ou utiliser des méthodes de résolution adaptées au problème, pas toujours efficaces en théorie, mais parfois plus efficaces que les précédentes en pratique (programmation dynamique),
- ou calculer à l'aide d'un algorithme efficace une solution approchée (heuristique/méta-heuristique ; cf encore le biflot).

Pour la suite...

- Quatre prochaines séances : autres problèmes “faciles” d’optimisation dans les réseaux (tables de routage, arbres de connexion, flots maximums/avec demandes/à coût minimum via la programmation linéaire), et retour sur un problème difficile pour illustrer P vs NP,
- Modélisation et résolution par la PLNE (via GLPK),
- Résolution par la programmation dynamique,
- Résolution par des (méta-)heuristiques.

Illustration systématique de ces méthodes sur des cas d’étude.

Pour la suite...

- Quatre prochaines séances : autres problèmes “faciles” d’optimisation dans les réseaux (tables de routage, arbres de connexion, flots maximums/avec demandes/à coût minimum via la programmation linéaire), et retour sur un problème difficile pour illustrer P vs NP,
- Modélisation et résolution par la PLNE (via GLPK),
- Résolution par la programmation dynamique,
- Résolution par des (méta-)heuristiques.

Illustration systématique de ces méthodes sur des cas d’étude.

Pour la suite...

- Quatre prochaines séances : autres problèmes “faciles” d’optimisation dans les réseaux (tables de routage, arbres de connexion, flots maximums/avec demandes/à coût minimum via la programmation linéaire), et retour sur un problème difficile pour illustrer P vs NP,
- Modélisation et résolution par la PLNE (via GLPK),
- Résolution par la programmation dynamique,
- Résolution par des (méta-)heuristiques.

Illustration systématique de ces méthodes sur des cas d’étude.

Pour la suite...

- Quatre prochaines séances : autres problèmes “faciles” d’optimisation dans les réseaux (tables de routage, arbres de connexion, flots maximums/avec demandes/à coût minimum via la programmation linéaire), et retour sur un problème difficile pour illustrer P vs NP,
- Modélisation et résolution par la PLNE (via GLPK),
- Résolution par la programmation dynamique,
- Résolution par des (méta-)heuristiques.

Illustration systématique de ces méthodes sur des cas d’étude.

Pour la suite...

- Quatre prochaines séances : autres problèmes “faciles” d’optimisation dans les réseaux (tables de routage, arbres de connexion, flots maximums/avec demandes/à coût minimum via la programmation linéaire), et retour sur un problème difficile pour illustrer P vs NP,
- Modélisation et résolution par la PLNE (via GLPK),
- Résolution par la programmation dynamique,
- Résolution par des (méta-)heuristiques.

Illustration systématique de ces méthodes sur des cas d’étude.