Aide à la Décision

c'est Graphe Docteur !!! - version provisoire

Patrice Bonhomme

November 4, 2013

Outline

Outline

Aide à la décision

Quelques définitions

Wikipédia dit quoi ?

Dans la vie courante ou dans les domaines complexes et pleins d'incertitudes de la gouvernance, de la sûreté, de la sécurité, de la santé, des transports, de la logistique, de l'aménagement du territoire ou l'environnement ou d'autres..., les élus, décideurs,médecins, chirurgiens ont besoin d'indicateurs et d'outils d'aide au diagnostic et à la décision pour effectuer, valider, justifier, évaluer ou corriger les décisions importantes qu'ils doivent prendre. Ceci se fait en fonction de critères de plus en plus complexes interdépendants et mondialisés, tels que par exemple la consommation d'énergie, de foncier, de ressources naturelles (ressources pas, peu, difficilement, couteusement, ou lentement renouvelables), ou encore selon les ressources humaines, financières ou en temps disponibles, ou selon divers critères d'acceptabilité, socioéconomiques ou plus socio-politiques.

Aide à la décision
Mais encore !!

Toujours Wikipédia !

- Les progrès de l'informatique ont intégré l'aide à la décision, domaine visant à concevoir des outils informatiques (dont logiciel-expert) pour aider un décideur à analyser un problème ou une situation, et à lui fournir des solutions, éventuellement hiérarchisées sur la base des critères logiques qu'il aura sélectionné. Ainsi l'informatique décisionnelle, qui fait partie de ce que les anglophones appellent "' business intelligence "' désigne les moyens, les outils et les méthodes qui permettent de collecter, consolider, modéliser et restituer les données, matérielles ou immatérielles, d'une entreprise en vue d'offrir une aide à la décision et de permettre aux responsables de la stratégie d'entreprise ou d'une collectivité d'avoir une vue d'ensemble de l'activité traitée.

Aide à la décision
Mais encore !!

Toujours Wikipédia ! (suite)

- **La recherche opérationnelle** travaille dans ce domaine à la production de modèles réalistes du problème d'un décideur dans son contexte, puis à la résolution des problèmes, et hiérarchisation des solutions possibles.

Aide à la décision

Mais encore !!

vu sur le site du Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la DEcision

Le pôle "'Aide à la Décision "' se caractérise par une vision transversale de sa thématique, en mettant l'accent d'une part sur le développement de modèles et d'algorithmes qui visent à apporter des éléments de réponse à de nombreux problèmes de décision .... Nos activités de recherche se situent au carrefour de plusieurs disciplines (notamment **la recherche opérationnelle**, les mathématiques discrètes et l'intelligence artificielle).

Aide à la décision

La ROADEF - Société française de Recherche Opérationnelle et Aide à la Décision

Recherche Opérationnelle et Aide à la Décision

Le Recherche Opérationnelle (RO) est la discipline des méthodes scientifiques utilisables pour élaborer de meilleures décisions. Elle permet de rationaliser, de simuler et d'optimiser l'architecture et le fonctionnement des systèmes de production ou d'organisation. La RO propose des modèles conceptuels pour analyser des situations complexes et permet aux décideurs de faire les choix les plus efficaces grâce à:

- une meilleure compréhension des problèmes,
- une vision complète des données,
- la considération de toutes les solutions possibles,
- des prédictions prudentes de résultats incluant une évaluation des risques,
- des outils et des méthodes modernes d'aide à la décision.

Aide à la décision

La ROADEF - Société française de Recherche Opérationnelle et Aide à la Décision

Recherche Opérationnelle et Aide à la Décision

La RO apparaît comme une discipline carrefour associant les mathématiques, l'économie et l'informatique. Elle est par nature en prise directe sur l'industrie et joue un rôle-clé dans le maintien de la compétitivité.
Les apports de la RO sont visibles tout autour de nous et dans les domaines les plus divers: de l'organisation des lignes de production de véhicules à la planification des missions spatiales, de l'optimisation des portefeuilles bancaires à l'aide au séquençage de l'ADN, voire dans la "'vie de tous les jours"' dans l'organisation des produits recyclables, l'organisation des ramassages scolaires ou la couverture satellite des téléphones portables ...

http://www.roadef.org

ROADEF

Société Française de Recherche Opérationnelle et d'Aide à la Décision

Qu'est-ce que la R.O. ?

La Recherche Opérationnelle (R.O.) est la discipline des méthodes scientifiques utilisables pour élaborer de meilleures décisions. Elle permet de rationaliser, de simuler et d'optimiser l'architecture et le fonctionnement des systèmes de production ou d'organisation.

La R.O. propose des modèles conceptuels pour analyser des situations complexes et permet aux décideurs de faire les choix les plus efficaces grâce à :

- une meilleure compréhension des problèmes,
- une vision complète des données,
- la considération de toutes les solutions possibles,
- des prédictions prudentes de résultats incluant une évaluation des risques,
- des outils et des méthodes modernes d'aide à la décision.

La R.O. apparaît comme une discipline carrefour associant les mathématiques, l'économie et l'informatique.
Elle est par nature en prise directe sur l'industrie et joue un rôle-clé dans le maintien de la compétitivité.

Qu'est-ce que la R.O. ?

Les apports de la R.O. sont visibles tout autour de nous et dans les domaines les plus divers : de l'organisation des lignes de production de véhicules à la planification des missions spatiales, de l'optimisation de portefeuilles bancaires à l'aide au séquençage de l'ADN, voire dans la " vie de tous les jours " dans l'organisation du traitement des produits recyclables, l'organisation des ramassages scolaires ou la couverture satellite des téléphones portables…

" Life itself is a matter of O.R. " comme le clame le slogan de EURO, l'association des sociétés européennes de R.O.

" La vie elle-même est affaire de R.O. " !

D'où vient la R.O. ?

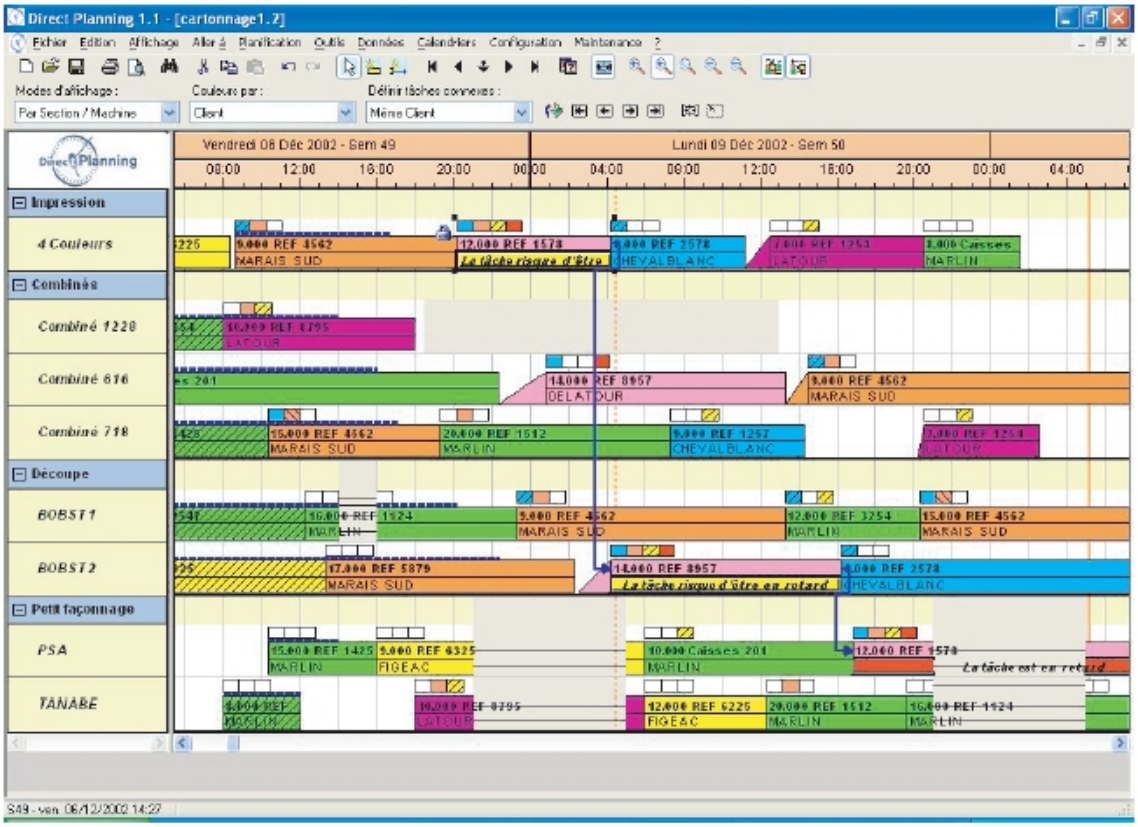
Malgré l'apparente jeunesse de la R.O., la démarche est ancienne : la légende ne dit-elle pas que Carthage fut construite en arc de cercle grâce à la reine Didon qui sut maximiser la surface contenue dans un périmètre qui lui était imposé ? Et Aristote, dès le IVème siècle avant notre ère, ne traduisait-il pas en " désirs rationnels " les " préférences " exprimées par l'homme ? Pour peu qu'il ait un esprit scientifique, tout homme qui cherche à obtenir des résultats optimaux dans des conditions données fait de la R.O. sans le savoir !

A la fin du XVIIIème siècle, des mathématiciens célèbres comme Monge ou Fourier se sont intéressés à ces questions. Toutefois, ce n'est qu'au cours de la seconde guerre mondiale que le terme " Recherche Opérationnelle " apparaît.
L'armée britannique confie à Blackett le " *Committee for operations research* " pour apporter un soutien scientifique à ses choix stratégiques : formation de convois dans l'Atlantique Nord, localisation de radars de surveillance, optimisation des gerbes de tirs anti-sous-marins…
Voilà qui explique la consonance militaire associée à une discipline qui s'est développée depuis les années 1940 et qui a aujourd'hui un champ d'applications beaucoup plus vaste.

La R.O. s'appuie sur la Théorie des Graphes (König, 1936 puis Berge, 1958), la Programmation Linéaire (Dantzig, 1949), la Programmation Dynamique (Bellmann, 1954) et la Théorie des Jeux (Von Neumann et Morgestern, 1944).

D'où vient la R.O. ?

Outre les noms prestigieux des prix Nobel Allais, Arrow, Kantorovich, Koopmans, Nash, Simon, il est impossible de citer tous les noms de ceux qui ont fait de la R.O. la discipline de l'Aide à la Décision ou qui ont joué un rôle dans son développement. En France, la R.O. doit beaucoup à Abadie, Faure, Kaufmann, Lesourne, Roy et Sakarovich soit pour leurs travaux scientifiques de renommée mondiale, soit pour leur application pionnière de la discipline dans le monde industriel, soit enfin pour leur diffusion précoce de la discipline dans les grandes écoles et les universités.



(Photo Volume Software www.directplanning.com)

Mais l'essor véritable de la R.O. est dû à celui de l'informatique qui lui a apporté les moyens de calculs nécessaires à la résolution des problèmes réels, et ce n'est pas un effet de mode ! En plus de soixante ans de tradition scientifique, la R.O. a permis de mener à terme efficacement de nombreux projets économiques, technologiques ou sociétaux. Discipline jeune et en pleine expansion, elle est devenue indispensable aux professionnels pour mieux gérer notre monde complexe.

Les domaines d'application de la R.O.

La R.O. est une discipline fortement implantée dans le tissu universitaire et professionnel des grands pays industrialisés. En France, elle est très présente dans tous les grands secteurs tels que les transports, l'énergie, la production et les télécommunications. Que ce soit dans un environnement stable, concurrentiel ou incertain, elle a été appliquée avec succès, entre autres, en :

- Optimisation dans les télécommunications
- Gestion de stocks et de la Production
- Planification et Ordonnancement de tâches
- Affectation et Localisation de ressources
- Gestion des horaires du personnel et élaboration d'emplois du temps
- Optimisation de la politique énergétique
- Gestion de l'eau et des ressources naturelles
- Evaluation des politiques publiques
- Aide à la décision pour la banque et la finance
- Optimisation des transports et de la chaîne logistique
- Gestion du trafic aérien ou ferroviaire
- Planification pour l'industrie agroalimentaire et l'agriculture
- Intelligence artificielle et robotique
- Calcul parallèle et Informatique répartie
- Classification de données (marketing, internet, assurances)

De plus, l'évolution de nos sociétés incite la R.O. à faire face à de nouveaux défis qui se présenteront dans un avenir proche, par exemple en :

- Ecologie
- Biologie moléculaire, génétique
- Extraction des connaissances dans les grandes bases de données
- Réseaux informatiques
- Calculs informatiques distribués (grilles)
- Domaine spatial

Les domaines d'application de la R.O.

Quel que soit son domaine d'application, la R.O. aide à l'émergence d'idées nouvelles, à la mise en évidence des choix et à l'identification puis à l'implémentation des solutions.

hoto CAV-SNCF)

L'optimisation et la rationalisation de la production et de l'organisation apparaissent comme des moyens indispensables au maintien de la compétitivité.

La R.O. poursuit son développement aussi bien dans les sociétés modernes que dans les pays émergents.

Les sociétés de R.O. dans le monde et en Europe

Dans le monde entier les spécialistes en R.O. se sont rassemblés en sociétés.
Les sociétés internationales dont la ROADEF est membre sont :

· IFORS *International Federation of Operational Research Societies*
qui regroupe 48 sociétés adhérentes et plus de 35 000 membres et publie la revue internationale ITOR. La France a été le troisième pays fondateur d'IFORS dès 1959.

· EURO *Association des sociétés européennes de Recherche Opérationnelle*
créée en 1975 au sein d' IFORS, qui regroupe 29 sociétés et 10 000 membres et qui publie une revue internationale (EJOR).

La société EURO a ses homologues en :

Afrique : AORN (African Operational Research Network),
Asie et Pacifique sud : APORS (Association of Asian-Pacific Operational Research Societies),
Amérique du sud : ALIO (Association of Latin-Iberoamerican Operational Research Societies)
Amérique du nord : NORAM (The Association of North American Operations Research Societies).

Ces sociétés fédèrent de nombreuses sociétés nationales dont certaines ont un nombre important de membres, comme INFORMS (10 000 membres) et ORS (3 000 membres).

La ROADEF entretient des relations étroites avec les principales sociétés nationales européennes dont plus particulièrement AIRO (Italian Society of Operations Research, Optimization and Decision Science), et SOGESCI-BVWB (Belgique).
Le Royaume Uni, l'Allemagne, l'Espagne, le Portugal, la Grèce, la Turquie et les Pays-Bas ont également des sociétés très actives.

http://www.roadef.org

Qu'est-ce que la ROADEF ?

· La "Société Française de Recherche Opérationnelle et d'Aide à la Décision" est une association loi 1901, membre de IFORS et de EURO.

· La ROADEF est la société de tous ceux qui élaborent ou utilisent des outils de recherche opérationnelle ou d'aide à la décision.

· Créée le 22 janvier 1998 à Paris, l'association a pris la suite de la SOFRO (1956), de l'AFIRO (1964) et enfin de l'AFCET (1968).

· La ROADEF regroupe environ 300 membres individuels (industriels, organisateurs, chercheurs, enseignants-chercheurs et doctorants) ou institutionnels (laboratoires et services universitaires ou industriels).

· La ROADEF publie une revue internationale 4'OR qui est co-éditée avec les sociétés sœurs belge (SOGESCI-BVWB) et italienne (AIRO) et qui paraît quatre fois par an.
Elle parraine avec la SMAI (Société de Mathématiques Appliquées et Industrielles) la revue RAIRO-RO.

· La ROADEF apporte un soutien scientifique aux groupes de recherche thématiques ou géographiques actifs en R.O.

· La ROADEF parraine les congrès et manifestations scientifiques traitant de la R.O.

· Le prix Robert Faure est décerné à de jeunes chercheurs tous les trois ans pour des travaux scientifiques, théoriques et appliqués de haut niveau, par un jury de scientifiques confirmés constitué par la ROADEF.

· Un " challenge " organisé tous les deux ans sur un sujet proposé par un industriel récompense les équipes ayant fourni les meilleurs résultats.

· Le congrès national annuel rassemble toute la communauté scientifique et industrielle, et favorise la publication d'articles de jeunes chercheurs.

La ROADEF, grâce à son site web, sa lettre électronique et son bulletin semestriel diffuse l'information, fait connaître la discipline
et rassemble les spécialistes du domaine.

Pourquoi devenir membre ?

La ROADEF existe grâce à ses membres, individuels ou institutionnels. Devenir membre est le moyen le plus important pour soutenir la ROADEF et lui permettre de jouer son rôle, et ainsi de participer individuellement et concrètement à la promotion et au développement de la recherche opérationnelle.

Tout membre bénéficie des avantages offerts par l'association :

· Réception des numéros de 4'OR, la revue scientifique de l'association.
· Réception du bulletin biannuel.
· Réception de la lettre électronique d'informations bimestrielle.
· Possibilité de participer aux groupes de travail parrainés par l'association et à toute manifestation organisée par la ROADEF.
· Droit au tarif réduit lors de la conférence annuelle.
· Droit de vote lors de l'Assemblée Générale annuelle, des AG extraordinaires et de l'élection du bureau.

Objectifs de la ROADEF

· Promouvoir et encourager l'emploi de la R.O. dans tous les secteurs de la société où elle peut intervenir.

· Aider à la reconnaissance du rôle et de la qualification professionnelle des chercheurs opérationnels.

· Faciliter les relations entre les associations, écoles ou universités et les organisations ou entreprises.

· Promouvoir les publications scientifiques et "grand public" dans la discipline.

· Faire connaître et aider à développer les enseignements relatifs à la discipline.

· Diffuser l'information et annoncer les conférences, réunions de groupes de travail et séminaires.

ROADEF
Société Française de Recherche Opérationnelle et d'Aide à la Décision
Université François-Rabelais de Tours
Laboratoire d'Informatique
64 avenue Jean Portalis, 37200 Tours

fax : +33 247 361 422
bureau@roadef.org
www.roadef.org

Aide à la décision

Quest ce que la recherche opérationnelle ?

Recherche Opérationnelle et Aide à la Décision

- Recherche opérationnelle = programmation mathématique = optimisation (mais pas optimisation de programme)
- Recherche opérationnelle = modélisation mathématique des processus de prise de décision.
  - Inconnues : les variables de décision.
  - Evaluation de la décision = fonction économique ou fonction objectif.
  - Trouver les valeurs des variables de décision qui minimisent (ou maximisent) la fonction objectif.

Outline

Introduction à la Complexité

Notion de complexité

complexité

- Comment évaluer les performances d'un algorithme
- différents algorithmes ont des couts différents en termes de
  - temps d'exécution (nombre d'opérations effectuées par l'algorithme),
  - taille mémoire (taille nécessaire pour stocker les différentes structures de données pour l'exécution).

Ces deux concepts sont appelés la complexité en temps et en espace de l'algorithme.

Introduction à la Complexité

Notion de complexité

complexité

- La complexité algorithmique permet de mesurer les performances d'un algorithme et de le comparer avec d'autres algortihmes réalisant les même fonctionnalités.
- La complexité algorithmique est un concept fondamental pour tout informaticien, elle permet de déterminer si un algorithme *a* est meilleur qu'un algortihme *b* et s'il est optimal ou s'il ne doit pas être utilisé. . .

Temps d'exécution

Le temps d'exécution d'un programme dépend :

1. du nombre de données,
2. de la taille du code,
3. du type d'ordinateur utilisé (processeur,mémoire),
4. de la complexité en temps de l'algorithme 'abstrait' sous-jacent.

Temps d'exécution

Soit n la taille des données du problème et $T(n)$ le temps d'exécution de l'algorithme.
On distingue :

- Le temps du plus mauvais cas $T_{max}(n)$ qui correspond au temps maximum pris par l'algorithme pour un problème de taille n
- le temps moyen T_{moy} le temps moyen d'exécution sur des données de taille n

Temps d'exécution

Règles générales

1. le temps d'exécution ($t.e.$) d'une affectation ou d'un test est considéré comme constant c,
2. Le temps d'une séquence d'instructions est la somme des ($t.e.$) des instructions qui la composent,
3. le temps d'un branchement conditionnel est égal au $t.e.$ du test plus le max des deux $t.e.$ correspondant aux deux alternatives (dans le cas d'un temps max).
4. Le temps d'une boucle est égal à la somme du coût du test + du corps de la boucle + test de sortie de boucle.

Temps d'exécution

Soit l'algorithme suivant :
Nom: Calcul d'une somme de carrés
Role: Calculer la valeur moyenne d'un tableau
Entrée: n : entier
Sortie: *somme* : réel
Déclaration: i: Naturel
début
$somme \leftarrow 0.0$
pour $i \leftarrow 0$ à $n-1$ **faire**
$somme \leftarrow somme + i * i$
fin pour
fin

Temps d'exécution

$$T_{moy}(n) = T_{max}(n) = c_1 + c_1 + n(c_2 + c_3 + c_4 + c_5 + c_1)$$

$$T_{moy}(n) = T_{max}(n) = 2c_1 + n(\sum_{i=1}^{5} c_i)$$

avec c_1 : affectation, c_2 : incrémentation, c_3 : test, c_4 : addition, c_5 : multiplication.

$$\text{Trop précis} \Longrightarrow \text{Inutilisable !!!!}$$

Notion de complexité : comportement asymptotique du $t.e.$:

$$T_{max}(n) = T_{moy}(n) = nC$$

l'algorithme a une complexité en $\mathcal{O}(n)$

Ordre de grandeur asymptotique : la notation de Landau $\mathcal{O}$

la notation $\mathcal{O}$

Soient f et g deux fonctions de $\mathcal{N}$ dans $\mathcal{R}^{*+}$
$f = \mathcal{O}(g)$ (f est en grand $\mathcal{O}$ de g) ssi :

$$\exists c \in \mathcal{R}^{*+}, \exists n_0 \text{ tel que}$$

$$\forall n > n_0, f(n) \leq cg(n)$$

- Exemple: $f(n) = 10n$ et $g(n) = \mathcal{O}(n^2)$
- **Attention** : il ne s'agit que d'une **borne supérieure**, et à partir d'un certain rang, cela indique juste que f ne croit pas plus vite que g à partir de ce rang (mais rien n'indique qu'elle croit moins vite, ni qu'elle croit aussi vite)

Propriétés de la notation $\mathcal{O}$

Propriétés

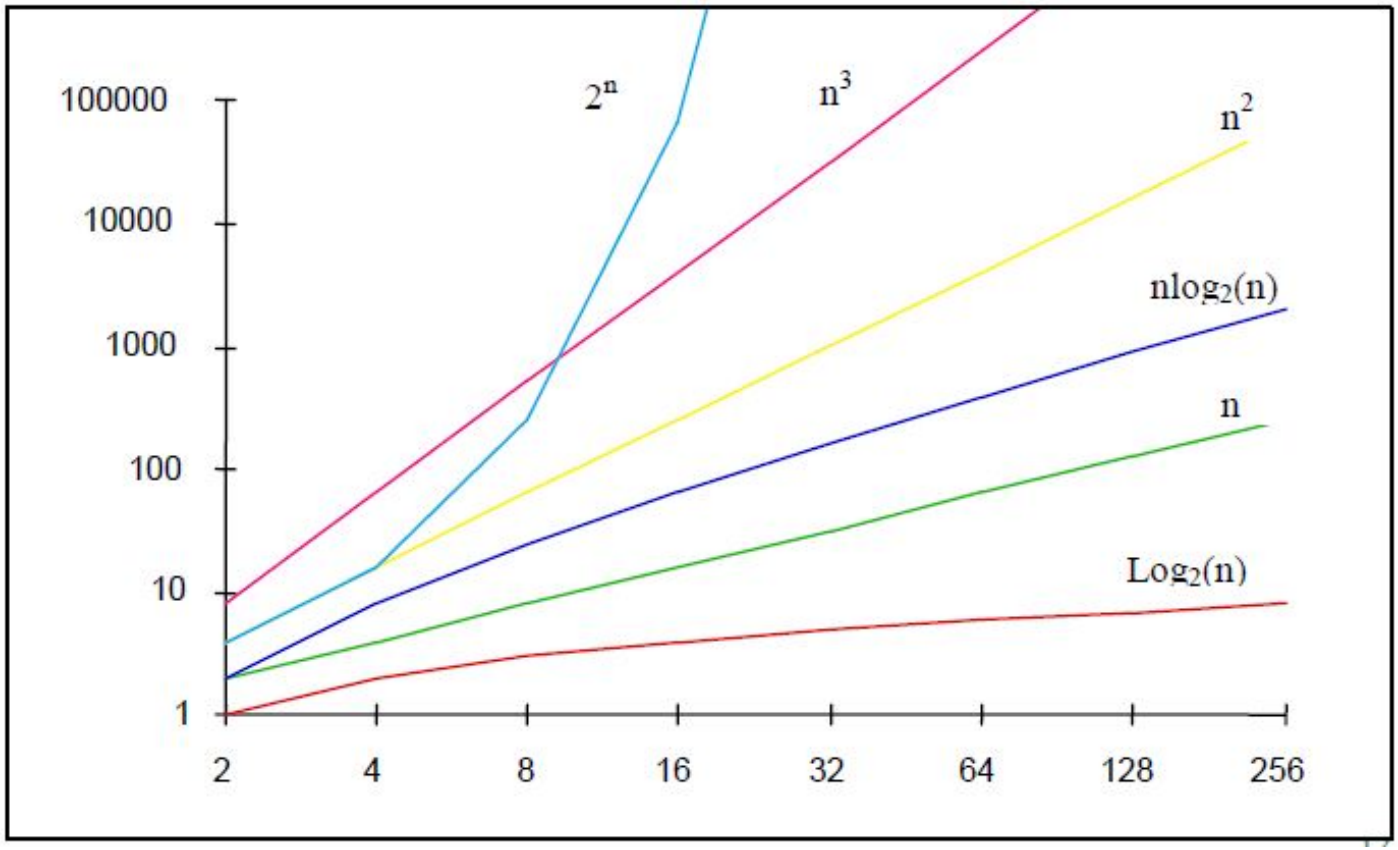
- Les constantes ne sont pas importantes
- Les termes d'ordre inférieur sont négligeables
  - Si $T(n) = a_k n^k + \ldots + a_1 n + a_0$ avec $a_k > 0$ alors $T(n)$ est en n^k.
  - Si $\frac{h(n)}{g(n)} \to 0$ quand $n \to +\infty$ alors $g(n) + h(n)$ est en $\mathcal{O}(g)$
- Si $T1(n) = \mathcal{O}(f1)$ et $T2(n) = \mathcal{O}(f2)$ alors $T1(n) + T2(n) = \mathcal{O}(f1 + f2)$ et $T1(n) \times T2(n) = \mathcal{O}(f1 \times f2)$
- Si $f(n) = \mathcal{O}(g)$ et $g(n) = \mathcal{O}(h)$ alors $f(n) = \mathcal{O}(h)$

Complexité

Principales complexités

$\mathcal{O}(1)$: temps constant,
$\mathcal{O}(log(n))$: complexité logarithmique,
$\mathcal{O}(n)$: complexité linéaire,
$\mathcal{O}(nlog(n))$,
$\mathcal{O}(n^2), \mathcal{O}(n^3), \mathcal{O}(n^p)$: complexités polynomiales
$\mathcal{O}(p^n)$:complexité exponentielle.

Croissance comparée des fonctions fréquemment utilisées en complexité



Influence de la complexité

Le tableau suivant illustre quelques croissances par rapport à des croissances modérées, c'est à dire polynomiales (les quatre premières). Les temps de calculs suposent une microseconde par instruction de haut niveau. Les cases non remplies ont des durées supérieures à 1000 milliards d'années (c'est à dire très supérieures à l'âge estimé de l'univers !!).

Complexité/Taille	20	50	100	200	500	1000
$10^3 n$	0.02s	0.05s	0.1s	0.2s	0.5s	1s
$10^3 n log_2 n$	0.09s	0.3s	0.6s	1.5s	4.5s	10s
$100n^2$	0.04s	0.25s	1s	4s	25s	2min
$10n^3$	0.02s	1s	10s	1min	21 min	27 h
$n^{log_2 n}$	0.4s	1.1 h	220 jours	12500 ans	5.10^{10} ans	-
$2^{n/3}$	0.0001s	0.1 s	2.7 h	$3.10^6 ans$	-	-
2^n	1s	36 ans	-	-	-	-
3^n	58 min	$2.10^{11} ans$	-	-	-	-
$n!$	77100 ans	-	-	-	-	-

Influence de la complexité

$\mathcal{O}(1)$	$\mathcal{O}(\log_2(n))$	$\mathcal{O}(n)$	$\mathcal{O}(n\log_2(n))$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(2^n)$
t	$t+1$	$2t$	$2t+2n$	$4t$	$8t$	t^2

	1	$\log_2(n)$	n	$n\log_2(n)$	n^2	n^3	2^n
$n=10^2$	$1\mu s$	$6\mu s$	$0.1ms$	$0.6ms$	$10ms$	$1s$	$4\times10^{16}a$
$n=10^3$	$1\mu s$	$10\mu s$	$1ms$	$10ms$	$1s$	$16.6min$	∞
$n=10^4$	$1\mu s$	$13\mu s$	$10ms$	$0.1s$	$100s$	$11,5j$	∞
$n=10^5$	$1\mu s$	$17\mu s$	$0.1s$	$1.6s$	$2.7h$	$32a$	∞
$n=10^6$	$1\mu s$	$20\mu s$	$1s$	$19.9s$	$11,5j$	32×10^3a	∞

Autres notation

Il existe d'autres notations pour decrire le comportement asymptotique des fonctions :

Autres notations

- f domine asymptotiquement g : $f(n) = \Omega(g(n))$ si et seulement s'il existe des constantes $c > 0$ et n_0 telles que $f(n) \geq c * g(n)$ pour tout $n \geq n_0$
- f est asymptotiquement equivalente a g : $f(n) = \Theta(g(n))$ si et seulement s'il existe des constantes c_1, c_2 strictement positives et n_0 telles que $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ pour tout $n \geq n_0$

Outline

Les graphes

Dans la vie courante !!!

Utilité

Les graphes modélisent de nombreuses situations concrêtes où interviennent des objets en interaction

- Les interconnexions routière, ferrovière ou aériennes entre différentes agglomérations,
- Les liens entre les composants d'un circuit électronique,
- Le plan d'une ville et de ses rues en sens unique,...

Mais encore !!

Les graphes permettent de manipuler plus facilement des objets et leurs relations avec une représentation graphique naturelle. L'ensemble des techniques et outils mathématiques mis au point en Théorie des Graphes permettent de démontrer facilement des propriétés, d'en déduire des méthodes de résolution, des algorithmes, ...

- Quel est le plus court chemin (en distance ou en temps) pour se rendre d'une ville à une autre?
- Comment minimiser la longueur totale des connexions d'un circuit?
- Peut-on mettre une rue en sens unique sans rendre impossible la circulation en ville?

Définition

Un graphe permet de décrire un ensemble d'objets et leurs relations, c'est à dire les liens entre les objets.
Les objets sont appelés les noeuds, ou encore les sommets du graphe.
Un lien entre deux objets est appelé une arête.

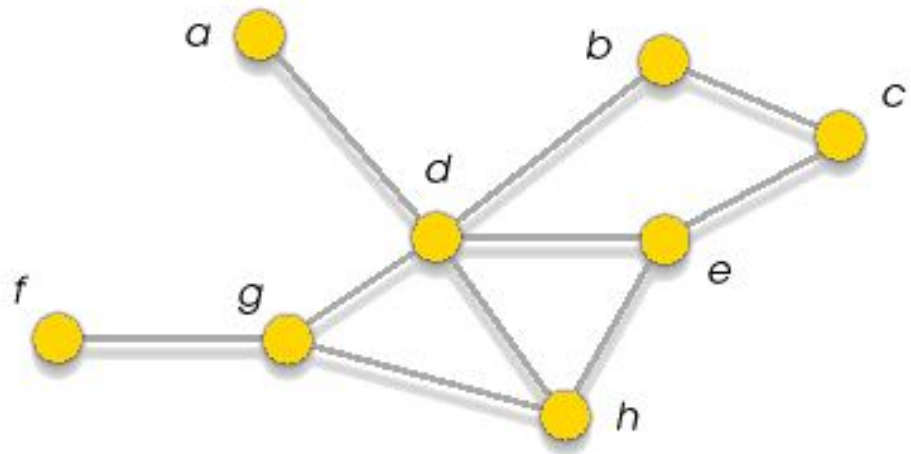
Définition

Un graphe G est un couple (V, E) où V est un ensemble (fini) d'objets.
Les éléments de V sont appelés les sommets du graphe.
E est sous-ensemble de $V \times V$. Les éléments de E sont appelés les arêtes du graphe.
Une arête e du graphe est une paire $e = (x, y)$ de sommets. Les sommets x et y sont les extrémités de l'arête.

Exemple

Un exemple de graphe à 8 sommets, nommés a à h, comportant 10 arêtes.

- $G = (V, E)$
- $V = a, b, c, d, e, f, g, h$
- $E = (a, d), (b, c), (b, d), (d, e), (e, c), (e, h), ...$



Notre définition d'un graphe correspond au cas des graphes simples, pour lesquels il existe au plus une arête liant deux sommets. Dans le cas contraire le graphe est dit multiple.

Définitions

Les objets représentés par les sommets sont sans importance pour la manipulation du graphe. Nous dirons simplement qu'un graphe est d'ordre n si il comporte n sommets. Toute la richesse des graphes vient évidemment de la grande diversité que peut avoir l'ensemble de ses arêtes. Pour appréhender la structure d'un graphe, nous pouvons commencer par la caractériser localement en regardant pour chaque sommet les autres sommets auquels il est relié, le nombre d'arêtes dont il est extrémité,...

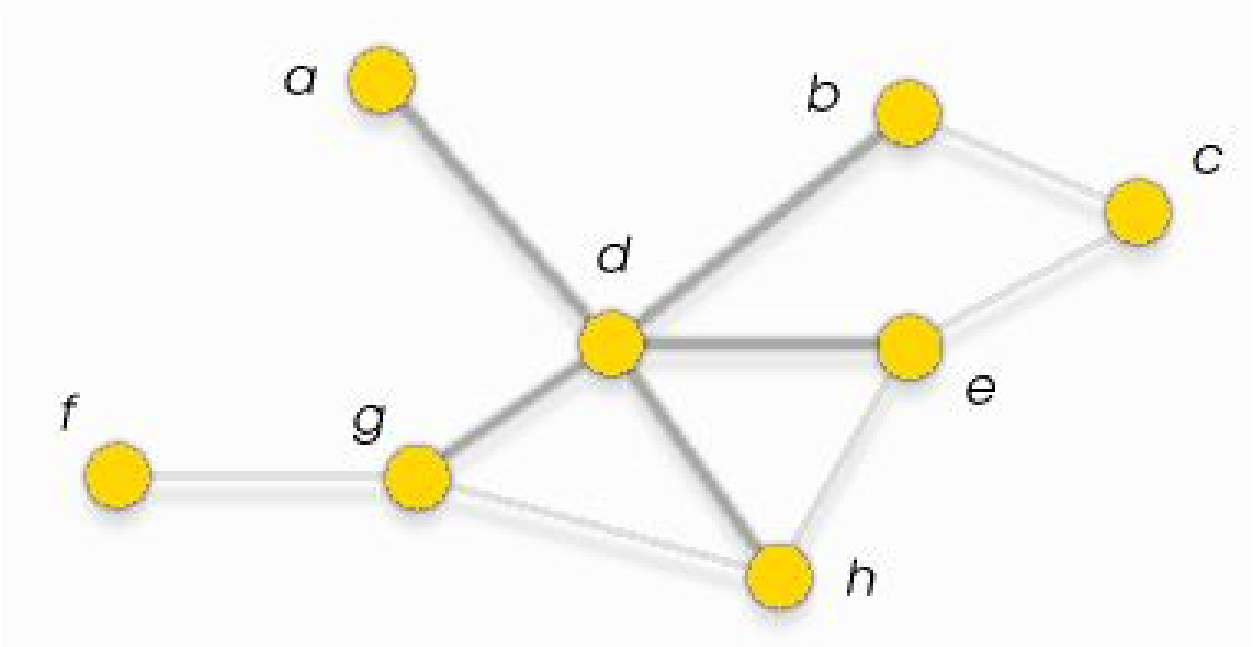
Degré

Degré

Deux sommets x et y sont adjacents si il existe l'arête (x, y) dans E. Les sommets x et y sont alors dits voisins Une arête est incidente à un sommet x si x est l'une de ses extrémités. Le degré d'un sommet x de G est le nombre d'arêtes incidentes à x. Il est noté $d(x)$. Pour un graphe simple le degré de x correspond également au nombre de sommets adjacents à x.

Degré

$d(d) = 5$.
Les arêtes incidentes à d sont : $(d, a), (d, b), (d, e), (d, h) et (d, g)$.



Sous graphe et graphe partiel

Pour caractériser de manière moins locale la structure d'un graphe, il est possible de rechercher des parties remarquables du graphe, en restreignant soit l'ensemble des sommets (sous-graphe), soit l'ensemble des arêtes (graphe partiel).

- Un sous-graphe de G consiste à considérer seulement une partie des sommets de V et les liens induits par E. Par exemple si G représente les liaisons aériennes journalières entre les principales villes du monde, un sous-graphe possible est de se restreindre aux liaisons journalières entre les principales villes européennes.
- Un graphe partiel de G consiste à ne considérer qu'une partie des arêtes de E. En reprenant le même exemple, un graphe partiel possible est de ne considérer que les liaisons journalières de moins de 3 heures entre les principales villes du monde.

Sous graphe et graphe partiel

Sous graphe et graphe partiel

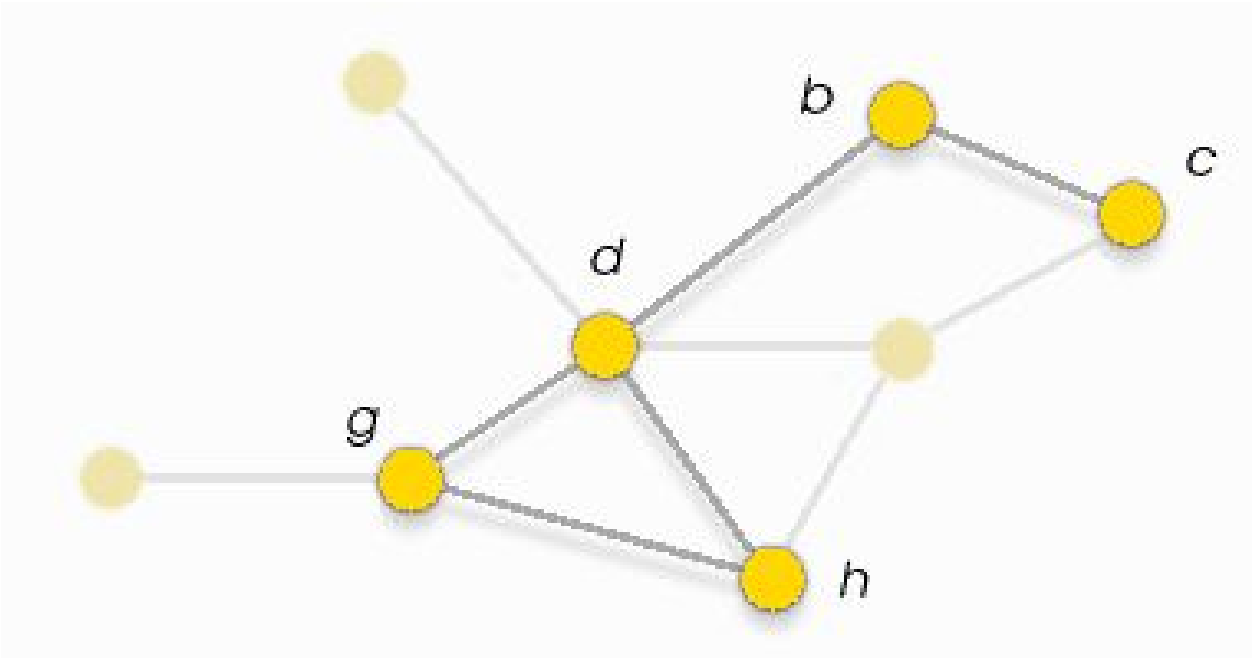
Pour un graphe $G = (V, E)$

- Un sous-graphe de G est un graphe $H = (W, E(W))$ tel que W est un sous-ensemble de V, et $E(W)$ sont les arêtes induites par E sur W, c'est à dire les arêtes de E dont les 2 extrémité sont des sommets de W.
- Un graphe partiel de G est un graphe $I = (V, F)$ tel que F est un sous-ensemble de E.

Un sous-graphe H de G est entièrement défini (induit) par ses sommets W, et un graphe partiel I par ses arêtes F.

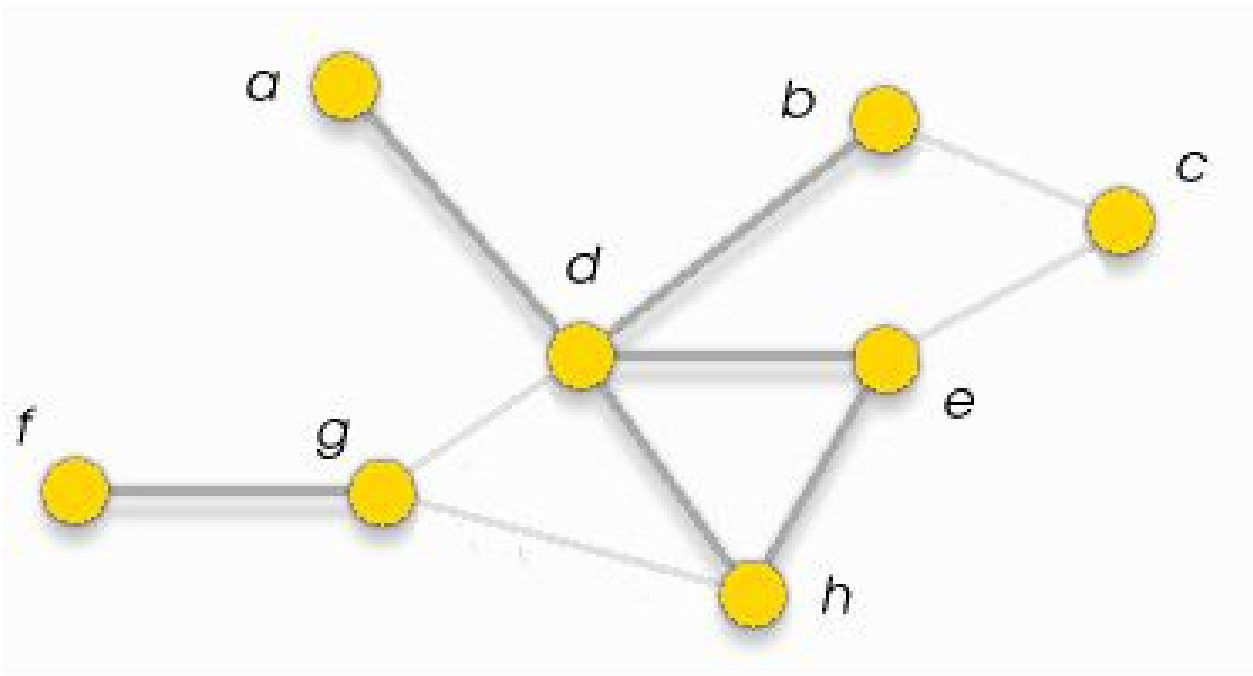
Exemple sous-graphe

Le sous-graphe H induit par l'ensemble $W = \{b, c, d, g, h\}$ de sommets.



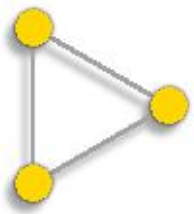
Exemple graphe partiel

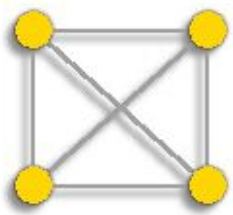
Le graphe partiel I défini par l'ensemble F constitué des arêtes $(a, d), (b, d), (d, e)$, et $(e, h), (h, d), (f, g)$.



Graphe complet

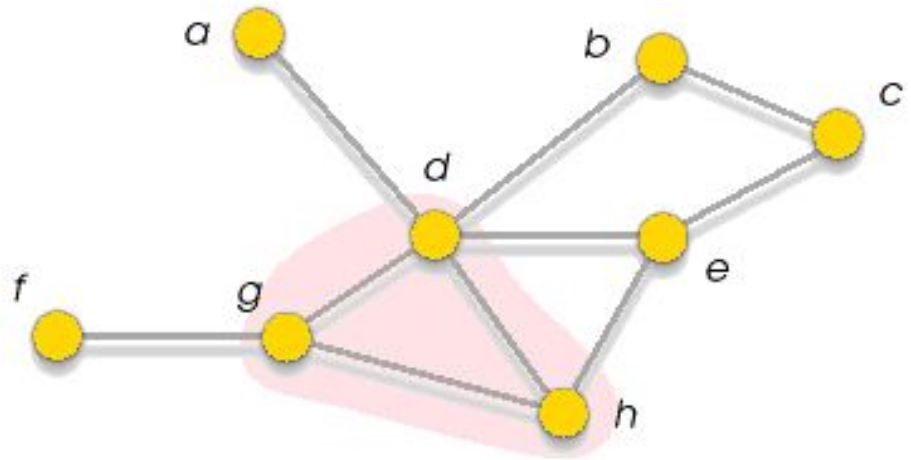
Un graphe complet est un graphe où chaque sommet est relié à tous les autres. Le graphe complet d'ordre n est noté Kn. Dans ce graphe chaque sommet est de degré $n-1$. De gauche à droite sont représentés les graphes $K2, K3$ et $K4$.





Clique et stable

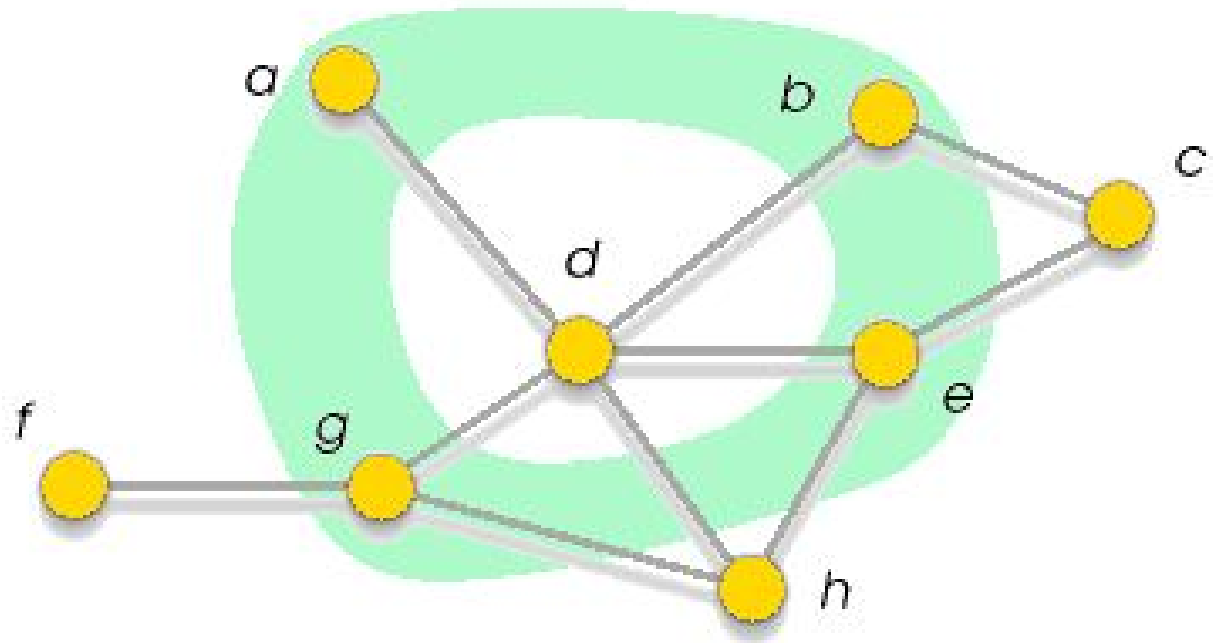
- Une clique est un sous-graphe complet.
- Un stable est un sous-graphe sans arête.



Le graphe G admet 2 cliques d'ordre 3 définies par les ensemble de sommets d, g, h et d, e, h
La clique d, g, h est représentée en surimpression. Le graphe n'admet pas de clique d'ordre 4.

Clique et stable

Le graphe G admet 4 stables d'ordre 4 définis par les ensemble de sommets a, b, e, g et a, b, h, f a, c, h, f et a, b, e, f
Le stable a, b, e, g est représenté en surimpression. Le graphe n'admet pas de stable d'ordre 5.



Outline

Chemin et cycle

Dans un graphe il est naturel de vouloir se déplacer de sommet en sommet en suivant les arêtes. Une telle marche est appelée une chaine ou un chemin. Un certain nombre de questions peuvent alors se poser : pour 2 sommets du graphe, existe-t-il un chemin pour aller de l'un à l'autre? Quel est l'ensemble des sommets que l'on peut atteindre depuis un sommet donné? Comment trouver le plus court chemin pour aller d'un sommet à un autre?

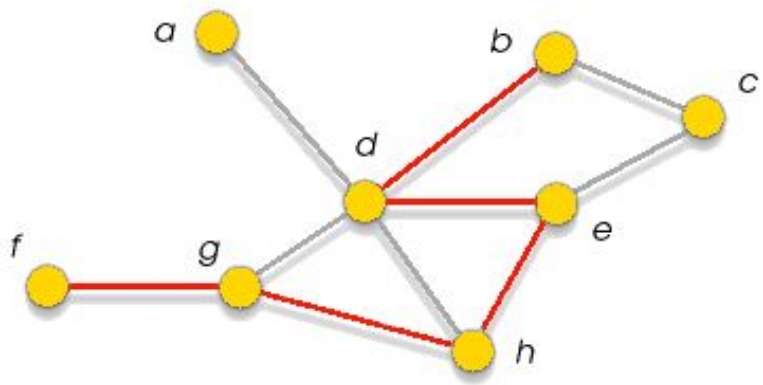
Définitions

Chemin

- Un chemin est une liste $p = (x_1, ..., x_k)$ de sommets telle qu'il existe dans le graphe une arête entre chaque paire de sommets successifs. $\forall i = 1, ..., k-1, (x_i, x_{i+1}) \in E$
- La longueur du chemin correspond au nombre d'arêtes parcourues : $k - 1$.

Chemin et cycle

Un chemin de longueur 5 dans le graphe reliant les sommets f à b. $p = (f, g, h, e, d, b)$



Il existe bien d'autres chemins pour aller de f à b : par exemple (f, g, d, b) de longueur 3, le chemin (f, g, d, h, e, d, b) de longueur 6, ou encore $(f, g, d, h, e, d, h, e, d, b)$ de longueur 9, ... (d, h, e, d) est appelé un cycle. Ce cycle pouvant être emprunté autant de fois que l'on veut, il y a un nombre infini de chemins de f à b.

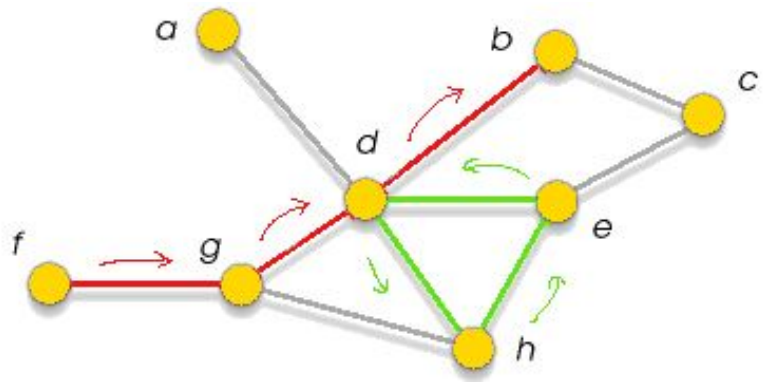
Chemin et cycle

Chemin et cycle

- Un chemin p est simple si chaque arête du chemin est empruntée une seule fois.
- Un cycle $c = (x_1, ..., x_k, x_{k+1})$ est un chemin simple finissant à son point de départ : $x_1 = x_{k+1}$

Chemin et cycle

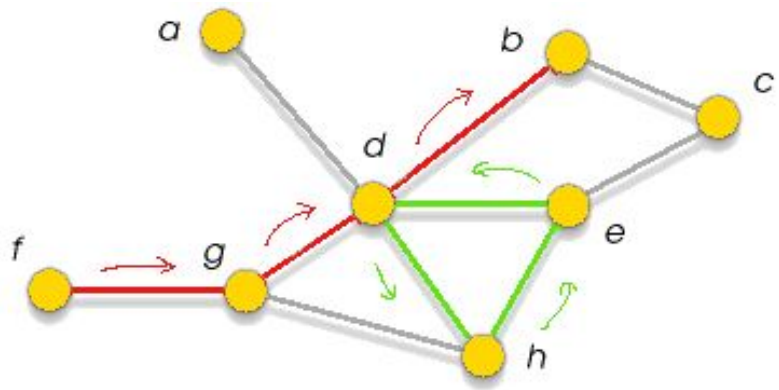
Les chemins (f, g, d, b) et (f, g, d, h, e, d, b) sont simples. Le chemin $(f, g, d, h, e, d, h, e, d, b)$ ne l'est pas : le cycle (d, h, e, d) est emprunté 2 fois.



Les termes de chemin et de circuit s'emploient en propre pour les graphes orientés. Pour les graphes non orientés on parle de chaine et de cycle. Cependant la définition formelle est exactement la même dans les 2 cas, seule change la structure (graphe orienté ou non) sur laquelle ils sont définis.

Chemin élémentaire

- Un chemin $p = (x_1, ..., x_k)$ est élémentaire si chacun des sommets du parcours est visité une seule fois :$\forall i, j = 1, .., k, i \neq j, x_i \neq x_j$
- Un chemin élémentaire est donc un chemin simple et sans cycle.



Le chemin (f, g, d, b) est élémentaire, le chemin (f, g, d, h, e, d, b) ne l'est pas : le sommet d est visité 2 fois, ce qui crée le cycle (d, h, e, d).

Connexité

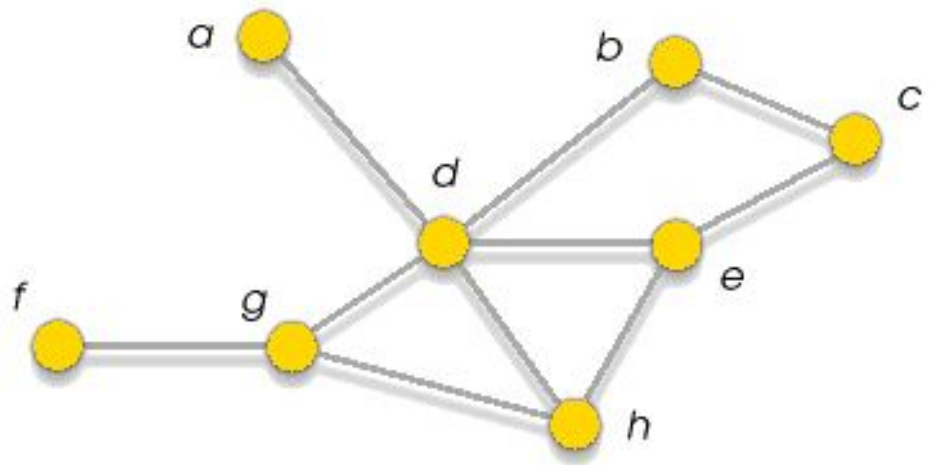
La notion de connexité est liée à l'existence de chemins dans un graphe : depuis un sommet, existe-t-il un chemin pour atteindre tout autre sommet? Les graphes connexes correspondent à la représentation naturelle que l'on se fait d'un graphe. Les graphes non connexes apparaissent comme la juxtaposition d'un ensemble de graphes : ses composantes connexes.

Connexité

Un graphe est connexe ssi il existe un chemin entre chaque paire de sommets.

Connexité

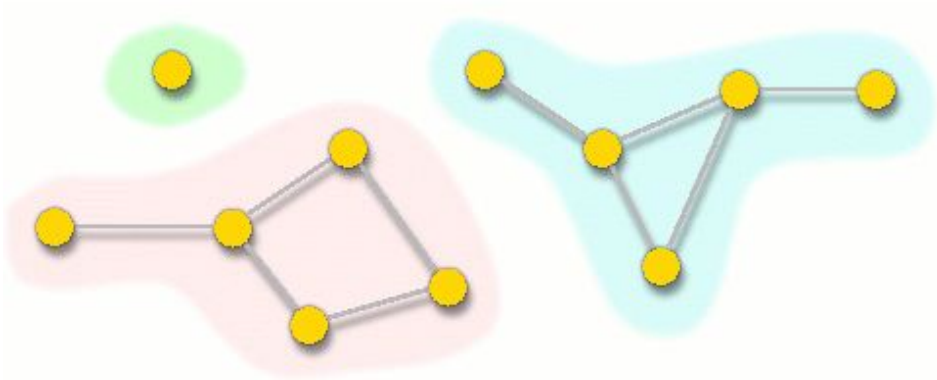
Que se passe-t-il si le graphe G n'est pas connexe? Il apparaît alors comme un ensemble de graphes connexes "mis" les uns à coté des autres. Chacun de ces graphes est un sous-graphe particulier de G, appelé composante connexe. Il est souvent utile de se placer sur les composantes connexes d'un graphe pour se ramener au cas d'un graphe connexe.



Connexité

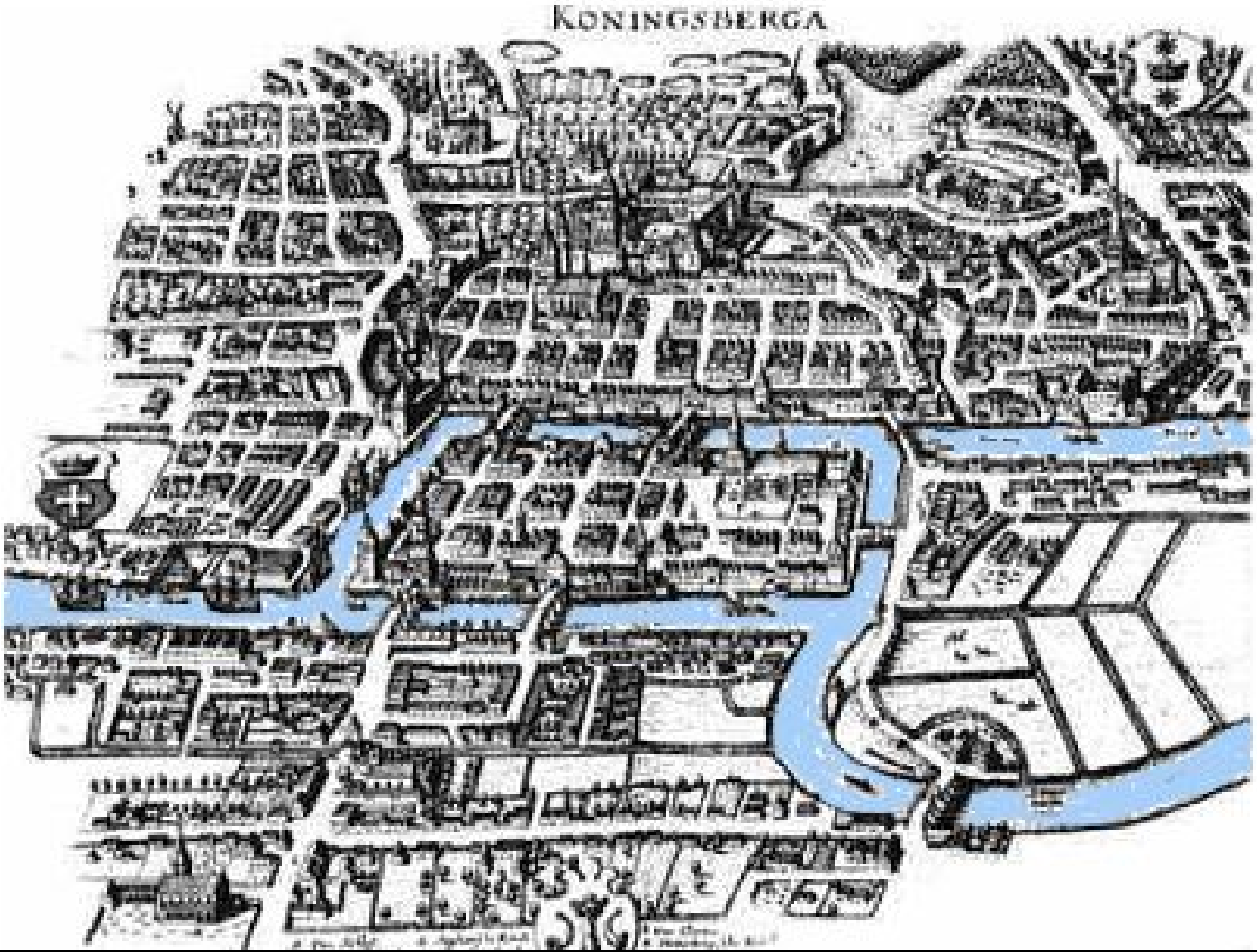
Composante connexe

Une composante connexe d'un graphe G est un sous-graphe $G' = (V', E')$ connexe maximal (pour l'inclusion) : il n'est pas possible d'ajouter à V' d'autres sommets en conservant la connexité du sous-graphe.



Ce graphe possède 3 composantes connexes, dont un sommet isolé.

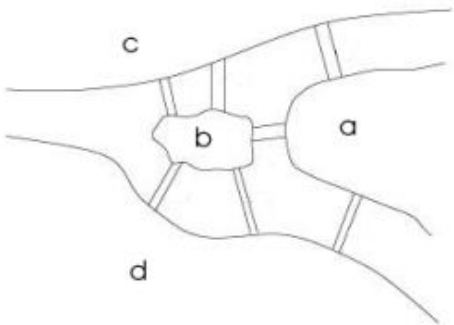
Euler

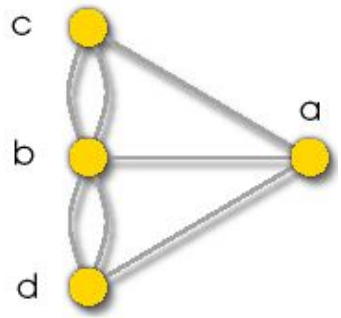


Connexité

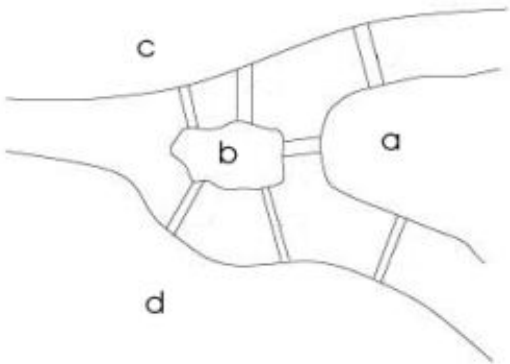
Graphe Eulérien

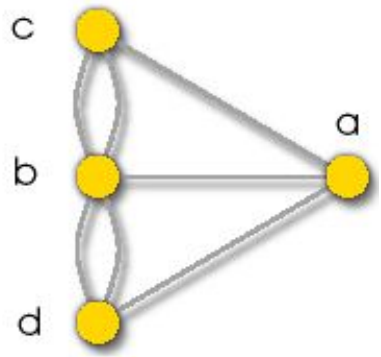
Un cycle eulérien est un cycle passant une et une seule fois par chaque arête du graphe. Un graphe est dit Eulérien si il admet un cycle eulérien.





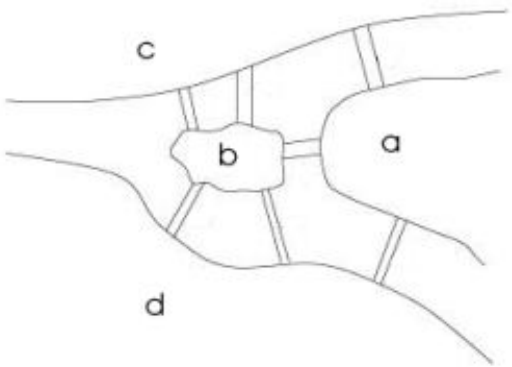
Euler

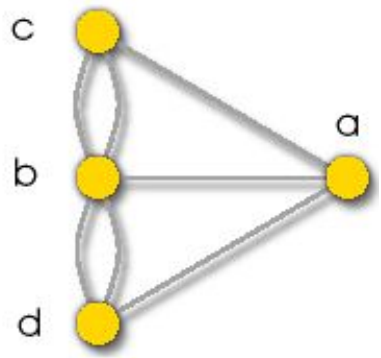




Comment savoir si un graphe est eulérien ou non ?

Euler

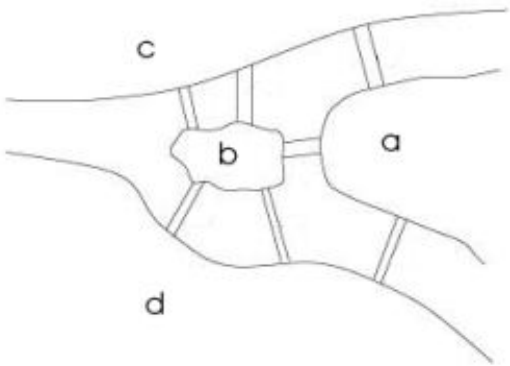


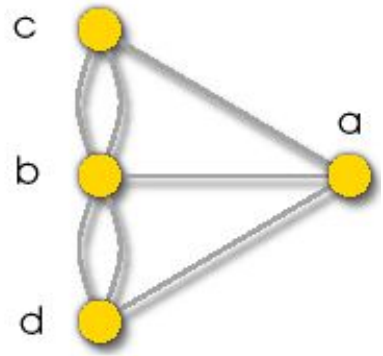


Théorème

Un graphe est eulérien ssi il est connexe et tous ses sommets sont de degré pair.

Euler





Avec cette caractérisation, les sommets a, b, c et d étant de degré impair, on sait immédiatement qu'il est imposible de parcourir tous les ponts de Koenigsberg seulement une fois au cours d'une promenade.

Cheminement

Lorsqu'un chemin existe entre deux sommets dans un graphe, l'être humain se pose rapidement la question non seulement de trouver un tel chemin, mais bien souvent il est intéressé par le plus court chemin possible entre ces deux sommets.

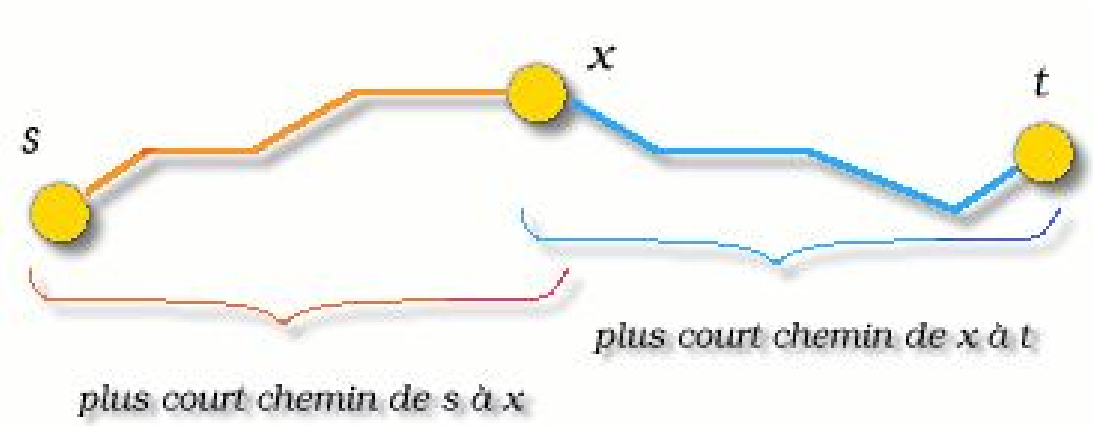
Cheminement

Plus court chemin

Un plus court chemin entre 2 sommets est élémentaire

Principe de sous-optimalité

Si $p = (s, ..., t)$ est un plus court chemin entre s et t, alors pour tout sommet x sur le chemin p, le sous-chemin de p jusqu'à x, $(s, ..., x)$, est un plus court chemin de s à x le sous-chemin de p depuis x, $(x, ..., t)$, est un plus court chemin de x à t.



Recherche

Pour visiter le graphe depuis le sommet s, nous allons utiliser un algorithme de recherche. Cet algorithme est un algorithme de marquage très simple qui correspond bien à l'exploration que nous ferions naturellement d'un graphe en visitant les sommets de proche en proche. Au départ tous les sommets sont non marqués, à l'exception de notre sommet de départ, s. A chaque étape nous sélectionnons simplement un sommet non marqué, adjacent à un sommet déjà marqué, et nous le marquons à son tour. Le choix d'un sommet à marquer voisin d'un sommet déjà marqué assure que l'ensemble des sommets marqués est un sous-graphe connexe à chaque étape de l'algorithme.

Recherche

ALGORITHME Recherche

ENTREES Graphe $G = (V, E)$, Sommet s

Initialiser tous les sommets à non marqué ;
Marquer s;
TantQue il existe un sommet non marqué adjacent à un sommet marqué
Sélectionner un sommet y non marqué adjacent à un sommet marqué;
Marquer y;
FinTantQue

Recherche

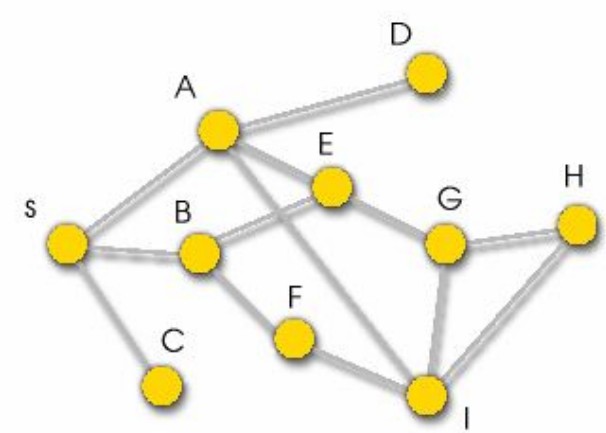
Il existe deux grandes stratégies "opposées" pour sélectionner à chaque étape le sommet à marquer :

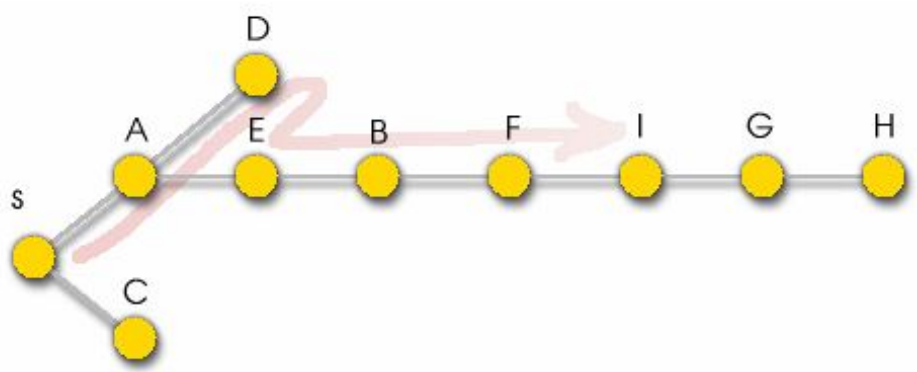
Stratégies de recherche

- La recherche en profondeur DFS (Depth First Search) Dans l'exploration, l'algorithme cherche à aller très vite "profondément" dans le graphe, en s'éloignant du sommet s de départ. La recherche sélectionne à chaque étape un sommet voisin du sommet marqué à l'étape précédente.
- La recherche en largeur BFS (Breath First Search). Dans l'exploration l'algorithme cherche au contraire à épuiser la liste des sommets proches de s avant de poursuivre l'exploration du graphe.

Recherche

Un exemple de recherches possibles en profondeur et en largeur. Le résultat de chacune des recherche est représenté par son arbre de recherche : une arête existe entre deux sommets x et y si la visite (marquage) de y se fait à partir de x.

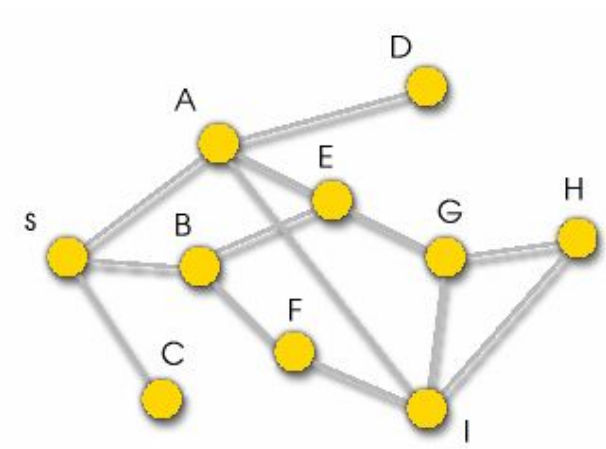


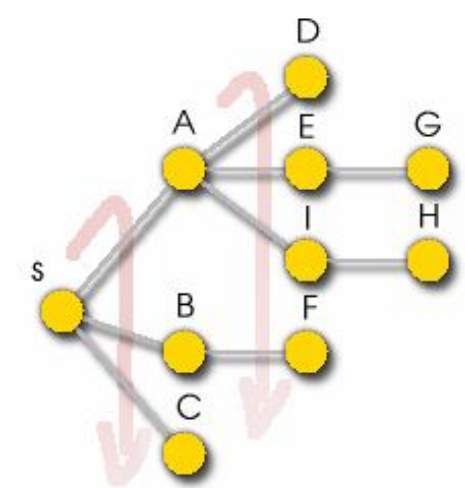


Les sommets sont visités depuis s dans l'ordre $A, D, E, B, F, I, G, H, C$.

Recherche

Un exemple de recherches possibles en profondeur et en largeur. Le résultat de chacune des recherche est représenté par son arbre de recherche : une arête existe entre deux sommets x et y si la visite (marquage) de y se fait à partir de x.





Les sommets sont visités depuis s dans l'ordre $A, B, C, D, E, I, F, G, H$.

Recherche améliorée

ALGORITHME Recherche

ENTREES Graphe $G = (V, E)$, Sommet s

M : structure de données ordonnée
Initialiser tous les sommets à non marqué ; Marquer s
$M \leftarrow s$
Tant Que M n'est pas vide
$x \leftarrow M$ // Retirer le "premier" sommet de l'ensemble M
Pour chaque voisin y non marqué de x
Marquer y
$M \leftarrow y$ // Ajouter y à l'ensemble M
Fin Pour
Fin TantQue

Recherche

- La Recherche en Profondeur correspond à prendre pour M une structure de PILE, c'est à dire une liste LIFO LastIn/FirstOut (dernier entré/premier sorti)
- La Recherche en Largeur correspond à prendre pour M une structure de FILE, c'est à dire une liste FIFO FirstIn/FirstOut (premier entré/premier sorti)

Plus court chemin

Avec la recherche en largeur, nous avons toutes les cartes en main pour écrire un algorithme calculant la longueur de tous les plus court chemins depuis un sommet s. Il suffit au cours de la visite de mettre à jour un label L pour chaque sommet. Le label $L(y)$ du sommet y est calculé comme le label $L(x)$ de son voisin x depuis lequel il est visité, plus 1. A la fin de l'algorithme, les labels sont égaux aux distances $D(y)$.

Plus court chemin

ALGORITHME BFS

ENTREES Graphe $G = (V, E)$, Sommet s F : FILE (liste FIFO)
Initialiser tous les sommets à non marqué ; Marquer s
$L(s) := 0$
$F \leftarrow s$
Tant Que F n'est pas vide
$x \leftarrow F$ // Retirer le premier sommet de la file
Pour chaque voisin y non marqué de x
Marquer y
$L(y) := L(x) + 1$
$F \leftarrow y$ // Ajouter y à la fin de la file
Fin Pour
Fin TantQue

Plus court chemin

Nous allons généraliser notre notion de plus court chemin dans le cas de graphes valués, où chaque arête e est associée à une valeur, appelée souvent son poids, $c(e)$. La valuation des arêtes peut représenter des coûts de transit, des distances kilométriques, le temps nécessaire pour parcourir les arêtes.

Plus court chemin

Dans un graphe valué, le poids $c(p)$ d'un chemin p est la somme des poids des arêtes le long du chemin. Dans ce qui suit nous appellerons le poids d'un chemin sa longueur. Le plus court chemin entre 2 sommets s et t est alors défini comme le chemin de plus faible poids reliant s et t.

Introduction et principe d'optimalité de Bellman

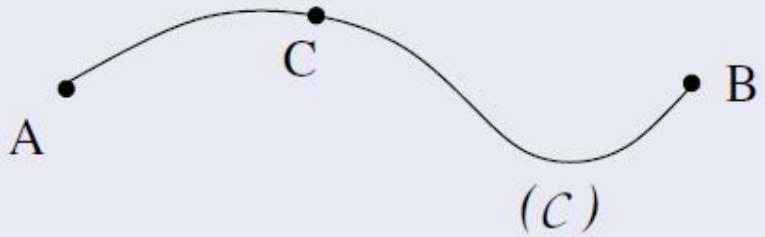
Programmation dynamique

- C'est une méthode de construction d'algorithme très utilisée en optimisation combinatoire ($\rightarrow$ recherche de solution optimale dans un ensemble **fini** de solutions **mais très grand**).
- Il s'agit d'une méthode d'énumération implicite (idem PSE) : on retient ou rejette des sous-ensembles de solutions mais on ne construit pas toutes les solutions. On rejette certaines solutions sans les avoir contruites explicitement si elles appartiennent à un sous-ensemble qui n'est pas intéressant.
- Programmation dynamique développée par Bellman ($\approx$ 1954) pour résoudre des pb de chemins optimaux (longueur max. ou min.)

Introduction et principe d'optimalité de Bellman

Principe d'optimalité de Bellman

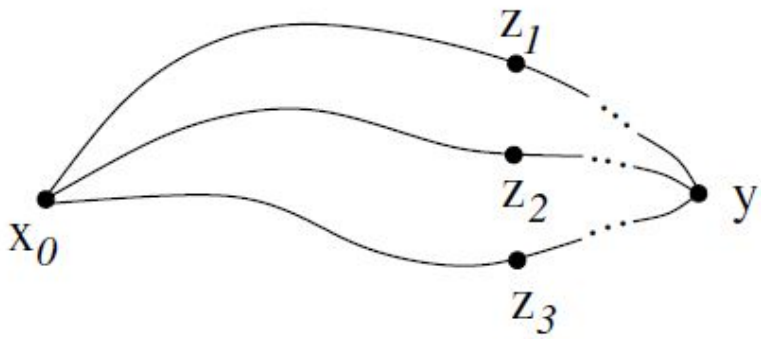
Un chemin optimal est formé de sous-chemins optimaux : Si (C) est un chemin optimal allant de A à B et si C appartient à (C) alors les sous-chemins de (C) allant de A à C et de C à B sont optimaux.



Introduction et principe d'optimalité de Bellman

Un exemple

On cherche le chemin le plus court allant de x_0 fixé à y quelconque dans un graphe valué.



On note $F(y)$ la longueur minimale de tous les chemins allant de x_0 à y.

Formule récursive

$$F(y) = min_z(F(z) + L(z, y))$$

où $L(z, y)$ est la longueur minimale de tous les chemins allant de z à y.

Introduction et principe d'optimalité de Bellman

- La programmation dynamique appliquée à un problème donné, consiste à trouver une **formulation récursive** du problème.
- En procédant ensuite à un découpage étape par étape, on obtient une **formule de récurrence**.

Plus court chemin

Dans un graphe avec une valuation positive $c()$ de ses arêtes, un plus court chemin entre 2 sommets est élémentaire.

Principe de sous-optimalité

Les plus courts chemins vérifient le principe de sous-optimalité : si $p = (s, ..., t)$ est un plus court chemin entre s et t, alors pour tout sommet x sur le chemin, le sous-chemin de p jusqu'à x, $(s, ..., x)$, est un plus court chemin de s à x.
Si $D(y)$ est la longueur du plus court chemin d'un sommet y à s, le principe de sous-optimalité se traduit localement par les égalités :

$$D(y) = 0 \text{ si } y = s$$
$$D(y) = min\{D(x) + c(x, y) | x \text{ voisin de } y\} \text{ sinon}$$

Distance partielle

Distance partielle

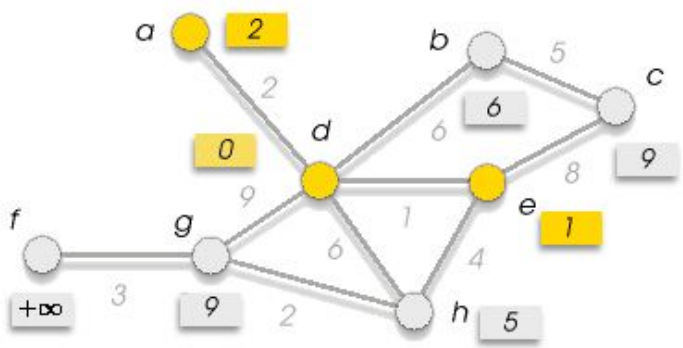
Considérons un ensemble S de sommets incluant la source s. Pour tout sommet z en dehors de S, nous définissons sa distance partielle à s par :
$DP(S,z) = min\{D(x) + c(x,z) | x \in S \text{ et } x \text{ voisin de } z\}$
Alors si y est un sommet avec la plus petite distance partielle, sa distance partielle est égale à sa distance à s :

$$DP(S,y) = D(y)$$

.

La distance partielle correspond à la longueur du plus court chemin de s à z qui n'emprunte que des sommets de S, à l'exception bien sûr du dernier sommet z. C'est donc le plus court chemin dans le sous-graphe induit par $S \bigcup \{z\}$.
Par convention la distance partielle d'un sommet est infinie si il n'est adjacent à aucun sommet de S.

Distance partielle



Le sommet d joue le rôle de la source s. L'ensemble S des sommets marqués est $\{a, d, e\}$. Les distances sont indiquées à coté de chaque sommet, en grisé pour les distances partielles des sommets non marqués, en jaune pour les plus courtes distances des sommets marqués. La distance partielle de f est infinie, sa distance à d est $D(f) = 10$. La distance partielle de g est $DP(S, g) = 9$, sa distance est $D(g) = 7$, pour h, le sommet de plus petite distance partielle, on a bien l'égalité $DP(h) = D(h) = 5$.

Plus court chemin

Nous sommes maintenant prêts pour donner un algorithme de recherche des plus court chemins. L'idée est de marquer les noeuds du graphe en sélectionnant à chaque étape le sommet non marqué de plus petite distance partielle. Un label est maintenu pour chaque sommet : à la fin de l'algorithme ce label est égal à sa distance à s. L'algorithme suit le schéma de notre algorithme de recherche : à chaque étape le sommet y sélectionné est en effet adjacent à un sommet déjà marqué, sinon son label serait infini.

ALGORITHME Recherche des plus court chemins

ENTREES $G = (V, E)$ un graphe avec une valuation positive c des arêtes, s un sommet de V

Initialiser tous les sommets à non marqué ; Marquer s
$L(s) := 0$ // Initialise le label de s à 0
Tant Que il existe un sommet non marqué
Pour chaque sommet y non marqué
Calculer $L(y) := min\{L(x) + c(x, y) | x$ voisin marqué de $y\}$
Fin Pour
Choisir le sommet y non marqué de plus petit label L
Marquer y
Fin TantQue

Plus court chemin

Plus court chemin

Edsger Wybe Dijkstra (1930-2002)

Plus court chemin

L'algorithme de Dijkstra est une implémentation efficace de l'algorithme de recherche des plus courts chemins. Dans cet algorithme les opérations les plus coûteuses sont le calcul à chaque étape des distances partielles. Cependant une bonne partie de ces opérations sont inutiles : lors du marquage d'un sommet x, seuls les voisins de x peuvent éventuellement voire leur distance partielle modifiée.

Mise à jour des distances partielles

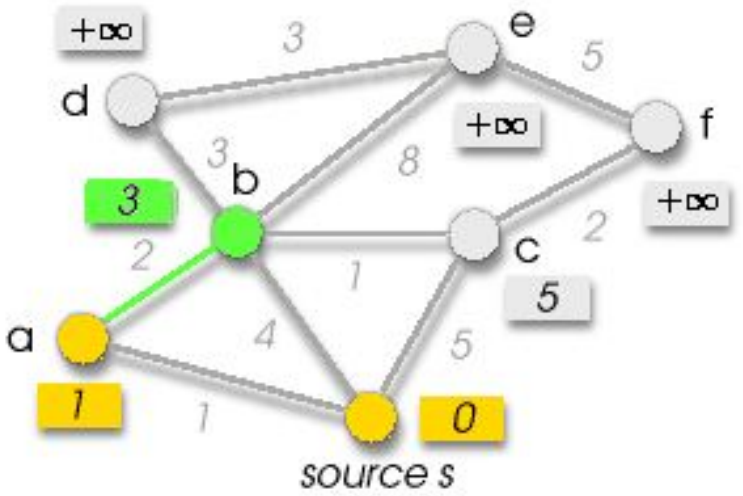
Considérons un ensemble S de sommets incluant la source s, et un sommet y.
Pour tout sommet z en dehors de $S \cup \{y\}$:
$DP(S \cup \{y\}, z) = min\{DP(S, z), D(y) + c(y, z)\}\ si (y, z) \in E$
$DP(S \cup \{y\}, z) = DP(S, z)$ Sinon.

L'algorithme de Dijkstra

L'algorithme de Dijkstra utilise cette propriété pour ne mettre à jour à chaque étape que les distances partielles des sommets adjacents au sommet qui vient d'être marqué. Initialement toutes les labels, représentant les distances, sont initialisés à $+\infty$, sauf celui de la source s mis à 0. Chaque étape de l'algorithme comporte ensuite 2 phases :

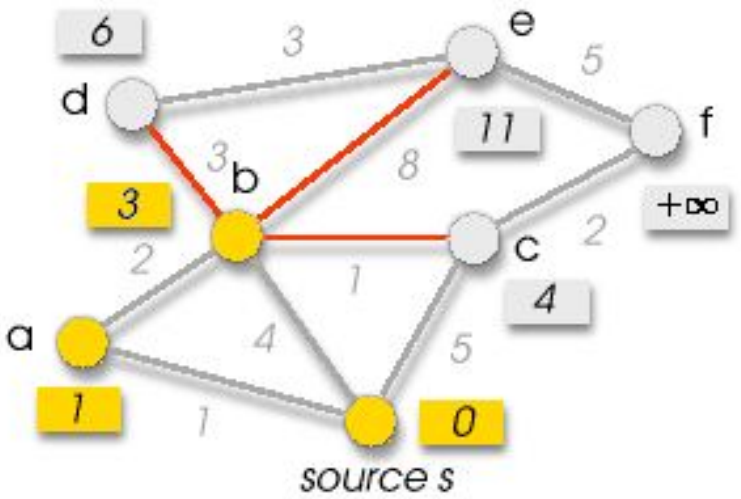
- Une phase de sélection

Comme pour l'algorithme de recherche des plus courts chemins, le sommet y non marqué de plus petit label est sélectionné. Sur l'exemple le sommet b de label 3 est le sommet sélectionné à la deuxième étape.



- Une phase de mise à jour

Les labels des sommets z non marqués adjacents au sommet y qui vient d'être sélectionné sont mis à jour : $L(z) = min\{L(z), L(y) + c(y, z)\}$ Sur l'exemple les labels des sommets c, d et e sont mis à jour.



L'algorithme de Dijkstra

Dijkstra

ALGORITHME Dijkstra ENTREES $G = (V, E)$ un graphe avec une valuation positive c des arêtes, s un sommet de V

Initialiser tous les sommets à non marqué ; Initialiser tous les labels L à $+\infty$
$L(s) := 0$
Tant Que il existe un sommet non marqué
// Sélection du plus 'proche' sommet non marqué
Choisir le sommet y non marqué de plus petit label L
Marquer y // Mise à jour des labels de ses voisins non marqués
Pour chaque sommet z non marqué voisin de y
$L(z) := min\{L(z), L(y) + c(y, z)\}$
Fin Pour Fin TantQue

L'algorithme de Dijkstra

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).

Dijkstra

/* Initialisations */ calculés $= \{A\}$;
$m(A) = 0$;
provisoires $= \emptyset$;
pour tout sommet y successeur de A
provisoires $=$ provisoires $\cup \{y\}$;
$m(y) = v(A, y)$;
$p(y) = A$;
fpour

Dijkstra-suite

/* Itérations */ tant que provisoires $\neq \emptyset$
soit $x \in$ provisoires tel que $m(x)$ est minimum sur provisoires ;
calculés = calculés $\cup \{x\}$; provisoires = provisoires$- \{x\}$;
pour tout sommet y successeur de x
cas
$y \in$ calculés : rien
$y \in$ provisoires :
si $m(x) + v(x, y) < m(y)$
$m(y) = m(x) + v(x, y)$; $p(y) = x$;
fsi
autrement :
provisoires = provisoires $\cup \{y\}$;
$m(y) = m(x) + v(x, y)$; $p(y) = x$;
fcas
fpour
ftantque
/* les sommets n'appartenant pas à calculés sont inaccessibles */

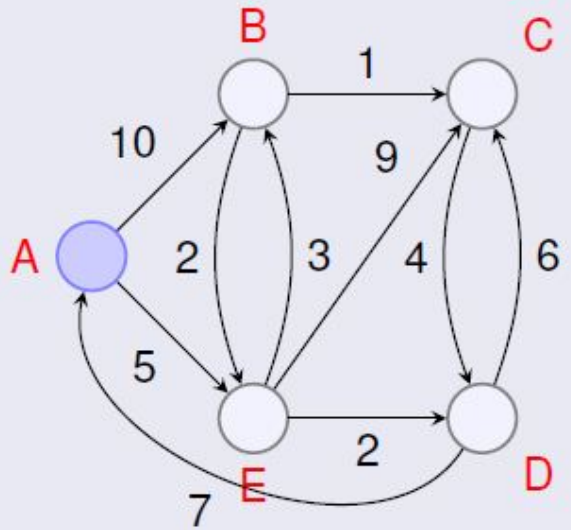
L'algorithme de Dijkstra

Complexité

L'algorithme de Dijkstra calcule en temps $O(N^2)$ (avec N le nombre de sommets du graphe) la longueur des plus courts chemins du sommet s à tous les autres sommets du graphe.

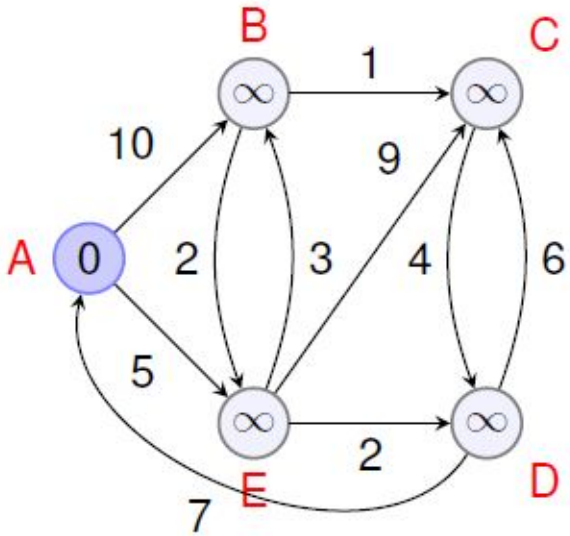
L'algorithme de Dijkstra : exemple

Cherchons les plus courts chemins d'origine *A* dans ce graphe:



L'algorithme de Dijkstra : exemple

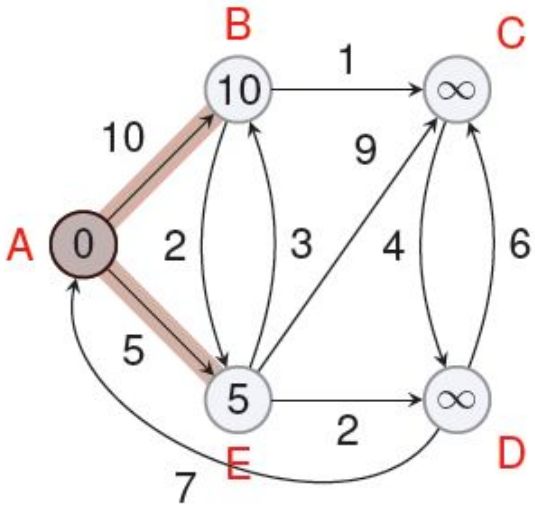
On se place au sommet de plus petit poids, ici le sommet *A*.



A	B	C	D	E
0	∞	∞	∞	∞
•				
•				
•				
•				
•				

L'algorithme de Dijkstra : exemple

On étudie chacune des arêtes partant du sommet choisi.

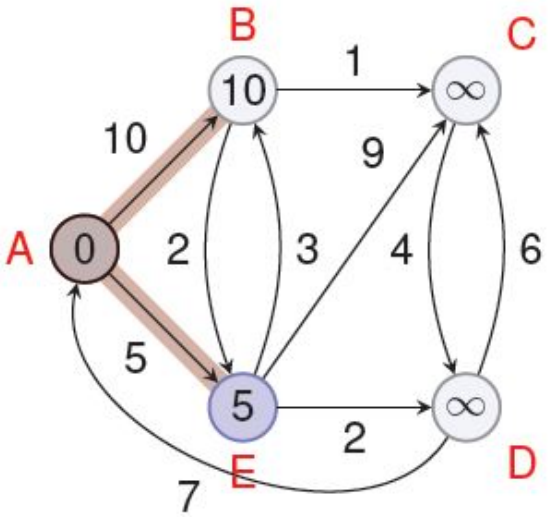


A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	5_A
•				
•				
•				
•				

Dans les colonnes, on mets la distance à A, et le sommet d'où l'on vient.

L'algorithme de Dijkstra : exemple

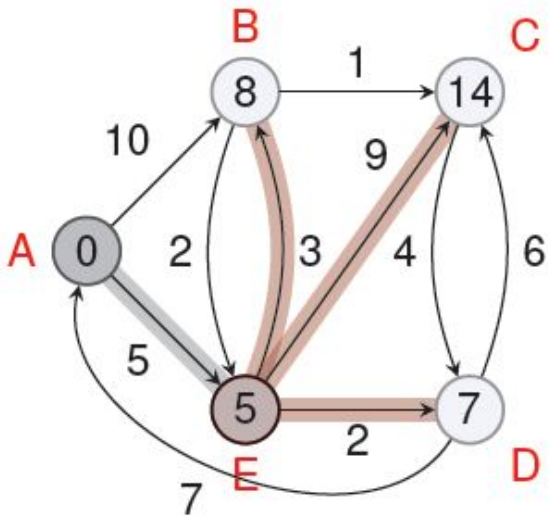
On se place de nouveau au sommet de plus petit poids, ici E.



A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	$\boxed{5_A}$
•				•
•				•
•				•
•				•

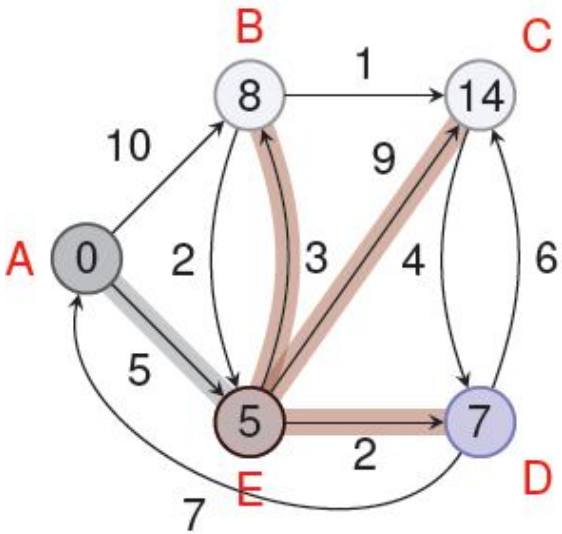
L'algorithme de Dijkstra : exemple

Et ainsi de suite.



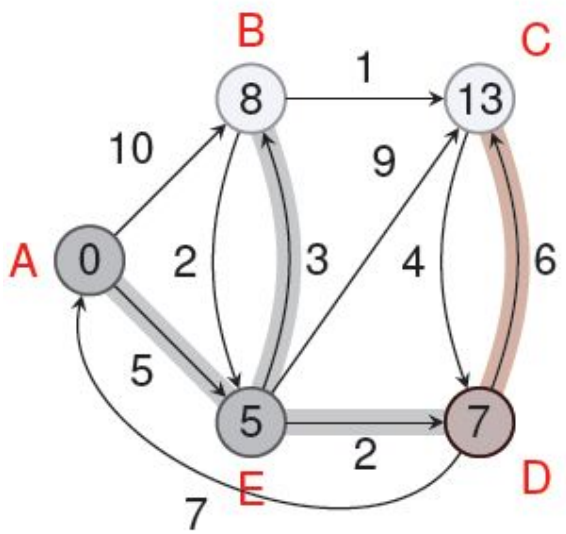
A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	$\boxed{5_A}$
•	8_E	14_E	7_E	•
•				•
•				•

L'algorithme de Dijkstra : exemple



A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	5_A
•	8_E	14_E	$\boxed{7_E}$	•
•			•	•
•			•	•

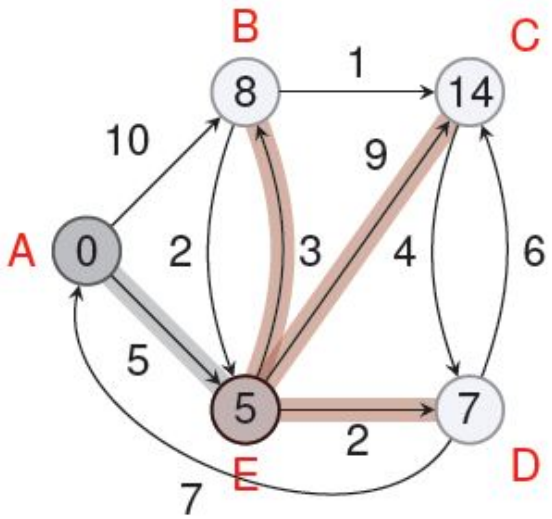
L'algorithme de Dijkstra : exemple



A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	5_A
•	8_E	14_E	$\boxed{7_E}$	•
•	8_E	13_D	•	•
•			•	•
•			•	•

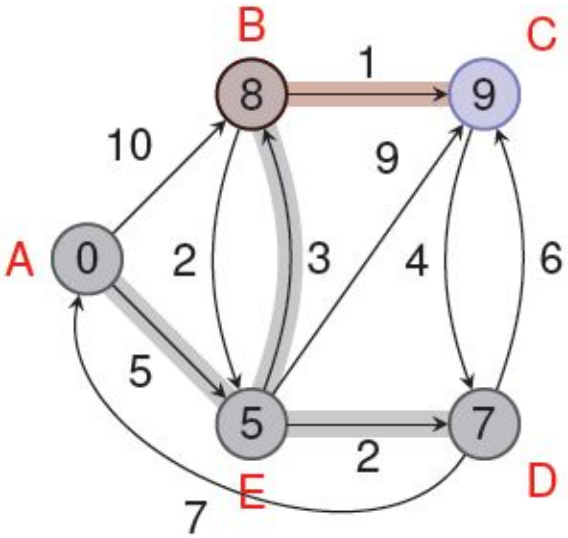
L'algorithme de Dijkstra : exemple

Et ainsi de suite.



A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	$\boxed{5_A}$
•	8_E	14_E	7_E	•
•				•
•				•

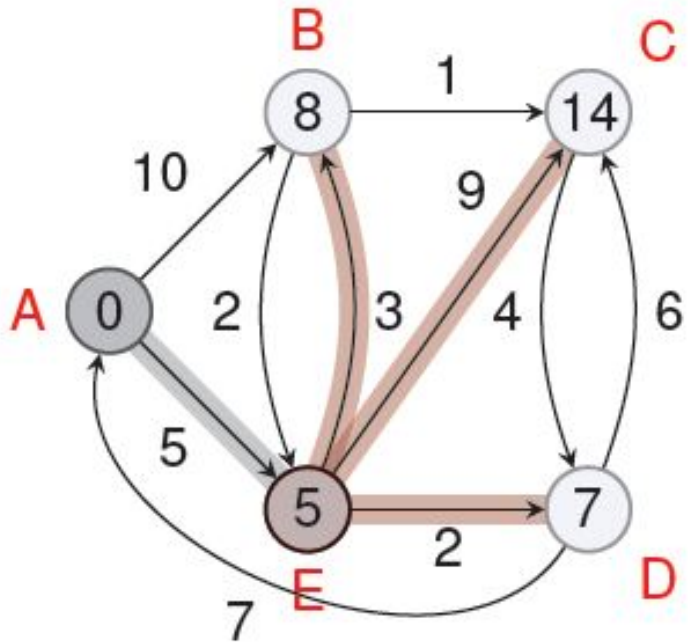
L'algorithme de Dijkstra : exemple



A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	5_A
•	8_E	14_E	7_E	•
•	8_E	13_D	•	•
•	•	$\boxed{9_B}$	•	•
•	•	•	•	•

L'algorithme de Dijkstra : exemple

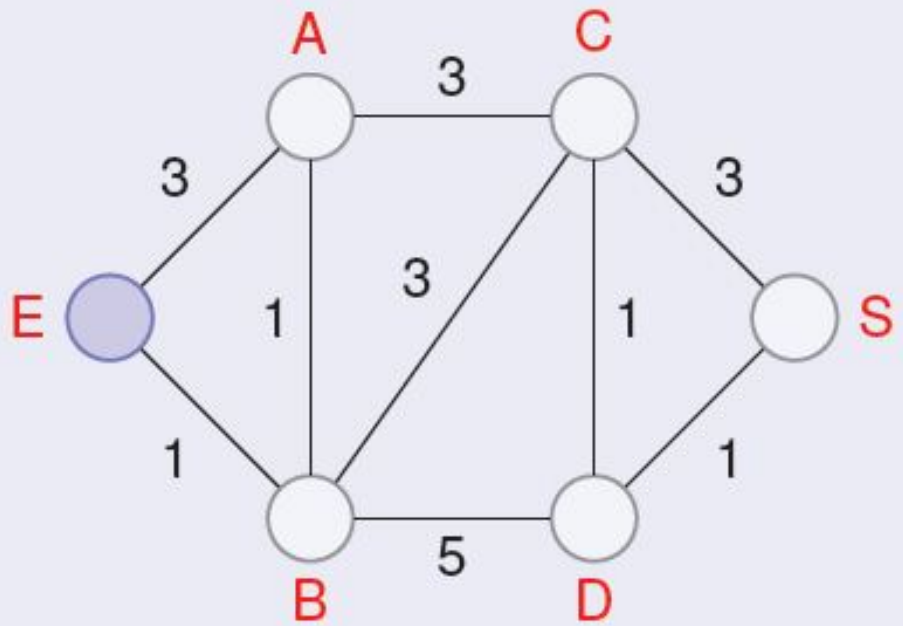
Et ainsi de suite.



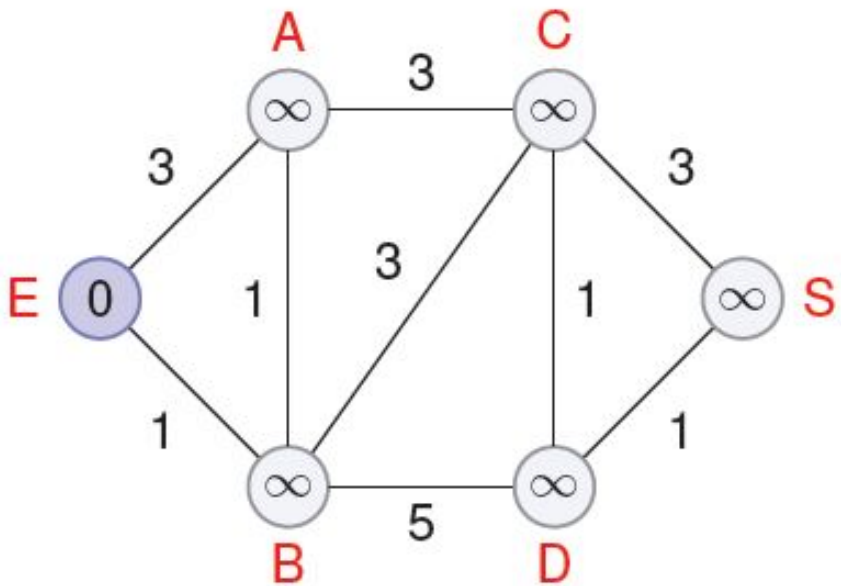
A	B	C	D	E
0	∞	∞	∞	∞
•	10_A	∞	∞	$\boxed{5_A}$
•	8_E	14_E	7_E	•
•				•
•				•

L'algorithme de Dijkstra : exemple

Cherchons les plus courts chemins d'origine *E* dans ce graphe:

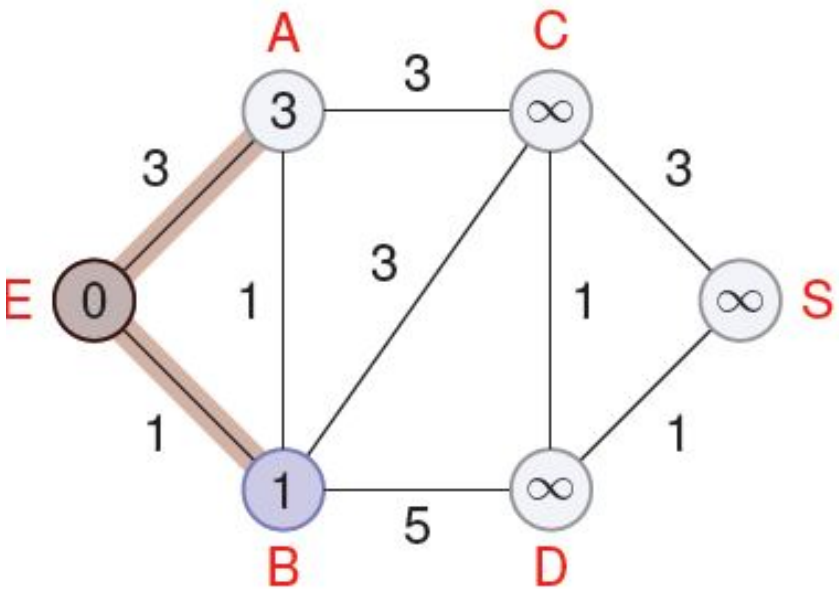


L'algorithme de Dijkstra : exemple



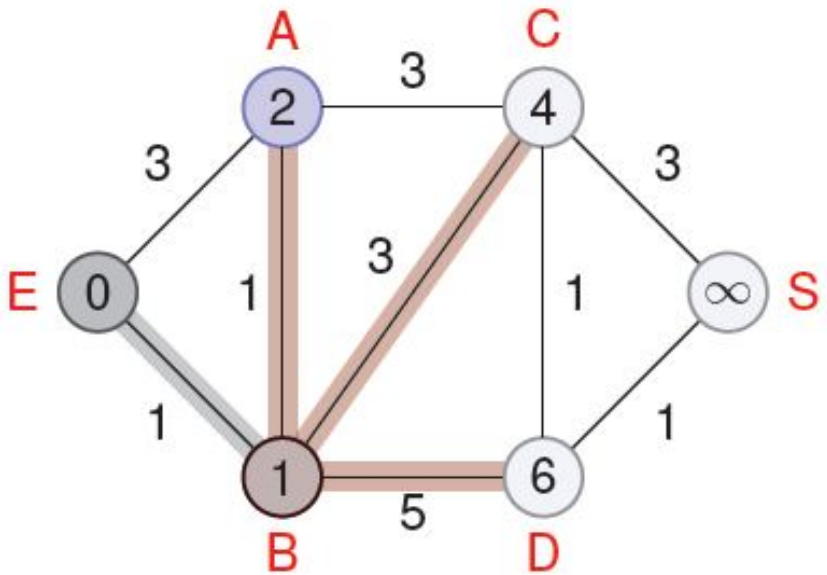
E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•					
•					
•					
•					
•					

L'algorithme de Dijkstra : exemple



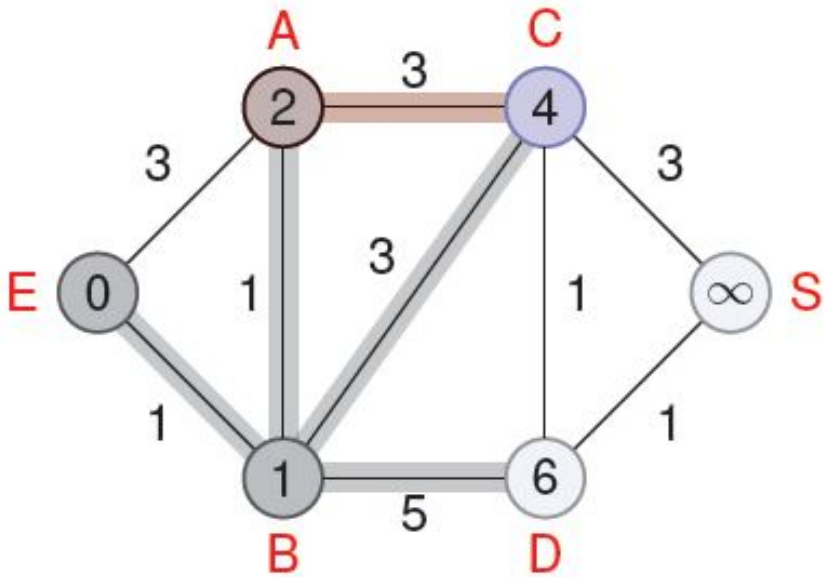
E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•	3_E	$\boxed{1_E}$	∞	∞	∞
•		•			
•		•			
•		•			
•		•			

L'algorithme de Dijkstra : exemple



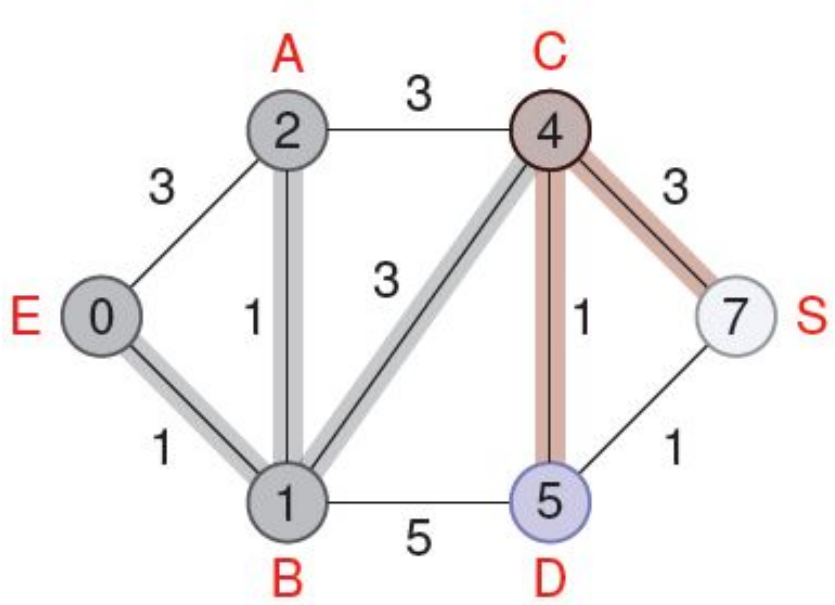
E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•	3_E	1_E	∞	∞	∞
•	$\boxed{2_B}$	•	4_B	6_B	∞
•	•	•			
•	•	•			
•	•	•			

L'algorithme de Dijkstra : exemple



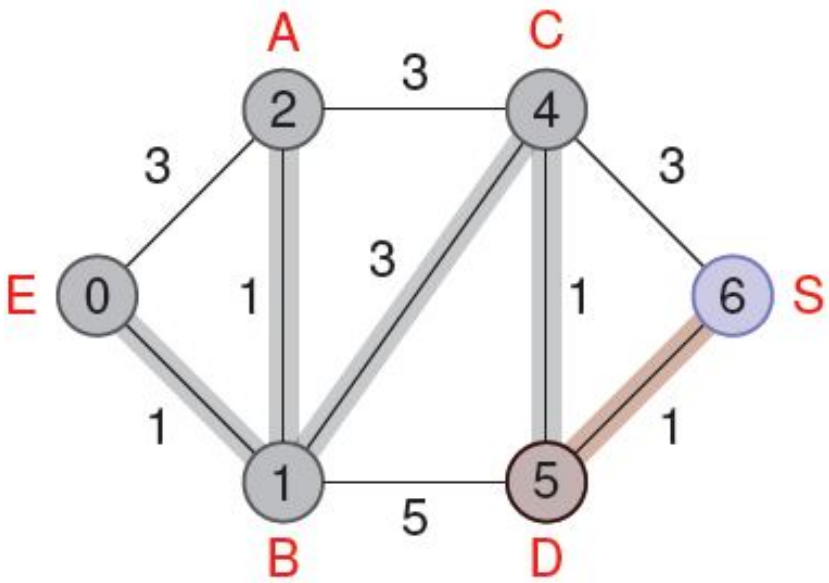
E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•	3_E	1_E	∞	∞	∞
•	2_B	•	4_B	6_B	∞
•	•	•	$\boxed{4_B}$	6_B	∞
•	•	•	•		
•	•	•	•		

L'algorithme de Dijkstra : exemple



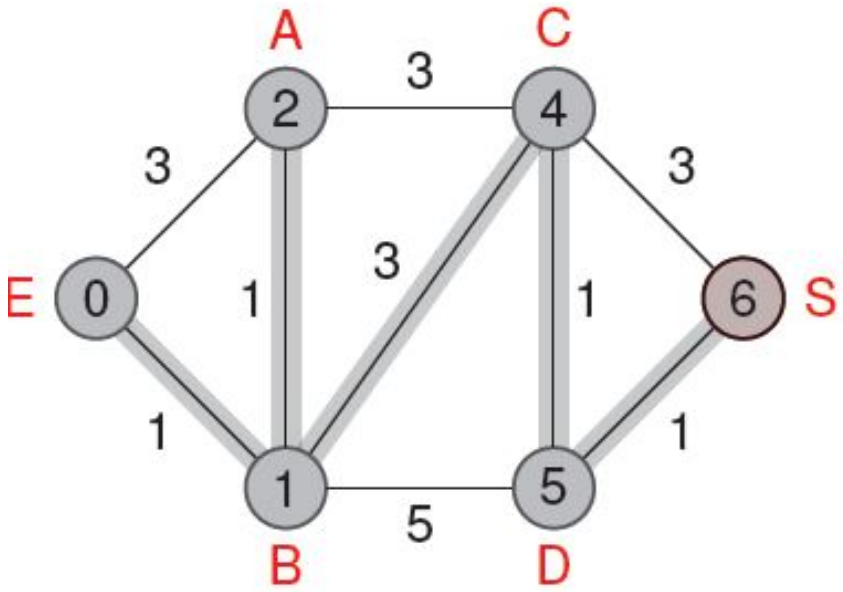
E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•	3_E	1_E	∞	∞	∞
•	2_B	•	4_B	6_B	∞
•	•	•	4_B	6_B	∞
•	•	•	•	$\boxed{5_C}$	7_C
•	•	•	•	•	

L'algorithme de Dijkstra : exemple



E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•	3_E	1_E	∞	∞	∞
•	2_B	•	4_B	6_B	∞
•	•	•	4_B	6_B	∞
•	•	•	•	5_C	7_C
•	•	•	•	•	$\boxed{6_D}$

L'algorithme de Dijkstra : exemple



E	A	B	C	D	S
0	∞	∞	∞	∞	∞
•	3_E	1_E	∞	∞	∞
•	2_B	•	4_B	6_B	∞
•	•	•	4_B	6_B	∞
•	•	•	•	5_C	7_C
•	•	•	•	•	6_B

L'algorithme de Dijkstra : animation

http://interstices.info/upload/chemin/dij09.swf

L'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).

L'algorithme général de Roy-Warshall-Floyd

- 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.

L'algorithme général de Roy-Warshall-Floyd

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 $m(x, y) = +\infty$ si l'arc est inexistant.

L'algorithme général de Roy-Warshall-Floyd

- Une première opération, nommée υ_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
$m(x, y) = min(m(x, y), m(x, 1) + m(1, y))$.

L'algorithme général de Roy-Warshall-Floyd

- Puis une deuxième opération, nommée υ_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 υ_1.
- Et ainsi de suite, avec les opérations υ_3, ..., υ_k, ...,υ_n.

L'algorithme général de Roy-Warshall-Floyd

On peut voir que, à l'issue de v_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 $\leq k$ (ces chemins sont donc de longueur $\leq k + 1$).
Par conséquent, à l'issue de v_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 général de Roy-Warshall-Floyd

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 $m(x, y) = +\infty$;
fsi
fpour
fpour

/* opérations v */
pour k depuis 1 jqa n
/* opération v_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

L'algorithme général de Roy-Warshall-Floyd

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).

L'algorithme général de Roy-Warshall-Floyd

Ces tables sont mises à jour par les opérations υ, 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 υ_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 $R(x)[y] = R(x)[k]$.

L'algorithme général de Roy-Warshall-Floyd

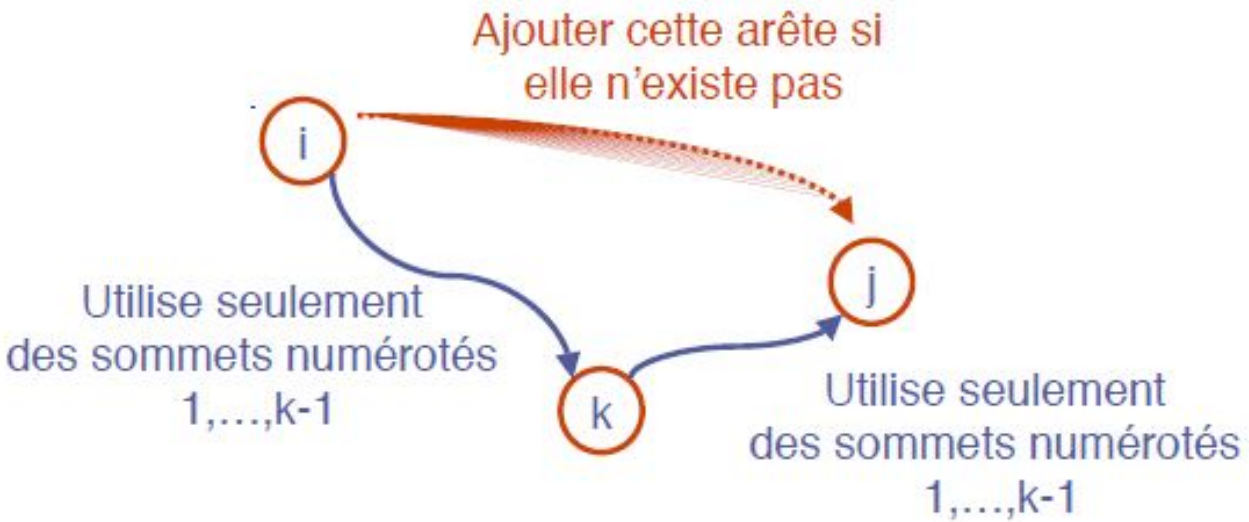
/* initialisation */
pour x depuis 1 jqa n
pour y depuis 1 jqa n
si y est successeur de x
$m(x, y) = v(x, y)$;
$R(x)[y] = y$;
sinon $m(x, y) = +\infty; R[x](y) = 0$;
fsi
fpour
fpour

L'algorithme général de Roy-Warshall-Floyd

/* opérations v */
pour k depuis 1 jqa n
/* opération v_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 général de Roy-Warshall-Floyd

- Idée 1: Numéroter les sommets de 1 à n
- Idée 2: À l'étape k, considérer les chemins utilisant seulement les sommets de 1 à k comme sommets intermédiaires

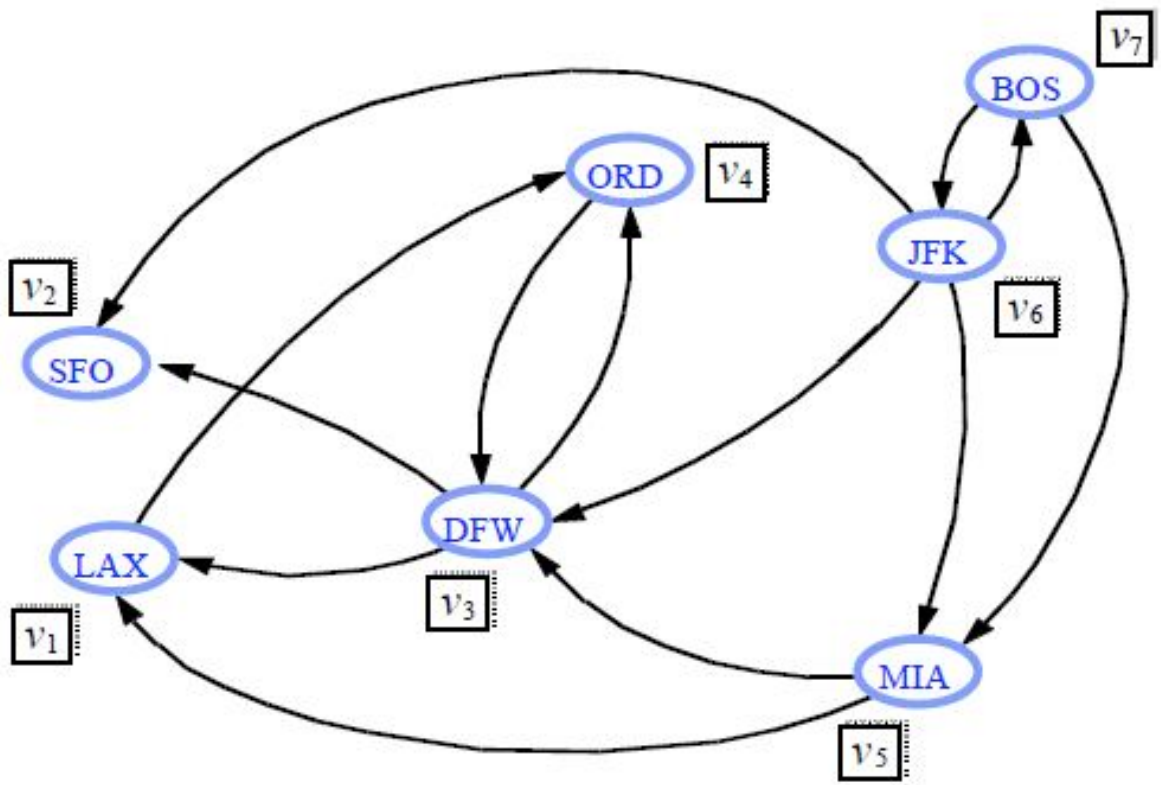


L'algorithme général de Roy-Warshall-Floyd

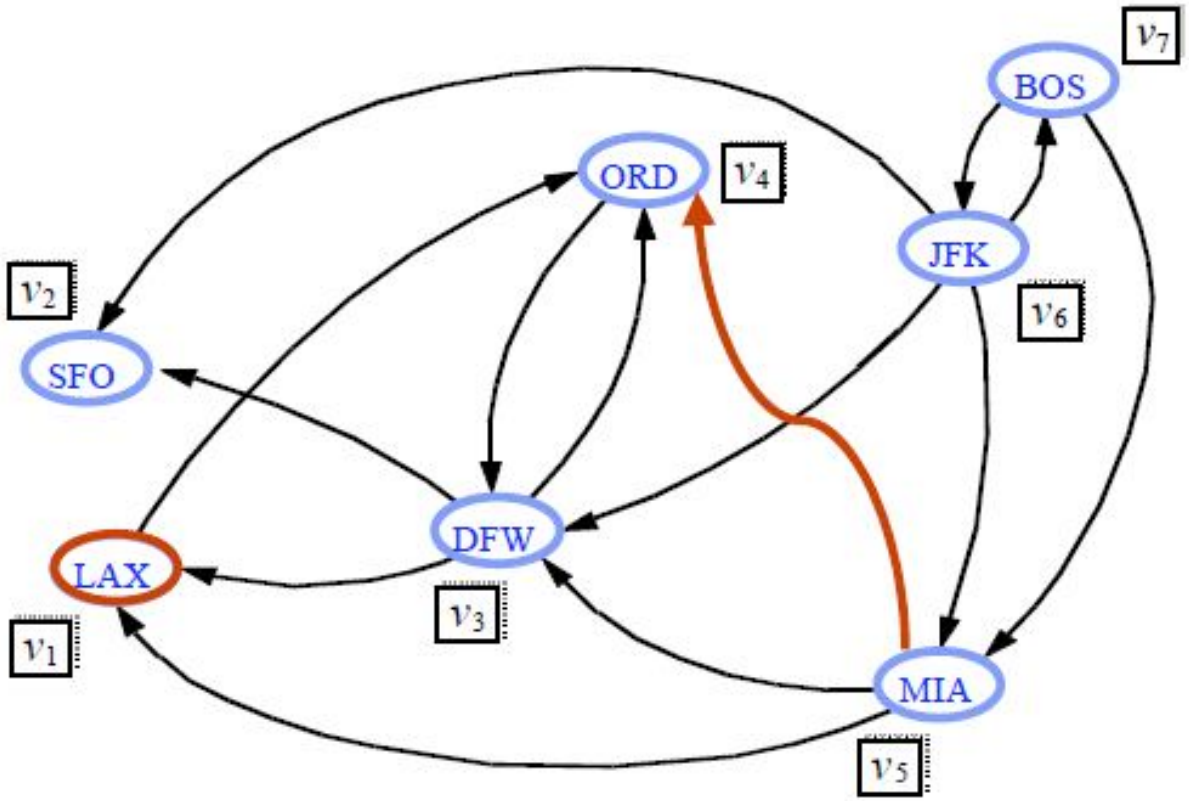
- L'algorithme de Floyd-Warshall numérote les sommets de 1 à n et calcule une séries de graphes orientés $G_0, G_1, \ldots, G_n$ tels que
  1) $G_0 = G$
  2) G_k a une arête dirigée (v_i, v_j) si G a un chemin de v_i à v_j dont les sommets intermédiaires sont dans $\{v_1, v_2, \ldots, v_k\}$
  3) $G_n = G^*$
- Complexité en temps: $O(n^3)$, si on assume que sontAdjacents peut être calculé en $O(1)$

```
Algorithme FloydWarshall(G)
  Entrée graphe orienté G
  Sortie fermeture transitive G* de G G_{k-1}
  i ← 1
  pour tout v ∈ G.sommets()
    numéroter v par v_i
    i ← i + 1
  G_0 ← G
  pour k ← 1 à n faire
    G_k ← G_{k-1}
    pour i ← 1 à n (i ≠ k) faire
      pour j ← 1 à n (j ≠ i, k) faire
        si G_{k-1}.sontAdjacents(v_i, v_k) ∧
           G_{k-1}.sontAdjacents(v_k, v_j)
          si ¬G_k.sontAdjacents(v_i, v_j)
            G_k.insérerArêteDirigée(v_i, v_j, k)
    retourner G_n
```

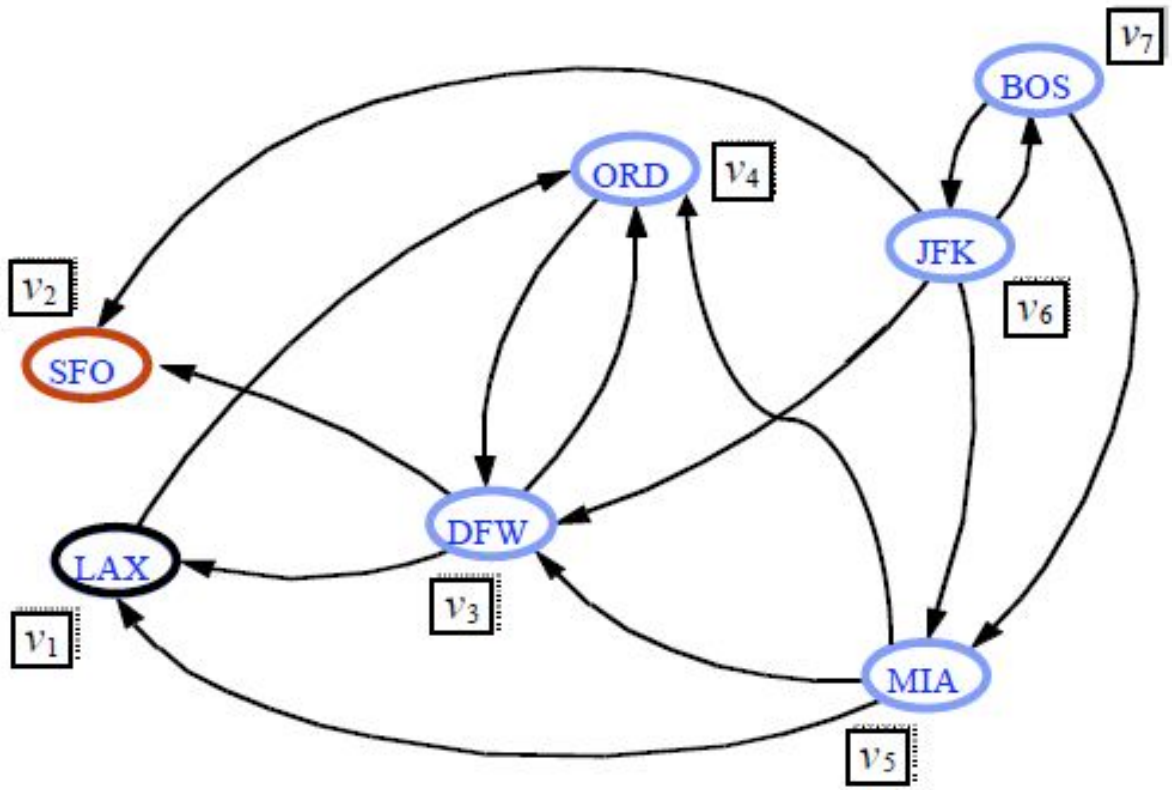
L'algorithme général de Roy-Warshall-Floyd



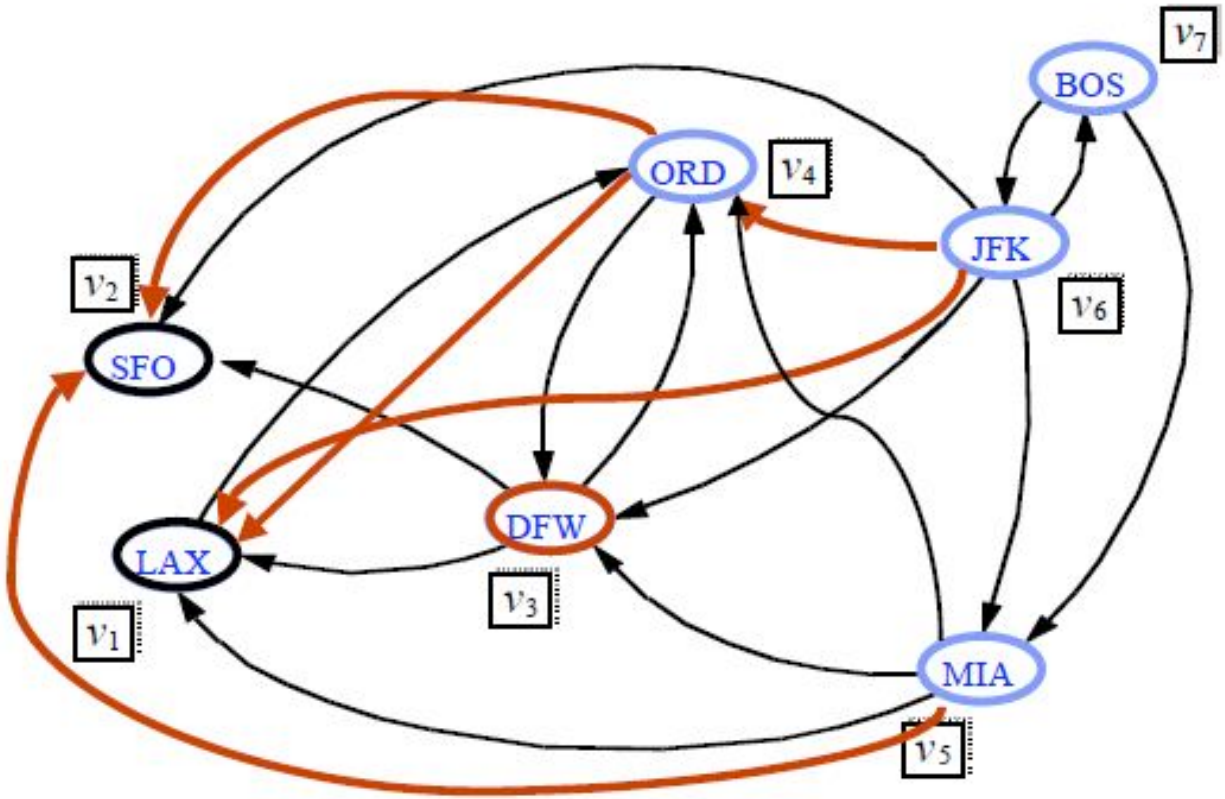
L'algorithme général de Roy-Warshall-Floyd



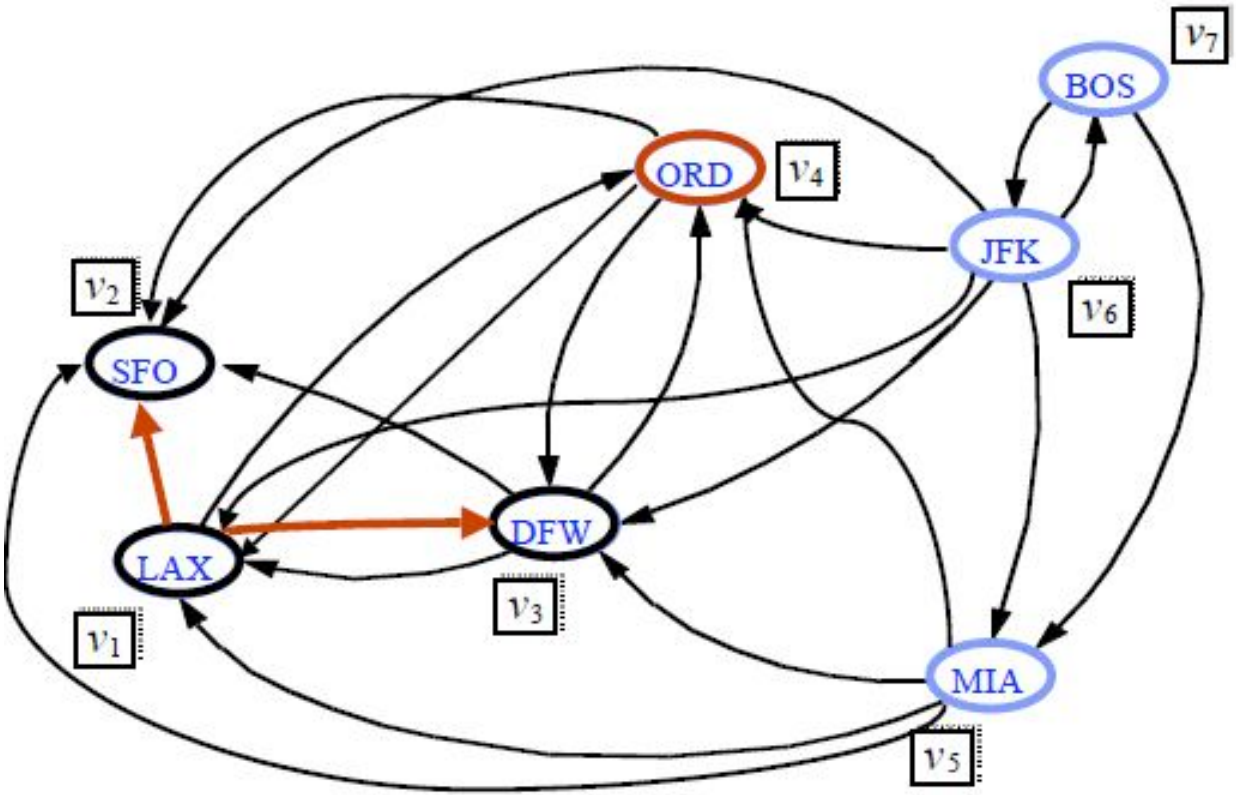
L'algorithme général de Roy-Warshall-Floyd



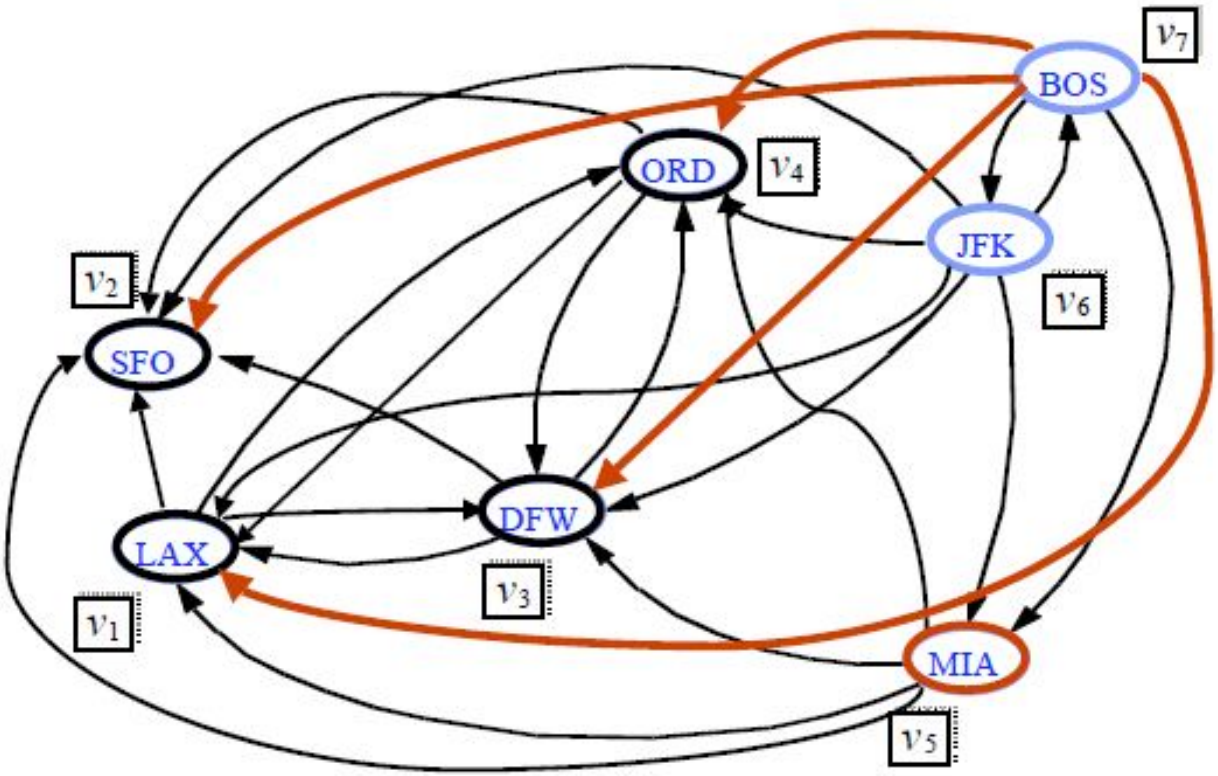
L'algorithme général de Roy-Warshall-Floyd



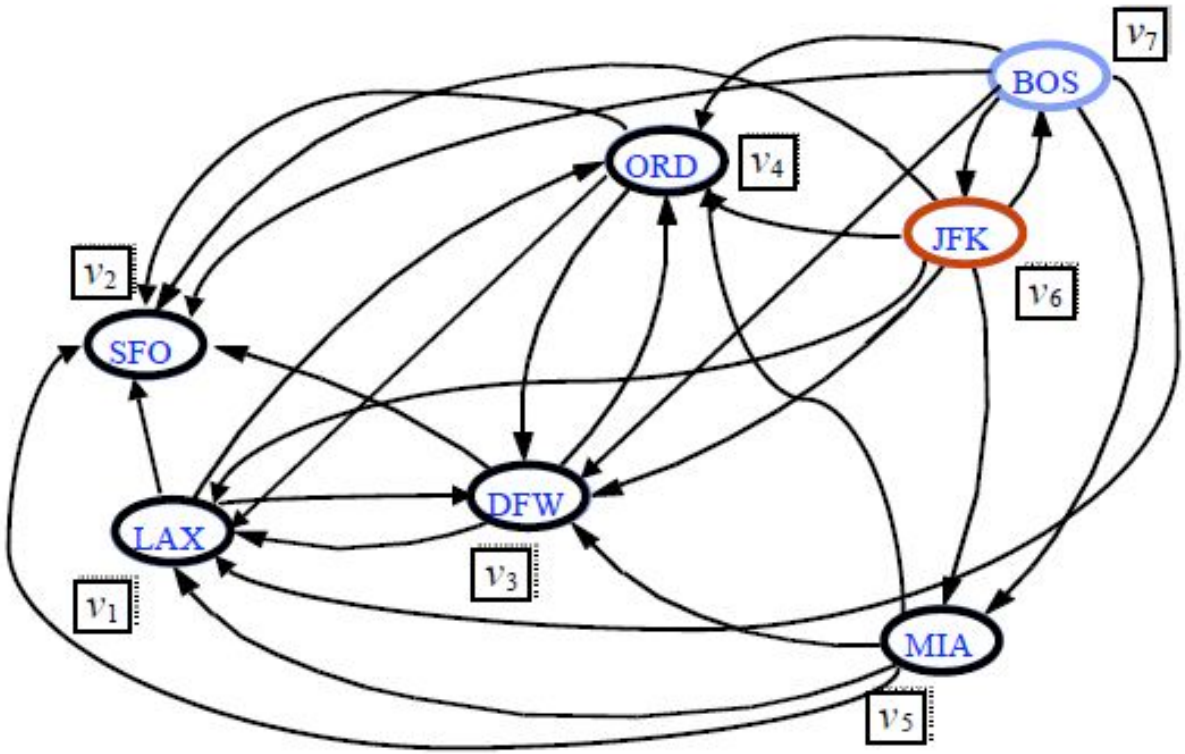
L'algorithme général de Roy-Warshall-Floyd



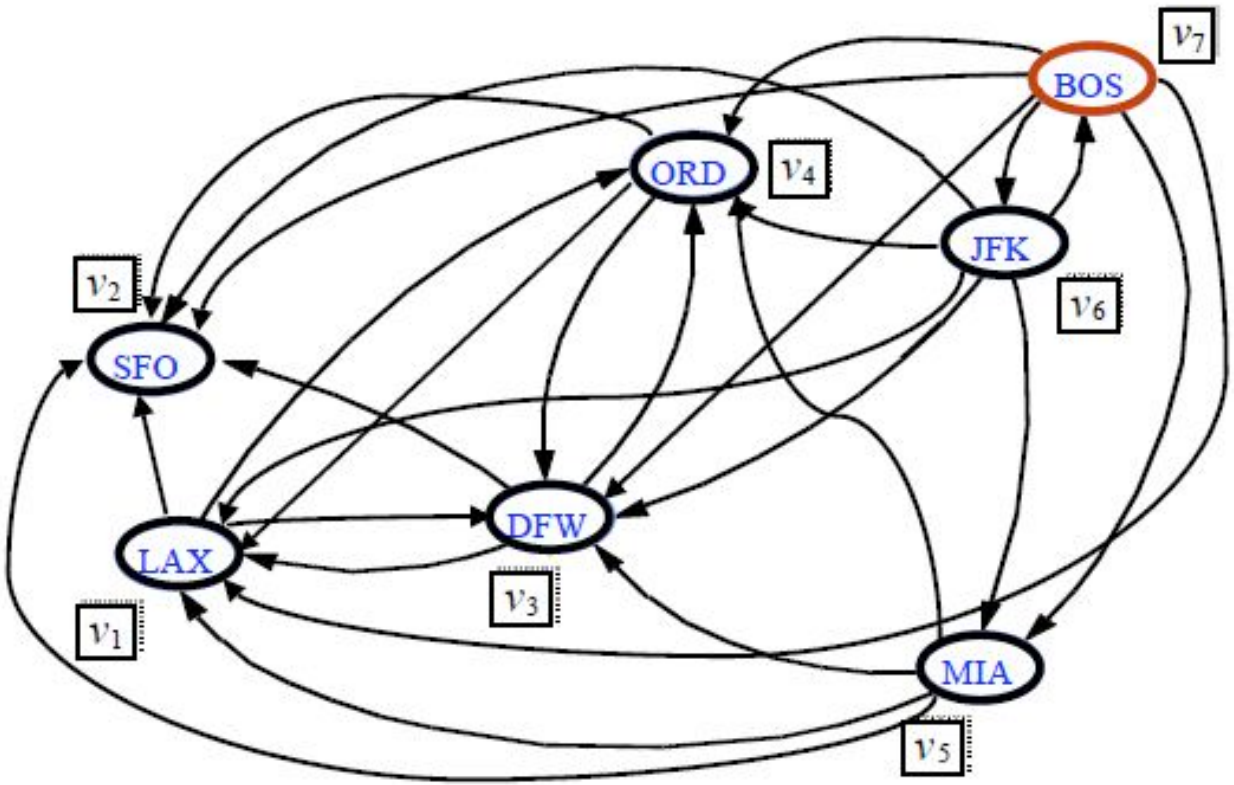
L'algorithme général de Roy-Warshall-Floyd



L'algorithme général de Roy-Warshall-Floyd



L'algorithme général de Roy-Warshall-Floyd



Lecture 15: The Floyd-Warshall Algorithm

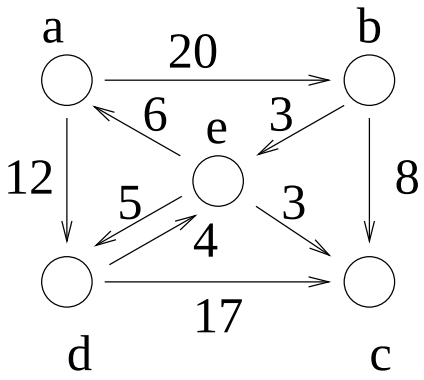
CLRS section 25.2

Outline of this Lecture

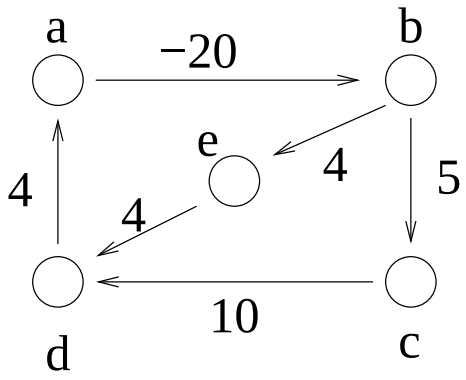
- Recalling the all-pairs shortest path problem.
- Recalling the previous two solutions.
- The Floyd-Warshall Algorithm.

The All-Pairs Shortest Paths Problem

Given a weighted digraph $G = (V, E)$ with a weight function $w : E \to \mathbf{R}$, where R is the set of real numbers, determine the length of the shortest path (i.e., distance) between all pairs of vertices in G. Here we assume that there are no cycle with zero or negative cost.



without negative cost cycle



with negative cost cycle

Solutions Covered in the Previous Lecture

Solution 1: Assume no negative *edges*.

Run Dijkstra's algorithm, n times, once with each vertex as source.

$O(n^3 \log n)$. $O(n^3)$ with more sophisticated data structures.

Solution 2: Assume no negative *cycles*.

Dynamic programming solution, based on a natural decomposition of the problem.

$O(n^4)$. $O(n^3 \log n)$ using " repeated squaring".

This lecture: Assume no negative *cycles*.

develop another dynamic programming algorithm, the *Floyd-Warshall algorithm,* with time complexity $O(n^3)$.

Also illustrates that there can be more than one way of developing a dynamic programming algorithm.

Solution 3: the Input and Output Format

As in the previous dynamic programming algorithm, we assume that the graph is represented by an $n \times n$ matrix with the weights of the edges:

$$w_{ij} = \begin{cases} 0 & \text{if } i = j, \\ w(i,j) & \text{if } i \neq j \text{ and } (i,j) \in E, \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E. \end{cases}$$

Output Format: an $n \times n$ distance $D = [d_{ij}]$ where d_{ij} is the distance from vertex i to j.

Step 1: The Floyd-Warshall Decomposition

Definition: The vertices $v_2, v_3, ..., v_{l-1}$ are called the *intermediate vertices* of the path $p = \langle v_1, v_2, ..., v_l \rangle$.

- Let $d_{ij}^{(k)}$ be the length of the shortest path from i to j such that *all* intermediate vertices on the path (if any) are in set $\{1, 2, \ldots, k\}$.

  $d_{ij}^{(0)}$ is set to be w_{ij}, i.e., no intermediate vertex.
  Let $D^{(k)}$ be the $n \times n$ matrix $[d_{ij}^{(k)}]$.

- Claim: $d_{ij}^{(n)}$ is the distance from i to j. So our aim is to compute $D^{(n)}$.

- Subproblems: compute $D^{(k)}$ for $k = 0, 1, \cdots, n$.

Step 2: Structure of shortest paths

Observation 1:

A shortest path does not contain the same vertex twice.

Proof: A path containing the same vertex twice contains a cycle. Removing cycle gives a shorter path.

Observation 2: For a shortest path from i to j such that any intermediate vertices on the path are chosen from the set $\{1, 2, \ldots, k\}$, there are two possibilities:

1. k is not a vertex on the path,
The shortest such path has length $d_{ij}^{(k-1)}$.

2. k is a vertex on the path.
The shortest such path has length $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

Step 2: Structure of shortest paths

Consider a shortest path from i to j containing the vertex k. It consists of a subpath from i to k and a subpath from k to j.
Each subpath can only contain intermediate vertices in $\{1, \ldots, k-1\}$, and must be as short as possible, namely they have lengths $d_{ik}^{(k-1)}$ and $d_{kj}^{(k-1)}$.

Hence the path has length $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

Combining the two cases we get

$$d_{ij}^{(k)} = \min \left\{ d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right\}.$$

Step 3: the Bottom-up Computation

- Bottom: $D^{(0)} = [w_{ij}]$, the weight matrix.

- Compute $D^{(k)}$ from $D^{(k-1)}$ using

$$d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$$

for $k = 1, ..., n$.

The Floyd-Warshall Algorithm: Version 1

Floyd-Warshall(w, n)

{ for $i = 1$ to n do initialize

  for $j = 1$ to n do

  { $D^0[i,j] = w[i,j]$;

  $pred[i,j] = nil$;

  }

  for $k = 1$ to n do dynamic programming

  for $i = 1$ to n do

  for $j = 1$ to n do

  if $\left(d^{(k-1)}[i,k] + d^{(k-1)}[k,j] < d^{(k-1)}[i,j]\right)$

  $\{d^{(k)}[i,j] = d^{(k-1)}[i,k] + d^{(k-1)}[k,j]$;

  $pred[i,j] = k$;$\}$

  else $d^{(k)}[i,j] = d^{(k-1)}[i,j]$;

  return $d^{(n)}[1..n, 1..n]$;

}

Comments on the Floyd-Warshall Algorithm

- The algorithm's running time is clearly $\Theta(n^3)$.
- The predecessor pointer `pred[i,j]` can be used to extract the final path (see later).
- Problem: the algorithm uses $\Theta(n^3)$ space.
  It is possible to reduce this down to $\Theta(n^2)$ space by keeping only one matrix instead of n.
  Algorithm is on next page. Convince yourself that it works.

The Floyd-Warshall Algorithm: Version 2

Floyd-Warshall(w, n)

```
{ for i = 1 to n do                                   initialize
      for j = 1 to n do
      { d[i,j] = w[i,j];
        pred[i,j] = nil;
      }

  for k = 1 to n do                                   dynamic programming
      for i = 1 to n do
          for j = 1 to n do
              if (d[i,k] + d[k,j] < d[i,j])
                    {d[i,j] = d[i,k] + d[k,j];
                     pred[i,j] = k;}
  return d[1..n, 1..n];
}
```

Extracting the Shortest Paths

The predecessor pointers `pred[i,j]` can be used to extract the final path. The idea is as follows.

Whenever we discover that the shortest path from i to j passes through an intermediate vertex k, we set $pred[i,j] = k$.

If the shortest path does not pass through any intermediate vertex, then $pred[i,j] = nil$.

To find the shortest path from i to j, we consult $pred[i,j]$. If it is nil, then the shortest path is just the edge (i,j). Otherwise, we recursively compute the shortest path from i to $pred[i,j]$ and the shortest path from $pred[i,j]$ to j.

The Algorithm for Extracting the Shortest Paths

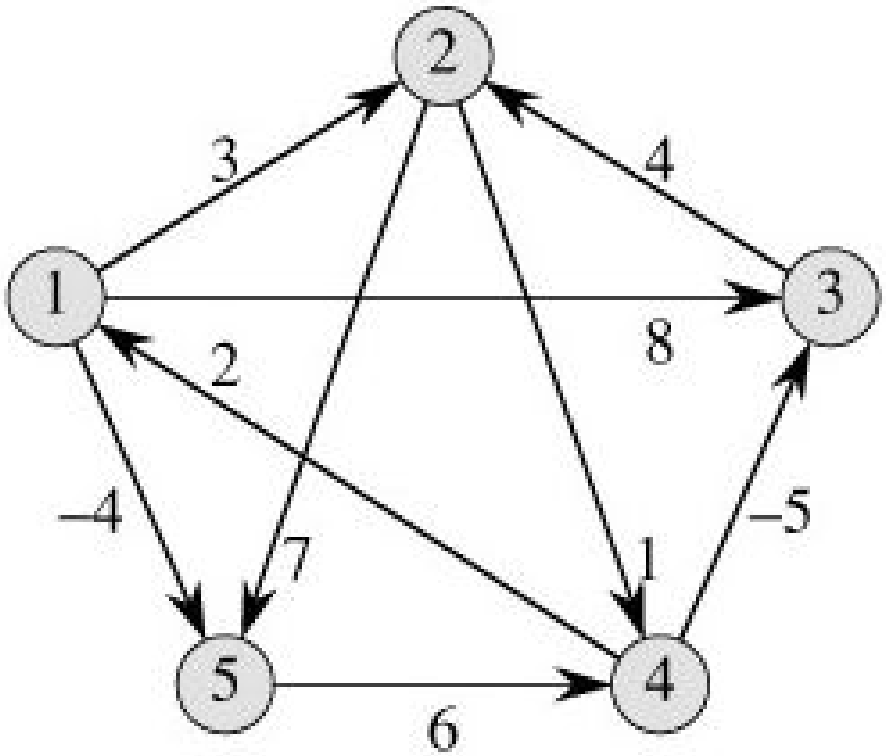
```
Path(i, j)
{
    if (pred[i, j] = nil)    single edge
        output (i, j);
    else          compute the two parts of the path
    {
        Path(i, pred[i, j]);
        Path(pred[i, j], j);
    }
}
```

Example of Extracting the Shortest Paths

Find the shortest path from vertex 2 to vertex 3.

2..3	Path$(2,3)$	$pred[2,3] = 4$	
2..4..3	Path$(2,4)$	$pred[2,4] = 5$	
2..5..4..3	Path$(2,5)$	$pred[2,5] = nil$	$Output$(2,5)
25..4..3	Path$(5,4)$	$pred[5,4] = nil$	$Output$(5,4)
254..3	Path$(4,3)$	$pred[4,3] = 6$	
254..6..3	Path$(4,6)$	$pred[4,6] = nil$	$Output$(4,6)
2546..3	Path$(6,3)$	$pred[6,3] = nil$	$Output$(6,3)
25463			

L'algorithme général de Roy-Warshall-Floyd



L'algorithme général de Roy-Warshall-Floyd

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

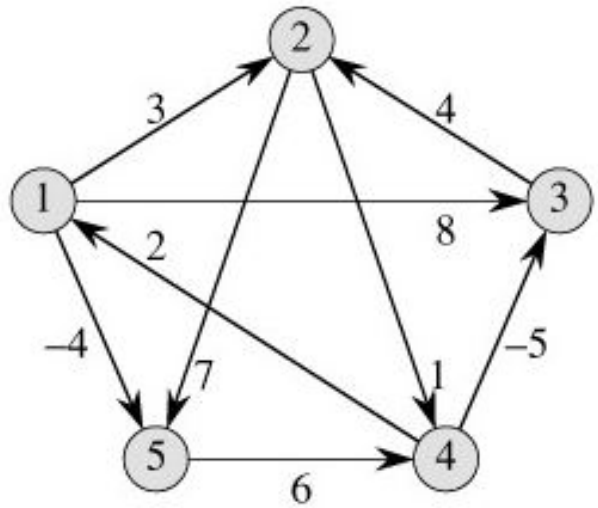
$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$



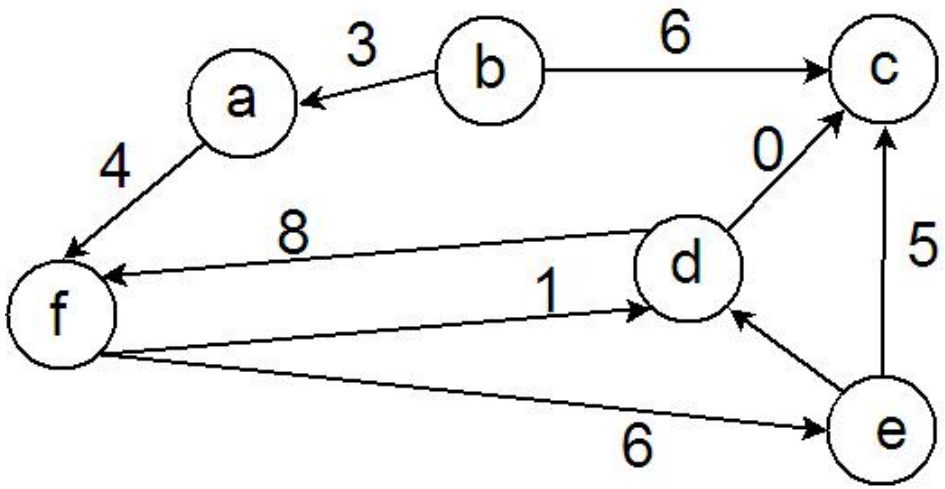
L'algorithme de Roy-Warshall-Floyd

Complexité

L'algorithme de Roy-Warshall-Floyd calcule en temps $O(N^3)$ (avec N le nombre de sommets du graphe) la longueur des plus courts chemins entre tout couple de sommets du graphe.

L'algorithme général de Roy-Warshall-Floyd

Appliquer ce magnifique algorithme au graphe suivant, en ayant soin de détailler les étapes intermédiaires :



L'algorithme de Bellman-Ford

Problème de plus court chemin

- **Entrée** : un graphe $G = (V, E)$, des longueurs $l(u, v)$ pour tout $(u, v) \in E$, deux sommets $s, t \in V$.
- **Sortie** : $\delta(s, t)$ la longueur du plus court chemin entre s et t.

Plusieurs cas

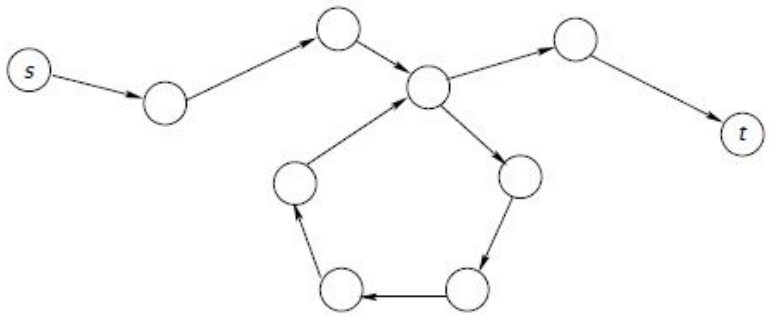
- **Arcs de longueurs positives** : l'algorithme de Dijkstra résout le problème en $|E|log|V|$.
- **Arcs de longueurs quelconques**
  - **Cycle de longueur négative** : pas de plus court chemin
  - **Pas de cycle de longueur négative** : on peut résoudre le problème en utilisant la programmation dynamique :

algorithme de Bellman-Ford

L'algorithme de Bellman-Ford

Idée de base

Si le graphe G ne contient aucun cycle de longueur négative, alors il existe un plus court chemin élémentaire entre s et t.



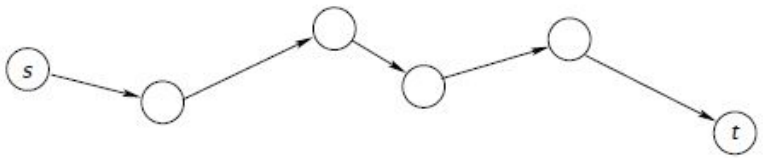
Notation

$OPT(i, v)$ la longueur minimale d'un chemin de v à t contenant au maximum i arcs (objectif : calculer $OPT(n-1, s)$).

L'algorithme de Bellman-Ford

Idée de base

Si le graphe G ne contient aucun cycle de longueur négative, alors il existe un plus court chemin élémentaire entre s et t.



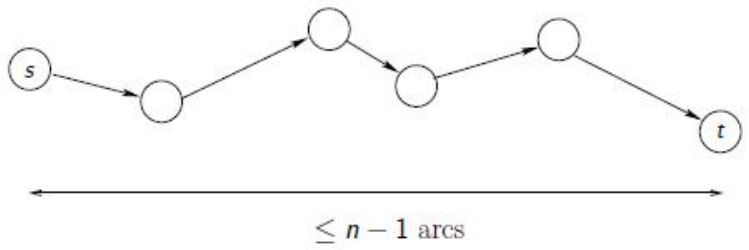
Notation

$OPT(i, v)$ la longueur minimale d'un chemin de v à t contenant au maximum i arcs (objectif : calculer $OPT(n-1, s)$).

L'algorithme de Bellman-Ford

Idée de base

Si le graphe G ne contient aucun cycle de longueur négative, alors il existe un plus court chemin élémentaire entre s et t.



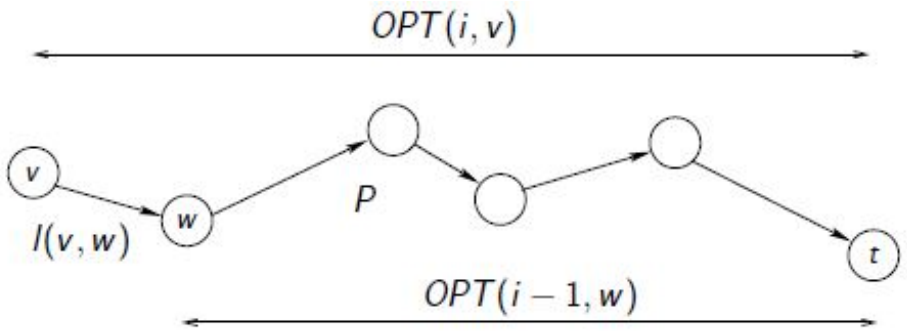
Notation

$OPT(i, v)$ la longueur minimale d'un chemin de v à t contenant au maximum i arcs (objectif : calculer $OPT(n-1, s)$).

L'algorithme de Bellman-Ford

Formule de récurrence

Soit P un chemin optimal pour le sous-problème $OPT(i, v)$



Formule

Deux cas :

1. si P utilise au plus $i-1$ arcs, $OPT(i, v) = OPT(i-1, v)$;
2. si P utilise exactement i arcs, $OPT(i, v) = l(v, w) + OPT(i-1, w)$.

On déduit la formule de récurrence suivante, pour tout $i > 0, v \in V - t$,

$$OPT(i, v) = min(OPT(i-1, v), min_{w \in V}(l(v, w) + OPT(i-1, w))).$$

L'algorithme de Bellman-Ford

En utilisant, la formule de récurrence précédente, on obtient l'algorithme de programmation dynamique suivant pour calculer $OPT(n-1, s)$.

Algorithme de Bellman-Ford (V1)

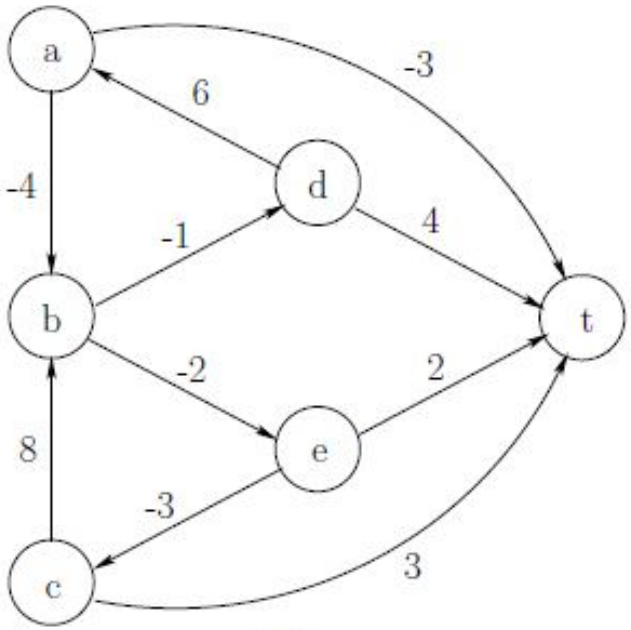
Bellman-Ford-V1(G, s, t)
Fixer $M[0, v] = +\infty$ pour tout $v \in V - t$ et $M[0, t] = 0$
Pour $i = 1, ..., n-1$ faire
Pour $v \in V$ faire
$M[i, v] = min(M[i-1, v], min_{w \in V}(l(v, w) + M[i-1, w]))$
Fin-faire
Fin-faire
Renvoyer $M[n-1, s]$

Complexité

Chacune des n^2 entrées de la table M est calculée en temps $\mathcal{O}(n)$, la complexité de cet algorithme est donc $\mathcal{O}(n^3)$.

L'algorithme de Bellman-Ford

Exemple



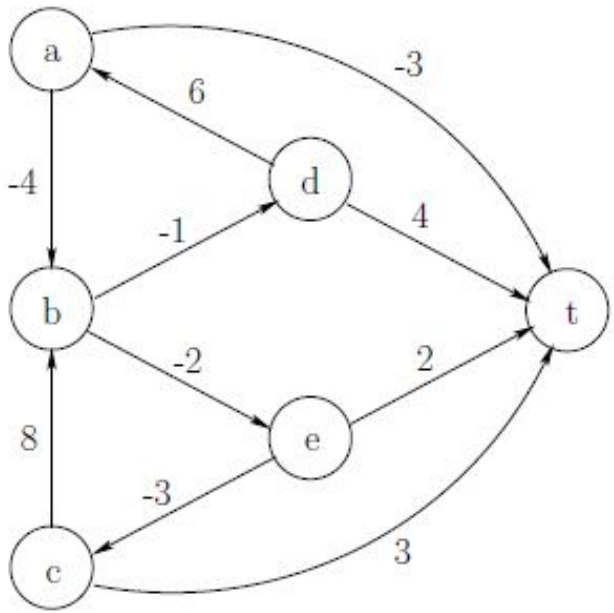
(a)

	0	1	2	3	4	5
t	0					
a	∞					
b	∞					
c	∞					
d	∞					
e	∞					

(b)

L'algorithme de Bellman-Ford

Exemple



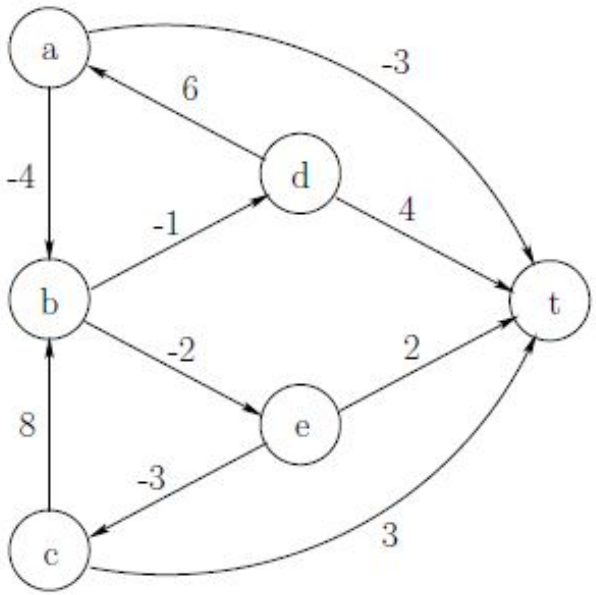
(a)

	0	1	2	3	4	5
t	0	0				
a	∞	−3				
b	∞	∞				
c	∞	3				
d	∞	4				
e	∞	2				

(b)

L'algorithme de Bellman-Ford

Exemple



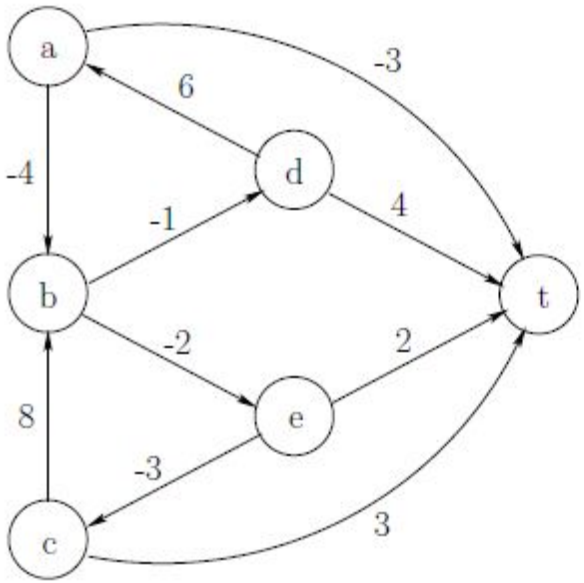
(a)

	0	1	2	3	4	5
t	0	0	0			
a	∞	−3	−3			
b	∞	∞	0			
c	∞	3	3			
d	∞	4	3			
e	∞	2	0			

(b)

L'algorithme de Bellman-Ford

Exemple



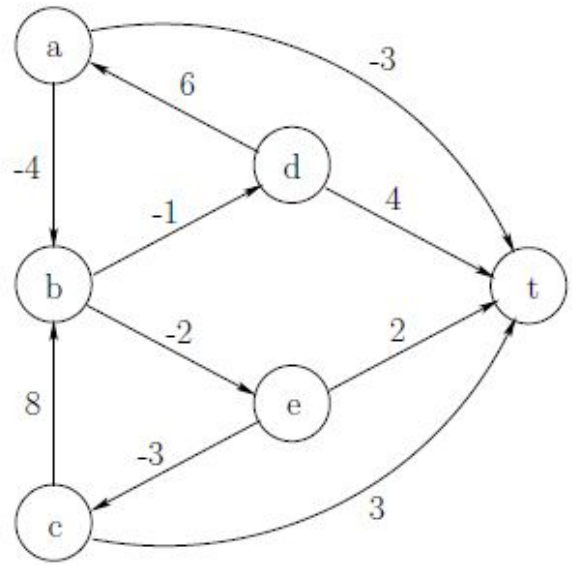
(a)

	0	1	2	3	4	5
t	0	0	0	0		
a	∞	−3	−3	−4		
b	∞	∞	0	−2		
c	∞	3	3	3		
d	∞	4	3	3		
e	∞	2	0	0		

(b)

L'algorithme de Bellman-Ford

Exemple



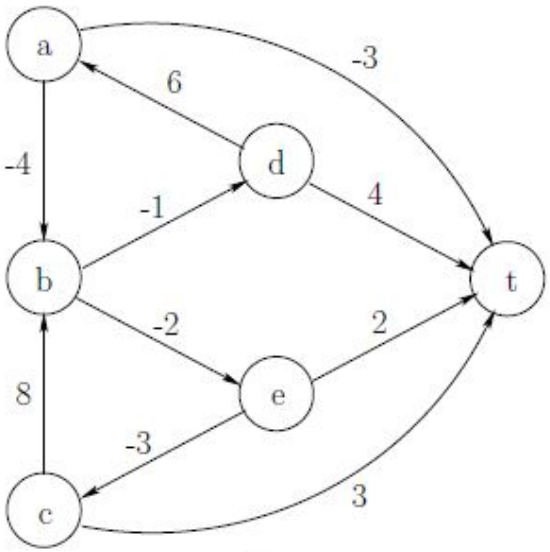
(a)

	0	1	2	3	4	5
t	0	0	0	0	0	
a	∞	-3	-3	-4	-6	
b	∞	∞	0	-2	-2	
c	∞	3	3	3	3	
d	∞	4	3	3	2	
e	∞	2	0	0	0	

(b)

L'algorithme de Bellman-Ford

Exemple



(a)

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	-3	-3	-4	-6	-6
b	∞	∞	0	-2	-2	-2
c	∞	3	3	3	3	3
d	∞	4	3	3	2	0
e	∞	2	0	0	0	0

(b)

L'algorithme de Bellman-Ford

Amélioration du temps de calcul

L'algorithme de Bellman-Ford peut-être implémenté en $\mathcal{O}(mn)$ avec $n := |V|$ et $m = |E|$.

Justification

- En effet, lors de l'évaluation de l'expression

$$min_{w \in V}(l(v, w) + M[i-1, w])$$

il suffit de considérer les sommets w qui sont des voisins de v (adjacents).

- Le coût d'évaluation de chacune des entrées $M[i, v]$ est donc en $\mathcal{O}(n_v)$, où n_v est le nombre de voisins de v.
- Le coût global de l'algorithme est $\mathcal{O}(n \sum_{v \in V} n_v)$, i.e. $\mathcal{O}(nm)$.

L'algorithme de Bellman-Ford

L'amélioration conduit à cette deuxième version de l'algorithme :

Algorithme de Bellman-Ford (V2)

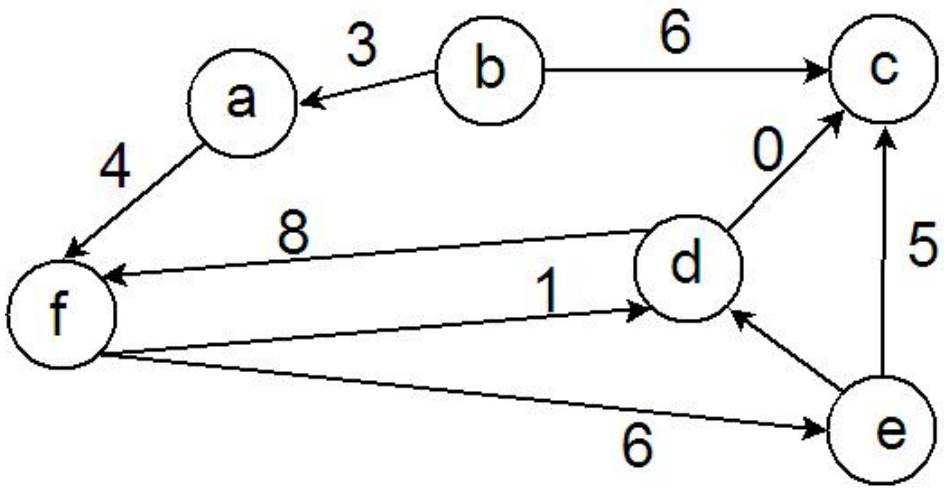
Bellman-Ford-V2(G, s, t)
Fixer $M[0, v] = +\infty$ pour tout $v \in V - t$ et $M[0, t] = 0$
Pour $i = 1, ..., n - 1$ faire
Pour $v \in V$ faire
$M[i, v] = min(M[i - 1, v], min_{w \in \mathbf{Adj(V)}}(l(v, w) + M[i - 1, w]))$
Fin-faire
Fin-faire
Renvoyer $M[n - 1, s]$

Complexité

Chacune des entrées de la table $M[i, v]$ est calculée en temps $\mathcal{O}(n_v)$, la complexité de cet algorithme est donc $\mathcal{O}(n \sum_{v \in V} n_v) = \mathcal{O}(nm)$.

L'algorithme de Bellman-Ford

Appliquer ce non moins magnifique algorithme au graphe suivant, en ayant soin de détailler les étapes intermédiaires :



Outline

Flots et Coupe

Réseau

- Soit $G = (S, A)$ un graphe orienté.
- Chaque arc est valué par une valeur correspondant à la capacité du lien associé à cet arc.
- Soit la fonction $c : A \rightarrow R^+$ qui associe à chaque arc une valeur réelle positive de capacité.
- Soient deux sommets particuliers : un sommet source s et un sommet puits p, tels que $\forall v \in S$ il existe un chemin qui va de s à p et qui passe par v.
- Un réseau est défini par la donnée du quadruplet $R = (G, c, s, p)$.

Flots et Coupe

Réseau

Un flot est une fonction $f : A \rightarrow R^+$ telle que :

1. $f(a,b) \leq c(a,b), \forall (a,b) \in A$
2. $\sum_{j \in Voisins(s)} f(s,j) - \sum_{j \in Voisins(s)} f(j,s) = v$
3. $\sum_{j \in Voisins(p)} f(p,j) - \sum_{j \in Voisins(p)} f(j,p) = -v$
4. $\sum_{j \in Voisins(i)} f(i,j) - \sum_{j \in Voisins(i)} f(j,i) = 0, \forall i \in S \setminus \{s,p\}$

- v représente la valeur du flot.
- Equation (4) : loi de Kirchhoff.

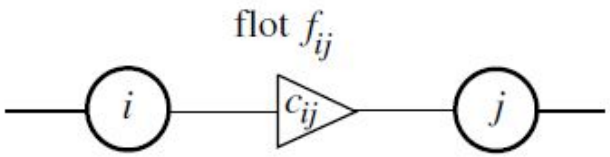
Flots et Coupe

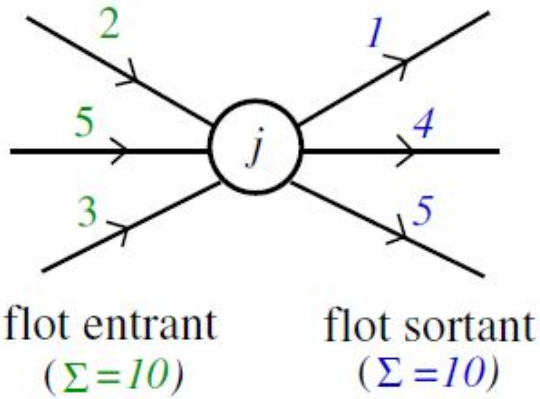
Flot réalisable

- Le flot est dit réalisable si pour toute arête $(i, j) \in A$:

$$0 \leq f_{ij} \leq c_{ij}$$

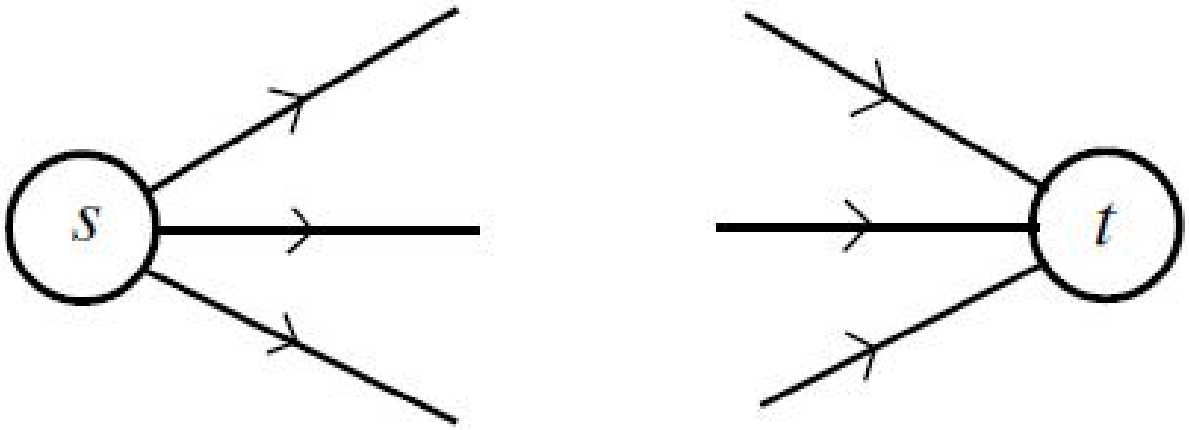
- La quantité $v = \sum_{i \in P(p)} f(i, p)$ est la valeur du flot de s à p.





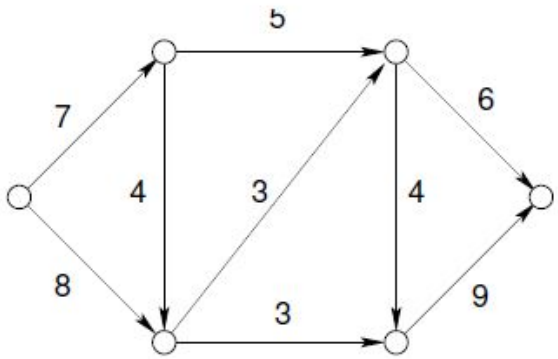
Flots et Coupe

- $S(i)$: ensemble des sommets j successeurs du sommet i i.e. tq l'arête (i, j) existe dans le graphe.
- $P(i)$: ensemble des sommets j prédécesseurs du sommet i i.e. tq l'arête (j, i) existe dans le graphe.
- Une source s (resp. un puits p) est un sommet ne possédant pas de prédécesseur (resp. de successeur).

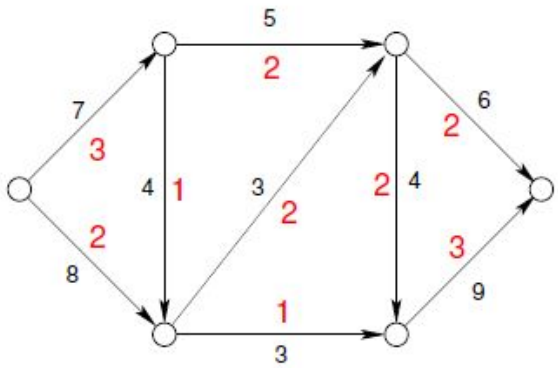


Flots et Coupe

Un réseau

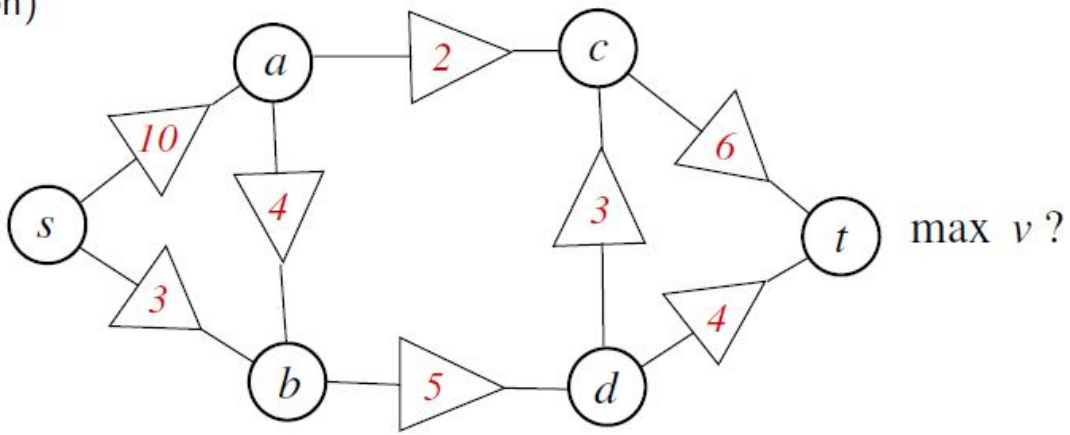


un flot réalisable pour ce réseau



Flots

On veut par ex. trouver le trafic maximal entre deux villes d'un réseau routier dont on connait la capacité (nb de voiture par heure sur chaque tronçon)



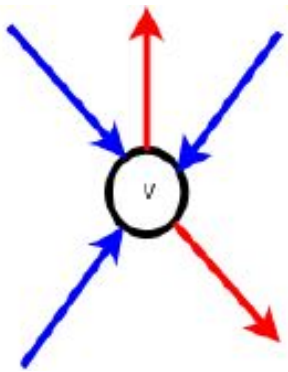
Problème de flot maximal

Etant donné un graphe valué possédant une seule source et un seul puits , trouver un flot réalisable maximal (i.e. dont la valeur est maximale).

Flots : Autre vue

Soit $G = (V, A)$ un graphe orienté. Pour tout sommet $v \in V$, on note:

- $\delta^+(v)$:l'ensemble des arcs sortant en v,
- $\delta^-(v)$:l'ensemble des arcs entrant en v,



Nous considérons deux noeuds spéciaux : un noeud source s et un noeud puits t. Une fonction $f : A \to \mathcal{R}$ est appelée un $s - t$ flot si :

1. $f(a) \geq 0, \forall a \in A$
2. $\sum_{a \in \delta^+(v)} f(a) = \sum_{a \in \delta^-(v)} f(a), \forall v \in V \setminus \{s, t\}$

Flots : Autre vue

- La condition 2 est appelée Loi de conservation de flot : pour tout noeud $v \neq s, t$ le flot entrant en v égale le flot sortant en v.
- La quantité

$$f_0 = \sum_{a \in \delta^+(s)} f(a) = \sum_{a \in \delta^-(s)} f(a)$$

est appelée la valeur du $s - t$ flot f .

Elle est aussi égale à:

$$\sum_{a \in \delta^-(t)} f(a) = \sum_{a \in \delta^+(t)} f(a)$$

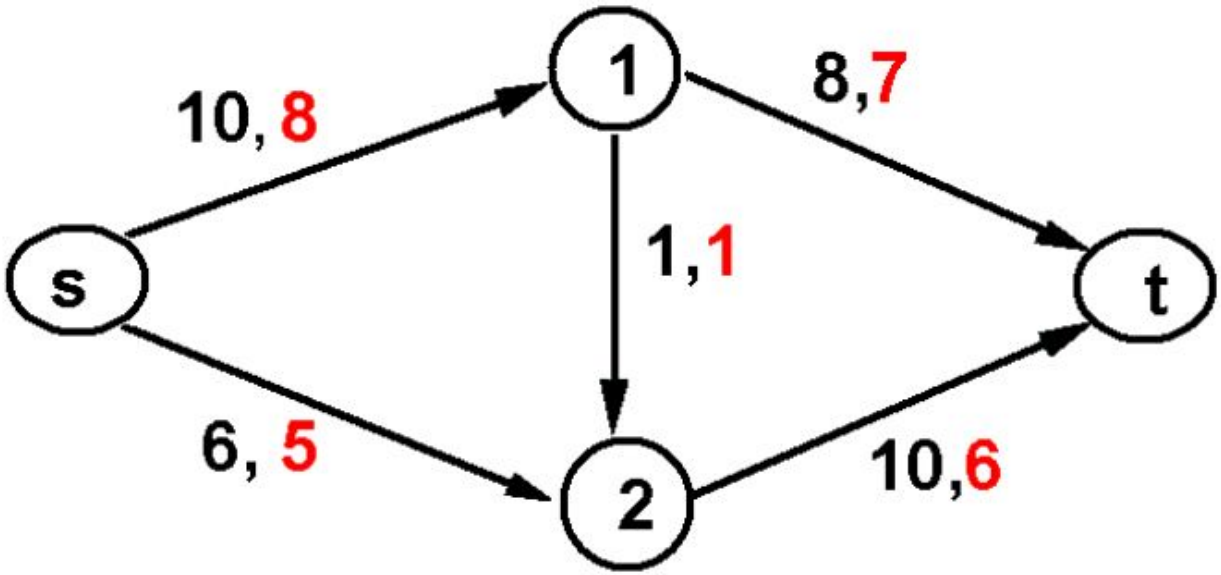
Formulation

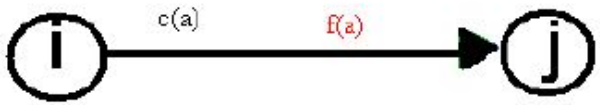
- Soit une fonction $c : A \to \mathcal{R}^+$ qui associe à chaque arc $a \in A$ une capacité $c(a)$. Le flot f est compatible avec c si

$$f(a) \leq c(a) \text{ pour tout arc } a \in A$$

  $c(a)$ indique la limite supérieure du flot admissible sur l'arc a.
- Le problème du flot maximum se formule comme suit
  - Etant donnés un graphe $G = (V, A)$, $s, t \in V$ et une fonction capacité $c : A \to \mathcal{R}^+$,
  - trouver un $s - t$ flot f compatible avec c et maximisant la valeur de f .

Formulation





Flots et Coupe

Coupe

Soit $G = (S, A)$, et $R = (G, c, s, p)$, une coupe est une partition de S en deux ensembles V et $\overline{V}$, tels que :

- $V \cup \overline{V} = S$
- $s \in V$ et $p \in \overline{V}$

la capacité de la coupe est égale à la somme des capacités des arcs qui font la coupe.

$$C(V, \overline{V}) = \sum_{i \in V, j \in \overline{V}} c_{ij}$$

Flots et Coupe

Soit $G = (V, A)$ un graphe orienté. Pour tout $W \subset V$, on note

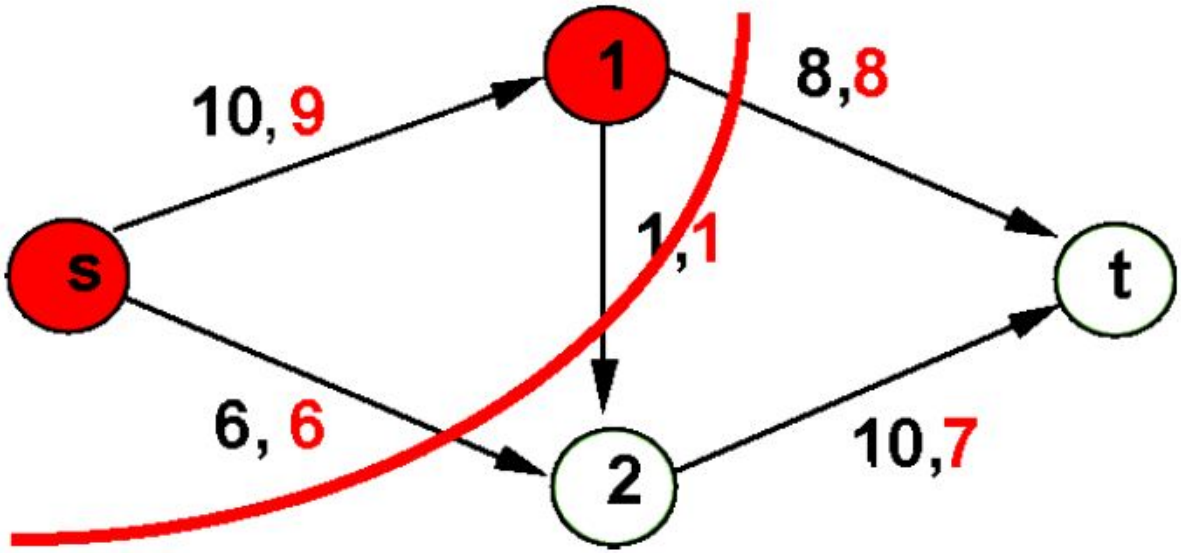
- $\delta^+(W) = \{(i,j) \in A, i \in W, j \notin W\}$, l'ensemble des arcs sortant de W.
- $\delta^-(W) = \{(i,j) \in A, i \notin W, j \notin W\}$, l'ensemble des arcs entrant en W.

Soit W un sous-ensemble de V

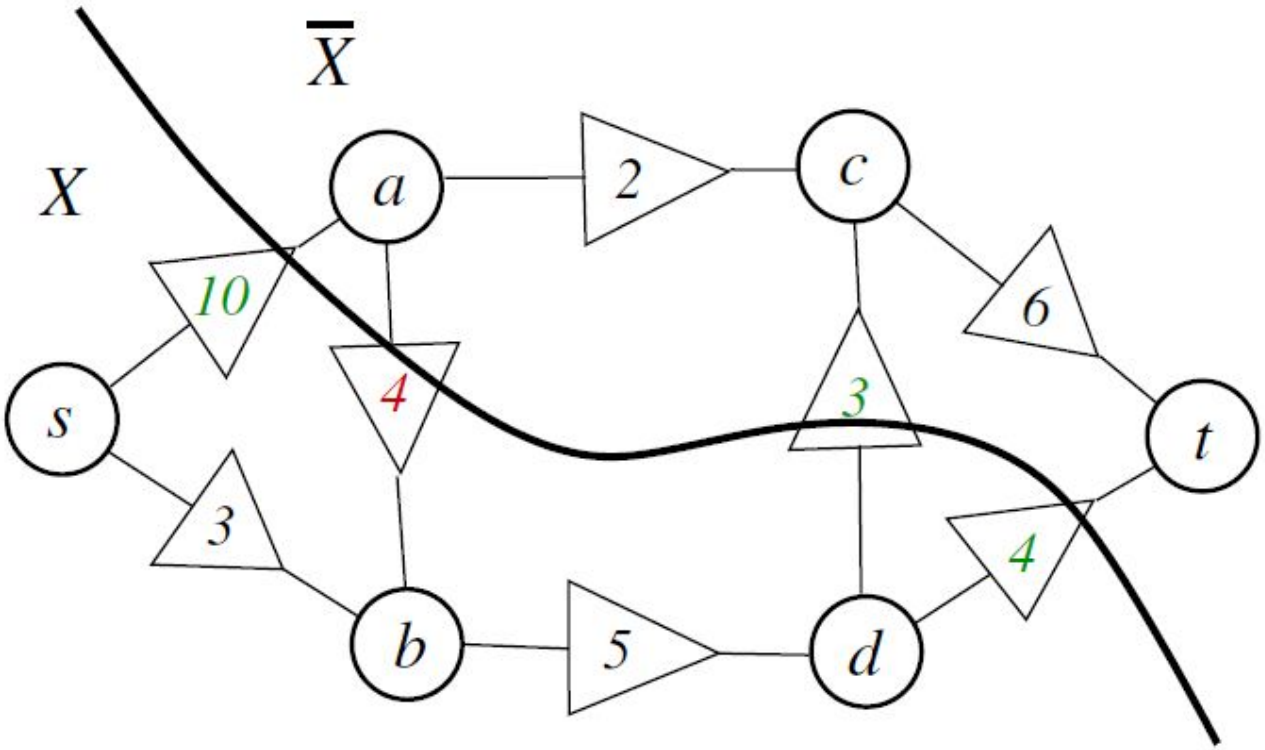
- si $s \in W, t \notin W$, on dit que $\delta^+(W)$ est une $s-t$ coupe,
- La capacité de $\delta^+(W)$ est la somme des capacités des arcs de $\delta^+(W)$

$$c(\delta^+(W)) = \sum_{a \in \delta^+(W)} c(a)$$

Flots et Coupe

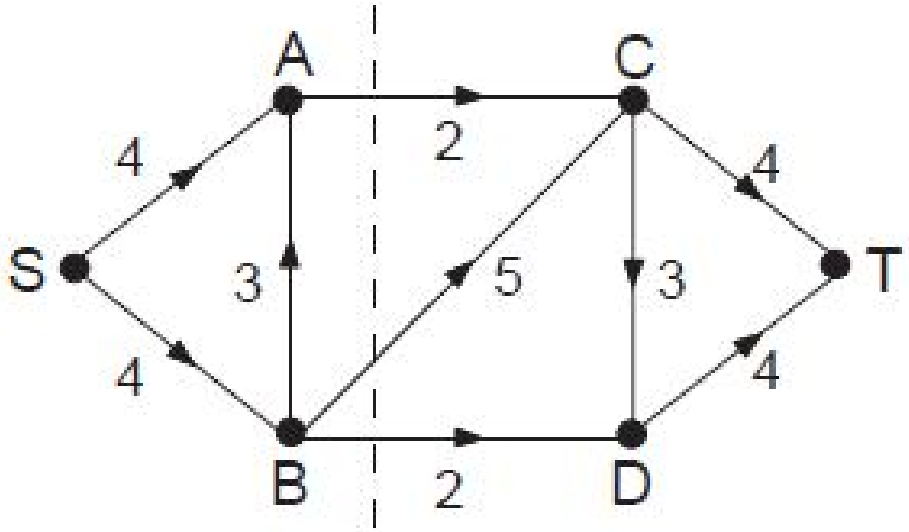


Flots et Coupe



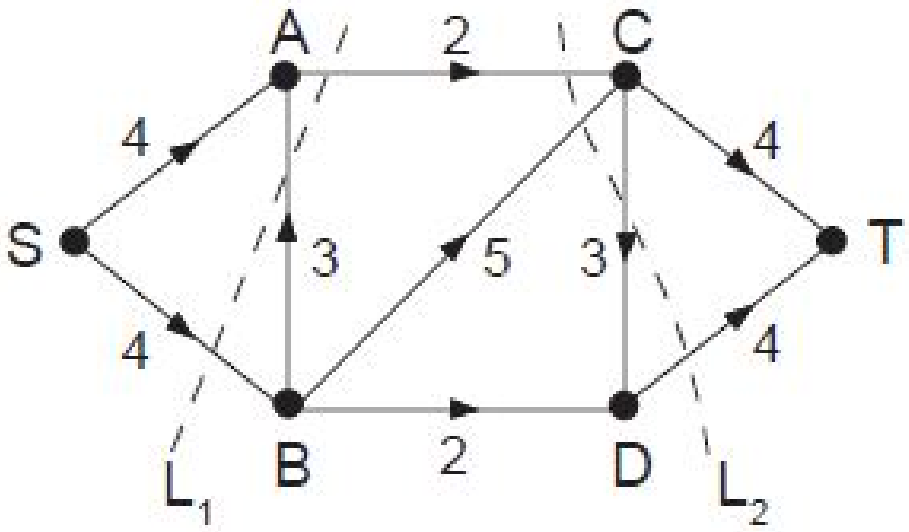
Capacité de la coupe $c(X, \overline{X}) = 10 + 3 + 4 = 17$.

Flots et Coupe



Capacité de la coupe : $2 + 5 + 2 = 9$

Flots et Coupe



- Capacité de la coupe L_1: $2 + 0 + 4 = 6$ car le flot en BA n'est pas dans le bon sens !
- Capacité de la coupe L_2: $2 + 5 + 0 + 4 = 11$.

Flots et Coupe

Proposition

Pour tout $s-t$ flot f et toute $s-t$ coupe $\delta^+(W)$, on a

$$f \leq c(\delta^+(W)).$$

La valeur maximum d'un $s-t$ flot compatible avec la capacité c n'excède jamais la capacité d'une $s-t$ coupe.

Flots et Coupe

Preuve

Soient une $s-t$ coupe $\delta^+(W)$ et f un flot dans un graphe $G=(V,A)$. On a

$$f = \sum_{v \in W} \left[\sum_{a \in \delta^+(v)} f(a) - \sum_{a \in \delta^-(v)} f(a) \right].$$

$$f = \sum_{a \in \delta^+(W)} f(a) - \sum_{a \in \delta^-(W)} f(a).$$

$f(a) \leq c(a)$ pour tout $a \in \delta^+(W)$ et $f(a) \geq 0$ pour tout $a \in \delta^-(W)$

$$\Rightarrow f \leq \sum_{a \in \delta^+(W)} c(a)$$

Flots et Coupe

Proposition

Si un $s-t$ flot f et une $s-t$ coupe $\delta^+(W)$, sont tels que :

$$f = c(\delta^+(W)).$$

alors f est un $s-t$ flot maximum et $\delta^+(W)$ une coupe de capacité minimale.

Proposition

Une condition nécessaire et suffisante pour que le problème de flot maximum ait une solution de valeur finie est qu'il n'existe pas de chemin de capacité infinie entre s et t.

Flots et Coupe

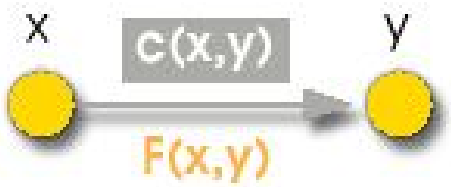
Graphe d'écart

Le concept de **graphe d'écart** (residual graph) est souvent utilisé dans les raisonnements sur les flots.
Pour un $s-t$ flot donné f de $G=(V,A)$, le graphe d'écart $G^e(f)=(V,A(f))$ décrit les possibilités d'augmentation de flot. Il a les mêmes sommets que G et ses arcs sont définis de la manière suivante :

- tout arc (i,j) (arc forward) non saturé de A ($f_{ij} < c_{ij}$) est reporté dans $A(f)$,
- pour tout arc (i,j) de flot non nul dans A ($f_{ij} > 0$), on crée un arc (j,i) (arc backward) dans $A(f)$.

Flots et Coupe

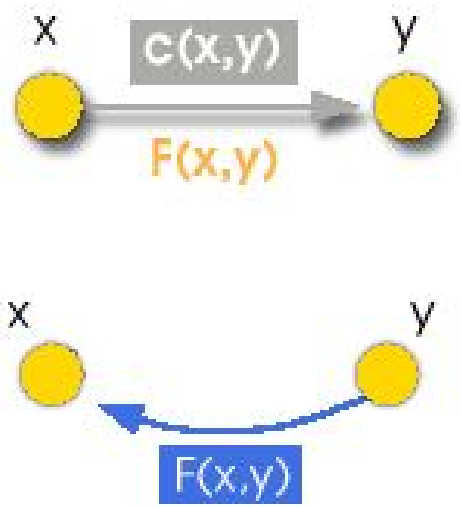
A un arc (x, y) du réseau N est associé dans le réseau résiduel l'arc forward noté $(x, y)_F$ de capacité $c(x, y) - F(x, y)$



La capacité de l'arc forward traduit que l'on peut encore augmenter le flot F sur (x, y), d'au plus $c(x, y) - F(x, y)$ qui correspond à la saturation de l'arc. Un arc forward "existe" (il est de capacité non nulle) donc dans le réseau résiduel ssi il n'est pas saturé dans N.

Flots et Coupe

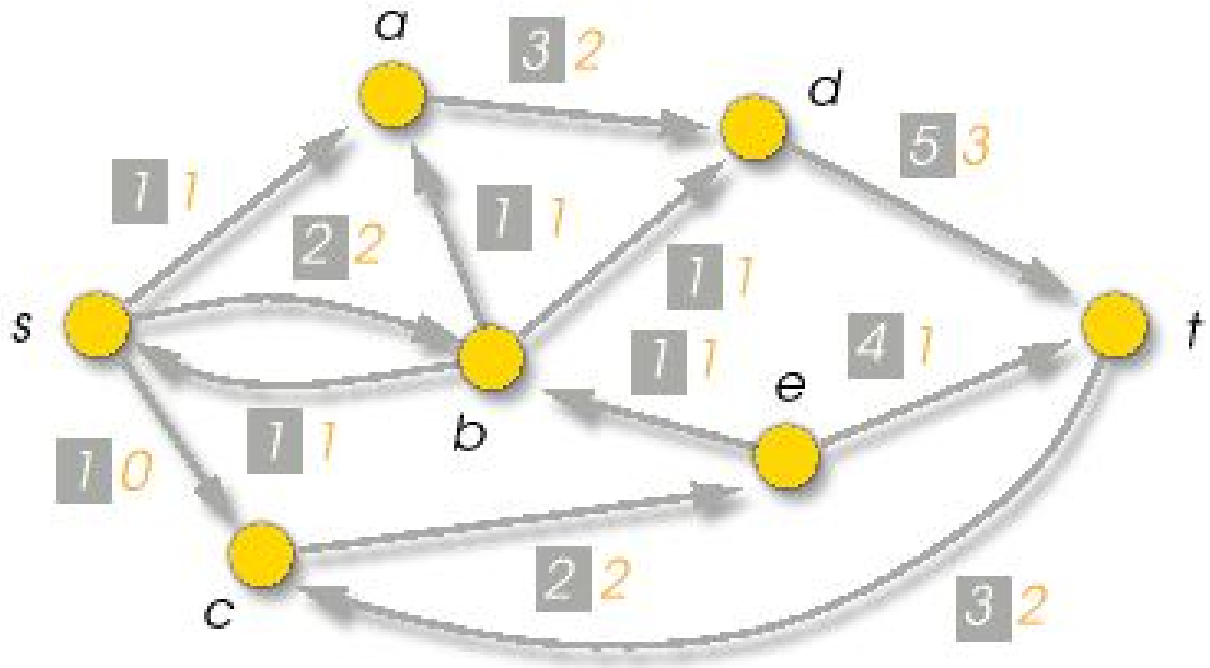
A un arc (x, y) du réseau N est associé dans le réseau résiduel l'arc backward noté $(y, x)_B$ de capacité $F(x, y)$



La capacité de l'arc backward traduit que l'on peut diminuer la valeur du flot F allant de x à y, d'au plus $F(x, y)$ qui correspond à annuler le flot. Un arc backward "existe" (il est de capacité non nulle) donc dans le réseau résiduel ssi le flot F n'est pas nul sur l'arc (x, y).

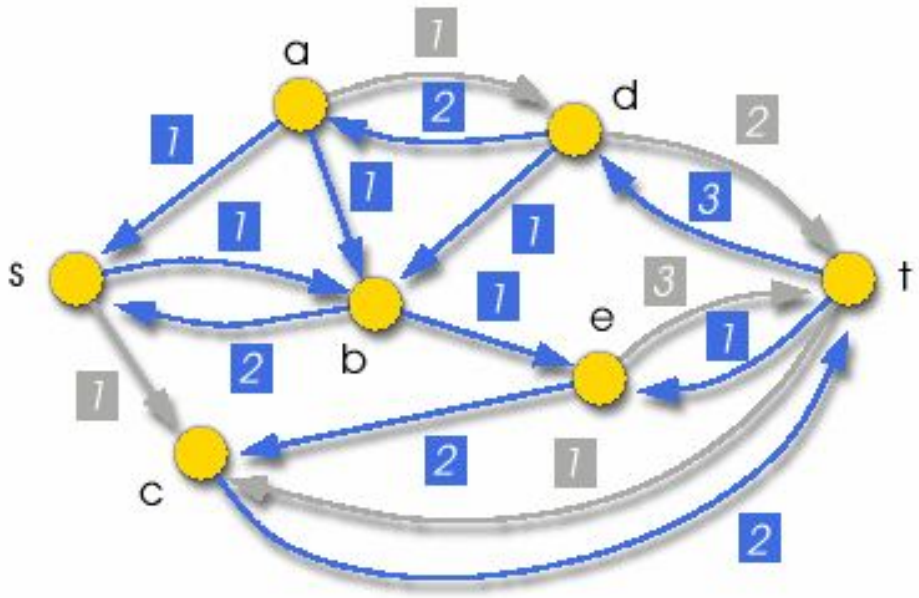
Flots et Coupe

Soit le graphe de départ suivant :

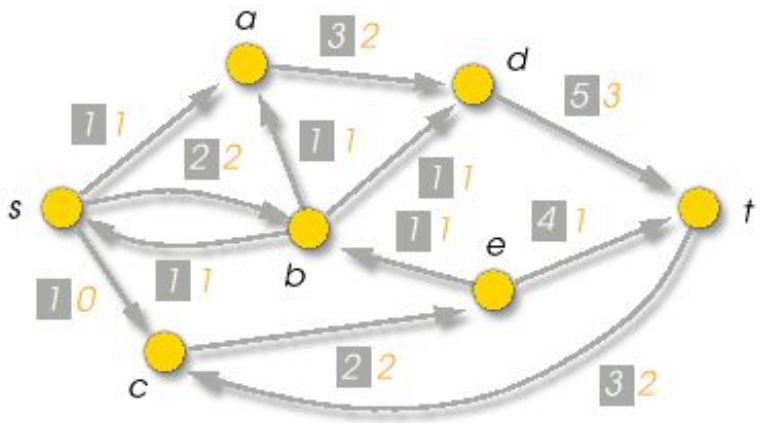


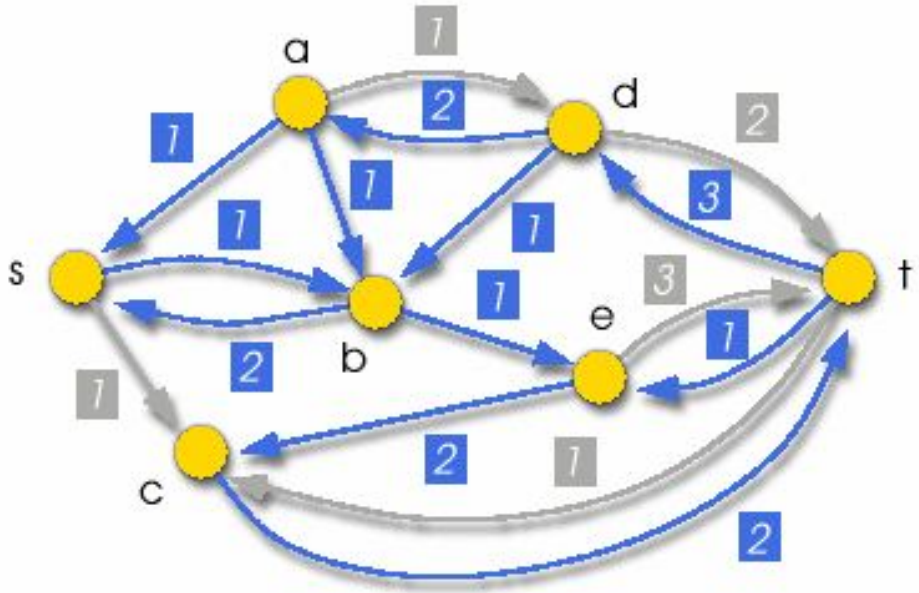
Flots et Coupe

Le réseau résiduel associé au flot F est représenté ci-dessous. Les arcs forward apparaissent en gris, les arcs backward sont en bleu. Les arcs de capacité nulle n'ont pas été représentés.



Flots et Coupe





Flots et Coupe

- Dans le réseau N, l'arc (s, a) est saturé : seul apparaît dans le réseau résiduel son arc backward (a, s), de capacité 1 (capacité de l'arc d'origine).
- Dans le réseau N, aucun flot ne passe par l'arc (s, c) : seul apparaît dans le réseau résiduel l'arc forward (s, c), de capacité 1 (capacité de l'arc d'origine)
- L'arc (d, t) a une capacité de 5 dans le réseau N et un flot de 3 y circule.
  Dans le réseau résiduel lui sont alors associés :
  1. Un arc forward (d, t) de capacité 2 qui correspond à la quantité de flot pouvant encore transiter par cet arc ,
  2. Un arc backward (t, d) de capacité 3 qui correspond à la quantité de flot pouvant être diminuée sur cet arc.

Flots et Coupe

Rapport entre un flot f sur le réseau résiduel N_F et le flot F sur le réseau N

A partir de ces 2 flots peut être construit un nouveau flot F' sur N :

- En ajoutant à F la valeur du flot f sur les arcs forward,
- En retranchant à F la valeur du flot f sur les arcs backward.

La valeur de ce nouveau flot F' est alors égale à la valeur de F plus celle de f : Une augmentation de la valeur du flot transitant sur le réseau N a été alors réalisée !!!

Flots et Coupe

Ajout d'un flot résiduel

Si f est un flot sur le réseau résiduel , nous définissons la valuation des arcs $F' = F \oplus f$ sur le réseau N par :

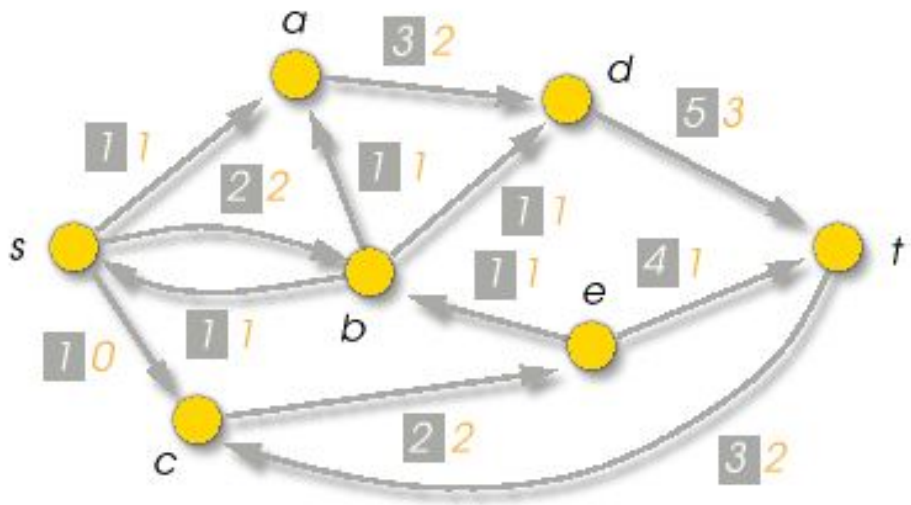
$$F'(x,y) = F(x,y) + f(x,y)_F - f(y,x)_B$$

Alors :

- F' est un flot sur le réseau de départ N
- La valeur du flot F' est la somme des 2 flots : $|F'| = |F| + |f|$

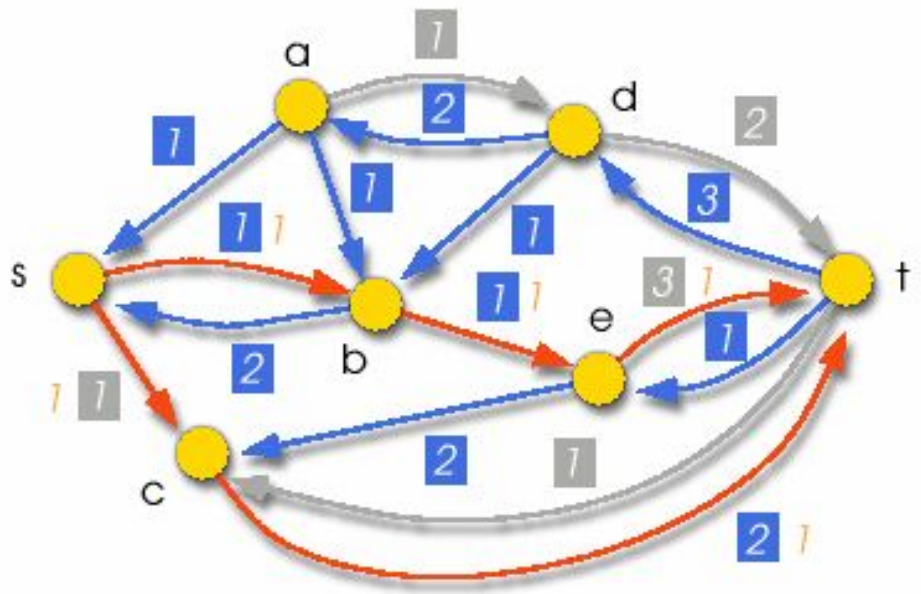
Flots et Coupe

Reprenons l'exemple du flot F sur le réseau N. La valeur du flot F est 2.



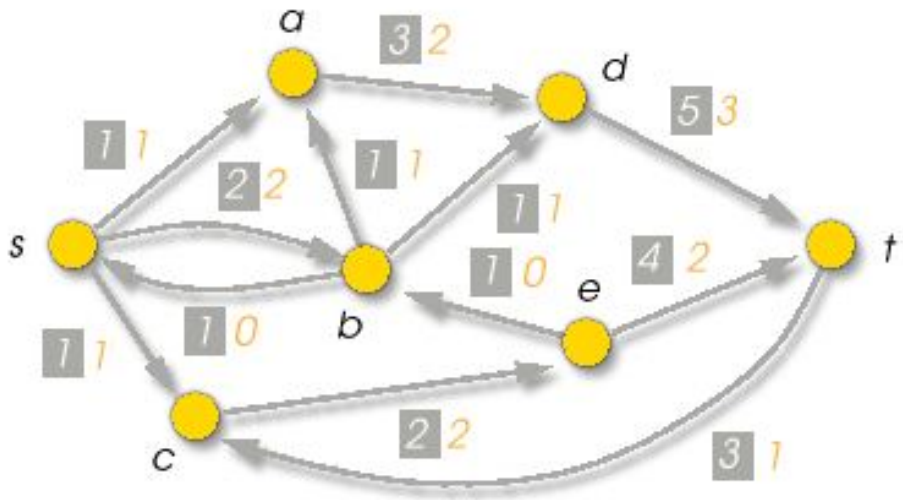
Flots et Coupe

Nous pouvons définir sur le réseau résiduel N_F le flot f suivant. Sa valeur est 2. Il emprunte les arcs forward (s, c) et (e, t), et les arcs backward $(s, b), (b, e)$ et (c, t). Sur cet exemple la valuation du flot f sur tous ces arcs est de 1.



Flots et Coupe

Le flot résultant $F \oplus f$ sur le réseau N est représenté. Les arcs (s, c) et (e, t) ont vu leur flot augmenté de 1, le flot f empruntant leur arc forward. A l'inverse les arcs $(b, s), (e, b)$ et (t, c) ont vu leur flot diminuer de 1, le flot f empruntant leur arc backward.
La valeur du flot $F \oplus f$ est bien de 4.



Flots et Coupe

Trouver un flot sur le réseau résiduel permet donc de faire augmenter, en l'ajoutant, le flot F sur le réseau N.
La manière la plus simple de rechercher un flot est de regarder si il existe un chemin (orienté) de capacité non nulle reliant s à t.
Il est alors possible de faire passer un flot non nul en saturant le chemin.
La notion de **chemin augmentant** reprend cette idée dans le réseau résiduel.

Flots et Coupe

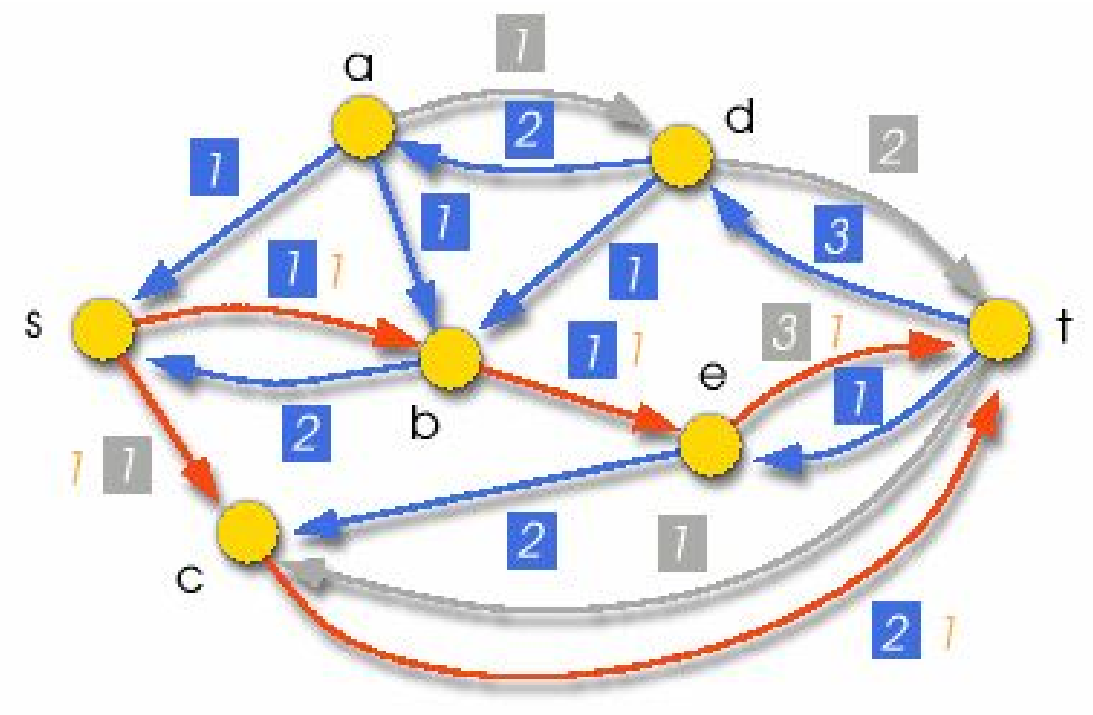
Chemin augmentant

Un chemin augmentant pour un flot F sur un réseau N est un chemin (orienté) de capacité non nulle reliant s à t dans le réseau résiduel N_F .

Avec la convention de ne considérer que les arcs de capacité non nulle, un chemin augmentant p est simplement un chemin orienté de s à t dans le réseau résiduel .
Il est "augmentant" puisque le flot résultant $F \oplus p$ sur le réseau N augmente de la capacité du chemin p. Lorsque est écrit $F \oplus p$, il est entendu implicitement pour p le flot correspondant à la saturation du chemin.

Flots et Coupe

Dans notre exemple le réseau résiduel admettait comme chemin augmentant (s, b, e, t) et (s, c, t), mais aussi (s, b, e, c, t), (s, c, t, d, b, e, t), etc, tous de capacité 1.



Flots et Coupe

Principe

Pour faire augmenter le flot F, les chemins (orientés) de s à t dans le réseau résiduel vont être saturés.
L'algorithme consiste alors à partir d'un flot F réalisable (le flot nul fait parfaitement l'affaire) et à l'améliorer itérativement.
Pour cela à chaque étape l'algorihtme vérifie si il existe un chemin augmentant pour le flot, c'est à dire un chemin (orienté) de s à t dans le réseau résiduel.
Si un tel chemin existe, il est saturé dans et le flot correspondant est "ajouté" à F.
Sinon, il n'existe plus de chemin augmentant et l'algorithme retourne le flot courant.
Le principe de cet algorithme est du à Ford-Fulkerson dans les années 60.

Flots et Coupe

Algorithme de Ford-Fulkerson

1. Partir d'un flot initial compatible avec les capacités, par exemple $f_0 = 0$ et $k = 0$.
2. Iteration k : Rechercher un chemin C_k de s à t dans le graphe d'écart (résiduel) G_f^k . S'il n'en existe pas, stop : le flot f^k est maximum. Sinon aller à 3.
3. Soit ϵ^k le minimum des capacités des arcs du $chemin_k$ (capacité résiduelle). Definir le flot f^{k+1} par :
   - $f^{k+1}(a) = f^k(a) + \epsilon^k$ si l'arc forward $a \in C_k$,
   - $f^{k+1}(a) = f^k(a) - \epsilon^k$ si l'arc backward $a \in C_k$,
   - $f^{k+1}(a) = f^k(a)$ sinon

   Faire $k = k + 1$ puis retourner à 2.

Flots et Coupe

Algorithme de Ford-Fulkerson - autre angle

ENTREES Réseau $N = (V, A)$, s, t des sommets de V
SORTIE F un flot maximum entre s et t

Initialiser $F := 0$ // On part d'un flot possible entre s et t
Tant Que il existe un chemin augmentant p de s à t
Saturer le chemin p dans le réseau résiduel
$F := F \oplus p$
Fin Tant Que
Retourner F

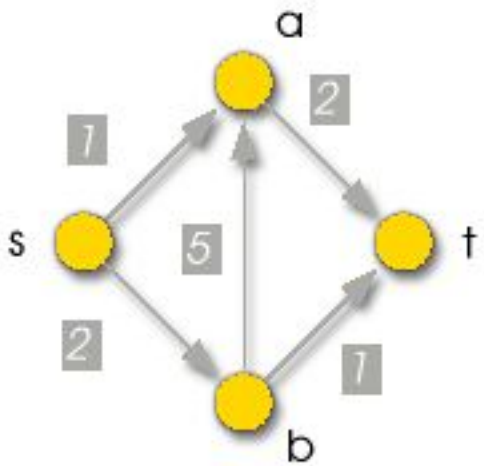
Flots et Coupe

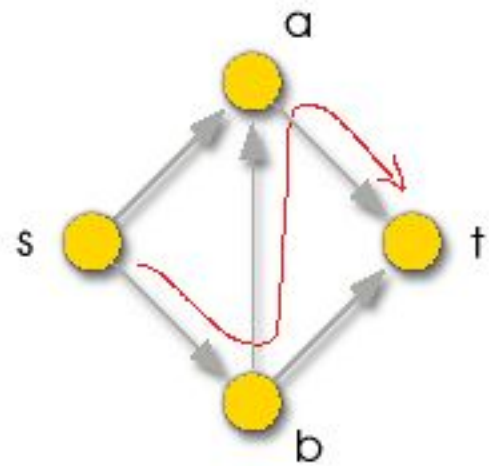
Théorème

L'algorithme de Ford-Fulkerson délivre un flot maximum.

Flots et Coupe

Déroulons l'algorithme sur un petit exemple où il est facile de voir que le flot maximum a une valeur de 3.

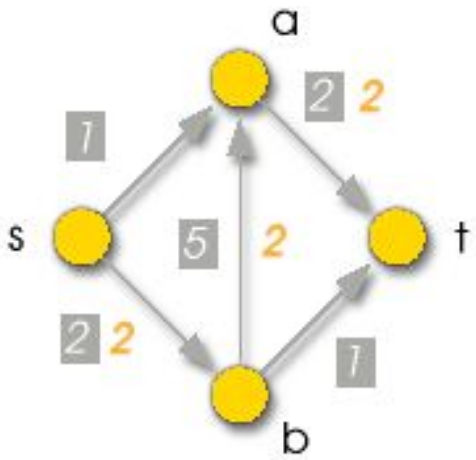


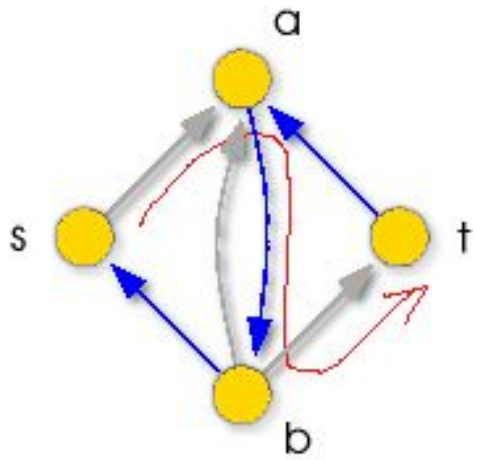


Le réseau apparaît à gauche. Initialement un flot nul transite sur les arcs. Son réseau résiduel représenté à droite est alors bien sûr identique au réseau de départ. Un chemin augmentant possible est (s, b, a, t), de capacité 2.

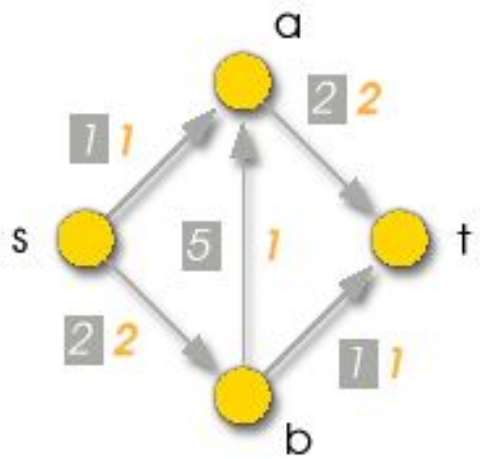
Flots et Coupe

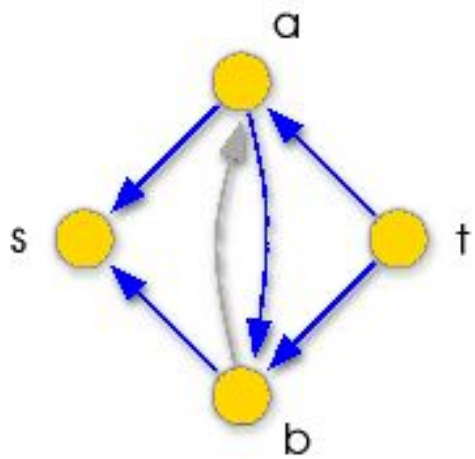
La saturation du chemin passe le flot F à 2. Il existe un chemin augmentant (s, a, b, t) dans le réseau résiduel.





Flots et Coupe

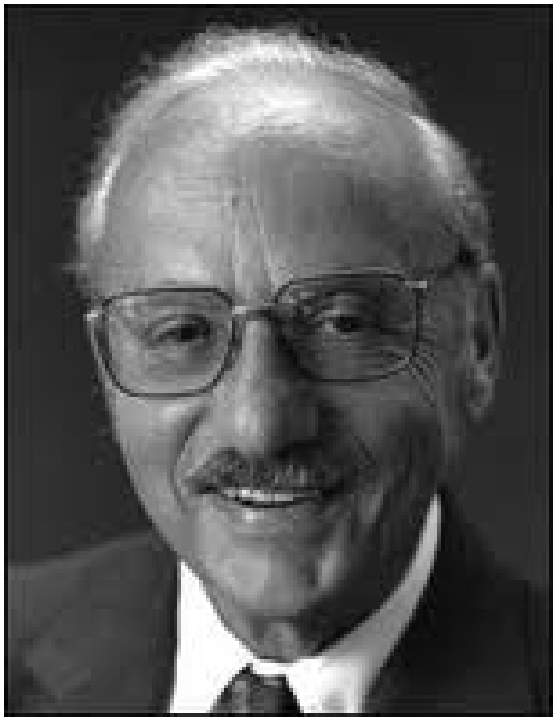




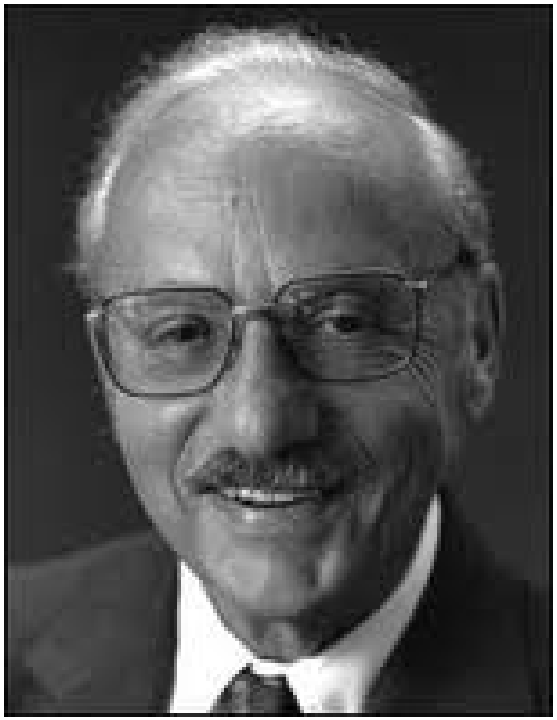
La saturation du chemin augmentant fait passer le flot F à 3. Il n'existe plus de chemin de s à t dans le réseau résiduel. L'algorithme de Ford-Fulkerson s'arrête et délivre le flot F, qui est optimal.

Outline

Programmation linéaire



Programmation linéaire



George B. Dantzig (1914-2005)

Programmation linéaire

Un peu d'histoire !

Son père, Tobias, est un mathématicien russe qui avait étudié avec Henri Poincaré à Paris. Il a épousé une collègue de la Sorbonne, Anja Ourisson, et le couple a émigré aux États-Unis.
Il est l'acteur principal d'une histoire fameuse en mathématique. Dans l'un de ses cours de doctorat à l'Université de Berkeley, le professeur Jerzy Neyman a proposé deux problèmes dits ouverts en statistiques. Un problème ouvert est un problème qui bien qu'ayant été formulé, n'a pas encore été résolu. De tels problèmes sont d'une difficulté importante et demandent des recherches pouvant s'étaler sur plusieurs années. Dantzig était en retard et croyait qu'il s'agissait de devoirs. Sans prendre plusieurs années mais bien quelques jours, il les a résolus.

Problèmes de programmation mathématique

De manière générale, la résolution de problèmes de programmation mathématique vise à déterminer l'allocation optimale (c'est-à-dire la meilleure combinaison possible) de ressources limitées pour atteindre certains objectifs. Les allocations doivent minimiser ou maximiser une fonction dite objectif. En économie, ces fonctions sont par exemple le profit ou le coût.

Problèmes de programmation mathématique

Programmation mathématique : forme générale

Optimiser $z = f(x_1, x_2, ..., x_n)$

sous les contraintes

$$h_i(x_1, x_2, ..., x_n) \quad \{\leq, =, \geq\}\ b_i$$

$$i = 1, 2, ..., m$$

où les fonctions f et h_i sont des fonctions numériques à n variables. La fonction f est la fonction objectif à optimiser, tandis que les équations ou inéquations sont les contraintes.
Lorsque les fonctions f et $h_i, i = 1, ..., m$ sont linéaires, il s'agit d'un problème de **programmation linéaire**.
Si de plus, on impose que les variables ne peuvent prendre que des valeurs entières, on parle de **programmation linéaire entière**.

Problèmes de programmation linéaire

La programmation linéaire est définie comme étant un cas particulier de la programmation mathématique pour laquelle la fonction objectif et les contraintes sont linéaires. De plus, les variables sont supposées être non-négatives. Un problème de programmation linéaire revient donc à :

Optimiser $z = c_1x_1 + c_2x_2 + ... + c_nx_n$

sous les contraintes

$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n \quad \{\leq, =, \geq\}\, b_i$$

$$i = 1, 2, ..., m$$

et

$$x_j \geq 0, j = 1, ..., n \text{ (contraintes de nonnégativité)}$$

où a_{ij}, b_i et c_j sont des constantes connues.

Problèmes de programmation linéaire : un exemple

La campagne électorale

Objectif : remporter les élections régionales à moindre frais. La zone du scrutin regroupe :

1. le centre ville : 100 000 inscrits
2. la banlieue : 200 000 inscrits
3. la campagne : 50 000 inscrits

Problèmes de programmation linéaire : un exemple

La campagne électorale : les frais

Il est possible d'influencer grâce à la publicité sur les thèmes suivants :

1. construction de routes
2. lutte anti-drogue
3. subvention aux agriculteurs
4. taxes sur les engrais pour améliorer l'eau

Problèmes de programmation linéaire : un exemple

La campagne électorale : Influence de la publicité dans les votes

Nombre de voies gagnées/perdues par 1 000 €dépensés en publicité.

thèmes	**ville**	**banlieue**	**campagne**
les routes	-2 000	+5 000	+3 000
la drogue	+8 000	+2 000	-5 000
subventions	0	0	+10 000
taxes pour l'eau	10 000	0	-2 000

Problèmes de programmation linéaire : un exemple

Le problème à résoudre

Le problème est donc de connaître la somme minimale à investir en communication pour que le nombre de voies en faveur du candidat soit de :

- 50 000 en ville ;
- 100 000 en banlieue ;
- 25 000 à la campagne.

$\Longrightarrow$ Comment résoudre ce problème : des essais/erreurs, le hasard, ... ?

Problèmes de programmation linéaire : un exemple

Définitions

- 20 000 € pour le thème (1) ;
- 0 € pour le thème (2) ;
- 4 000 € pour le thème (3) ;
- 9 000 € pour le thème (4) ;

$\Longrightarrow$

- 50 000 votes en ville ;
- 100 000 votes en banlieue ;
- 82 000 votes à la campagne.

Problèmes de programmation linéaire : un exemple

Définitions

- 20 000 € pour le thème (1) ;
- 0 € pour le thème (2) ;
- 4 000 € pour le thème (3) ;
- 9 000 € pour le thème (4) ;

$\Longrightarrow$

- 50 000 votes en ville ;
- 100 000 votes en banlieue ;
- 82 000 votes à la campagne.

$\Longrightarrow$ 33 000 € en communication.

Problèmes de programmation linéaire : un exemple

Définitions

- 20 000 € pour le thème (1) ;
- 0 € pour le thème (2) ;
- 4 000 € pour le thème (3) ;
- 9 000 € pour le thème (4) ;

$\Longrightarrow$

- 50 000 votes en ville ;
- 100 000 votes en banlieue ;
- 82 000 votes à la campagne.

$\Longrightarrow$ 33 000 € en communication.

Une meilleure solution existe-t-elle ?

Problèmes de programmation linéaire : un exemple

Formulation mathématique

Posons le problème sous forme mathématiques :

- soit x_1 la somme dépensée en communication pour le thème (1) ;
- soit x_2 la somme dépensée en communication pour le thème (2) ;
- soit x_3 la somme dépensée en communication pour le thème (3) ;
- soit x_4 la somme dépensée en communication pour le thème (4) ;

Problèmes de programmation linéaire : un exemple

Formulation mathématique

Posons le problème sous forme mathématiques :

- soit x_1 la somme dépensée en communication pour le thème (1) ;
- soit x_2 la somme dépensée en communication pour le thème (2) ;
- soit x_3 la somme dépensée en communication pour le thème (3) ;
- soit x_4 la somme dépensée en communication pour le thème (4) ;

D'où les contraintes du problème pour les 3 secteurs :

Problèmes de programmation linéaire : un exemple

Formulation mathématique

Posons le problème sous forme mathématiques :

- soit x_1 la somme dépensée en communication pour le thème (1) ;
- soit x_2 la somme dépensée en communication pour le thème (2) ;
- soit x_3 la somme dépensée en communication pour le thème (3) ;
- soit x_4 la somme dépensée en communication pour le thème (4) ;

D'où les contraintes du problème pour les 3 secteurs :

$$\begin{cases} \text{en ville} & -2x_1 + 8x_2 + 0x_3 + 10x_4 \geq 50 \\ \text{en banlieue} & 5x_1 + 2x_2 + 0x_3 + 0x_4 \geq 100 \\ \text{à la campagne} & 3x_1 - 5x_2 + 10x_3 - 2x_4 \geq 25 \end{cases}$$

Problèmes de programmation linéaire : un exemple

Formulation mathématique

Posons le problème sous forme mathématiques :

- soit x_1 la somme dépensée en communication pour le thème (1) ;
- soit x_2 la somme dépensée en communication pour le thème (2) ;
- soit x_3 la somme dépensée en communication pour le thème (3) ;
- soit x_4 la somme dépensée en communication pour le thème (4) ;

D'où les contraintes du problème pour les 3 secteurs :

$$\begin{cases} \text{en ville} & -2x_1 + 8x_2 + 0x_3 + 10x_4 \geq 50 \\ \text{en banlieue} & 5x_1 + 2x_2 + 0x_3 + 0x_4 \geq 100 \\ \text{à la campagne} & 3x_1 - 5x_2 + 10x_3 - 2x_4 \geq 25 \end{cases}$$

La fonction à minimiser est $x_1 + x_2 + x_3 + x_4$ avec $x_i \geq 0$ pour $i = 1, 2, 3$ et 4.

Problèmes de programmation linéaire : un exemple

Formulation mathématique

Posons le problème sous forme mathématiques :

- soit x_1 la somme dépensée en communication pour le thème (1) ;
- soit x_2 la somme dépensée en communication pour le thème (2) ;
- soit x_3 la somme dépensée en communication pour le thème (3) ;
- soit x_4 la somme dépensée en communication pour le thème (4) ;

D'où les contraintes du problème pour les 3 secteurs :

$$\begin{cases} \text{en ville} & -2x_1 + 8x_2 + 0x_3 + 10x_4 \geq 50 \\ \text{en banlieue} & 5x_1 + 2x_2 + 0x_3 + 0x_4 \geq 100 \\ \text{à la campagne} & 3x_1 - 5x_2 + 10x_3 - 2x_4 \geq 25 \end{cases}$$

La fonction à minimiser est $x_1 + x_2 + x_3 + x_4$ avec $x_i \geq 0$ pour $i = 1, 2, 3$ et 4.

$\Rightarrow$ L'ensemble de ces équations forment un **programme linéaire**.

Formulation générale

Soit la fonction linéaire $f(x_1, x_2, ..., x_n) = \sum_{j=1}^{n} a_j x_j$

Une équation linéaire s'exprime sous la forme $f(x_1, x_2, ..., x_n) = b$.

Une inégalité linéaire : $f(x_1, x_2, ..., x_n) \leq b$ ou $\geq b$ (inégalité au sens large en programmation linéaire).

Le problème de programmation linéaire est la **maximisation/minimisation** d'une fonction linéaire.

La résolution de ce problème se fait par la méthode du **simplexe**.

Résolution d'un programme linéaire

Plusieurs manières d'exprimer un programme linéaire :

- la forme canonique ($\leq$)
- la forme standard ($=$)

$\Rightarrow$ **Optimisation d'une fonction objectif linéaire**

Résolution d'un programme linéaire

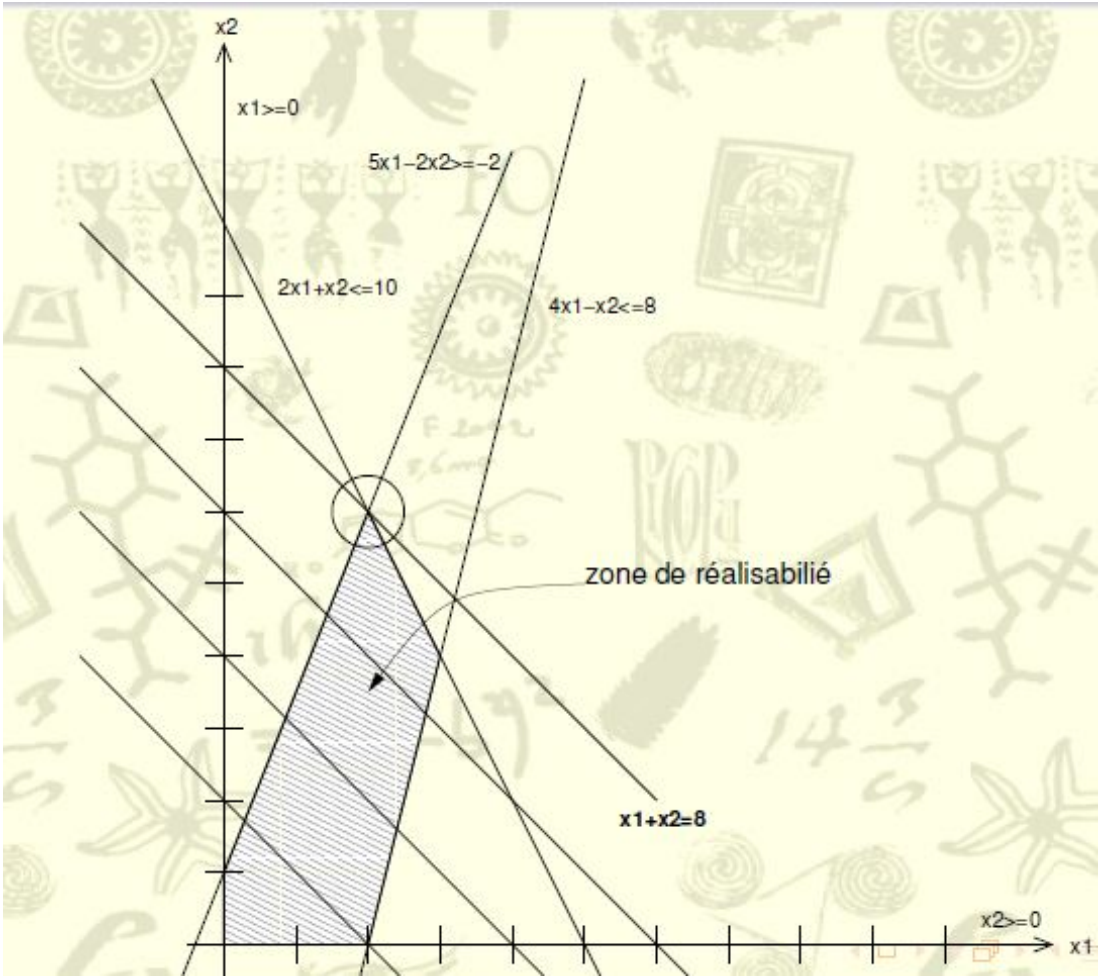
Exemple

Soit le programme linéaire à 2 variables suivant :

$$\begin{cases} \text{maximiser} & x_1 + x_2 \\ \text{sous les contraintes} & 4x_1 - x_2 \leq 8 \\ & 2x_1 + x_2 \leq 10 \\ & 5x_1 - 2x_2 \leq -2 \\ & \text{et} \quad x_1, x_2 \geq 0 \end{cases}$$

$\Longrightarrow$ Résolution graphique en 2D en partitionnant l'espace à chaque contrainte pour la définition d'une **zone réalisable**.

Résolution graphique



Généralisation

- Même démarche pour un programme linéaire avec un plus grand nombre de variables
- Définition de la zone réalisable (simplexe) par l'intersection des demi-espaces donnés par les contraintes
- Solution : intersection entre l'hyperplan mobile de la fonction objectif et le simplexe (un des sommets de la zone réalisable toujours convexe).

Applications

La programmation linéaire est utilisée pour résoudre des problèmes d'optimisation comme :

- la planification de l'affectation des personnels sur les vols des compagnies aériennes
- la maximisation du pétrole extrait en fonction du nombre de puits de forages
- la résolution du problème de graphes et de combinatoires des flots
- le calcul d'un ordonnancement de tâches sur des machines en régime permanent
- ...

Résolution algorithmique

- Algorithmique du simplexe (très bonne performances même s'il s'avère exponentiel dans le pire cas !!)
- Algorithme de l'ellipsoïde (linéaire mais très lent en pratique)
- Solutions en nombre entiers : problème NP-Complet !!

Forme canonique et forme standard

Différente manière d'écrire un programme linéaire :

- forme canonique
- forme standard (algorithme du simplexe)

Forme canonique

Le programme linéaire écrit avec la forme canonique $(n + m)$ inégalités avec n variables et m contraintes :

$$\begin{cases} \text{maximiser} & \sum_{j=1}^{n} c_j x_j \text{ fonction objectif} \\ \text{sous les contraintes} & \sum_{j=1}^{n} a_{ij} x_j \leq b_i \text{ avec } i = 1, .., m \\ \text{et les contraintes de positivité} & x_j \geq 0, \text{ avec } j = 1, ..., n \end{cases}$$

Exemple de conversion

Soit le programme linéaire suivant :

$$\begin{cases} \text{minimiser} & -2x_1 + 3x_2 \\ \text{sous les contraintes} & x_1 + x_2 = 7 \\ & x_1 \quad -2x_2 \leq 4 \\ & x_1 \quad \geq 0 \end{cases}$$

Exemple de conversion

Soit le programme linéaire suivant :

$$\begin{cases} \text{minimiser} & -2x_1 + 3x_2 \\ \text{sous les contraintes} & x_1 + x_2 = 7 \\ & x_1 \quad -2x_2 \leq 4 \\ & x_1 \quad \geq 0 \end{cases}$$

1. Minimiser une fonction = maximiser la fonction opposée ($\times -1$)

Exemple de conversion

Soit le programme linéaire suivant :

$$\begin{cases} \text{minimiser} & -2x_1 + 3x_2 \\ \text{sous les contraintes} & x_1 + x_2 = 7 \\ & x_1 \quad -2x_2 \leq 4 \\ & x_1 \quad \geq 0 \end{cases}$$

1. Minimiser une fonction = maximiser la fonction opposée $(\times -1)$
2. Les contraintes de positivité : $x_j \rightarrow x_j^{'} - x_j^{''}$ avec $x_j^{'}, x_j^{''} \geq 0$

Exemple de conversion

Soit le programme linéaire suivant :

$$\begin{cases} \text{minimiser} & -2x_1 + 3x_2 \\ \text{sous les contraintes} & x_1 + x_2 = 7 \\ & x_1 \quad -2x_2 \leq 4 \\ & x_1 \quad \geq 0 \end{cases}$$

1. Minimiser une fonction = maximiser la fonction opposée $(\times -1)$
2. Les contraintes de positivité : $x_j \rightarrow x_j^{'} - x_j^{''}$ avec $x_j^{'}, x_j^{''} \geq 0$
3. Transformation des égalités en deux contraintes : $x_1 + x_2^{'} - x_2^{''} = 7$

$$\begin{cases} x_1 \quad +x_2^{'} - x_2^{''} \leq 7 \\ x_1 \quad +x_2^{'} - x_2^{''} \geq 7 \end{cases}$$

Version canonique

Le programme linéaire devient le programme suivant :

$$\begin{cases} \text{maximiser} & 2x_1 - 3x_2^{'} + 3x_2^{''} \\ \text{sous les contraintes} & x_1 + x_2^{'} - x_2^{''} = 7 \\ x_1 & -2x_2^{'} + 2x_2^{''} \leq 4 \\ x_1, x_2^{'}, x_2^{''} & \geq 0 \end{cases}$$

Version canonique du programme linéaire

$$\begin{cases} \text{maximiser} & 2x_1 - 3x_2 + 3x_3 \\ \text{sous les contraintes} & x_1 + x_2 - x_3 \leq 7 \\ -x_1 & -x_2 + x_3 \leq -7 \\ x_1 & -2x_2 + 2x_3 \leq 4 \\ x_1, x_2, x_3 & \geq 0 \end{cases}$$

Avec $x_2^{'} = x_2$ et $x_2^{''} = x_3$

Conversion en forme standard

$\Rightarrow$ Forme standard utilisée par l'algorithme du simplexe
$\hookrightarrow$ Ajouts des variables d'écart à la valeur de la contrainte pour toutes les inéquations de la forme canonique

$$\text{Si } \sum_{j=1}^{n} a_{ij} x_j \leq b_i$$

Introduction de la variable d'écart s :

$$s = b_i - \sum_{j=1}^{n} a_{ij} x_j \text{ avec } s \geq 0$$

De manière générale

- Soit x_{n+i} la variable d'écart de la i-ème contrainte (si n contraintes)
- Écriture de la i-ème contrainte sous la forme de l'égalité :

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j \text{ avec } x_{n+i} \geq 0$$

Version standard

$$\begin{cases} \text{maximiser} & 2x_1 - 3x_2 + 3x_3 \\ \text{sous les contraintes} & x_4 = 7 - x_1 - x_2 + x_3 \\ x_5 = & -7 + x_1 + x_2 - x_3 \\ x_6 = & 4 - x_1 + 2x_2 - 2x_3 \\ x_1, x_2, x_3, x_4, x_5, x_6 & \geq 0 \end{cases}$$

Version standard (suite)

- Les variables d'écart : **les variables de base**
- Les autres : les variables hors base
- On pose z la valeur de l'objectif
- Exemple précédent :

$$\begin{cases} z = & 2x_1 - 3x_2 + 3x_3 \\ x_4 = & 7 - x_1 - x_2 + x_3 \\ x_5 = & -7 + x_1 + x_2 - x_3 \\ x_6 = & 4 - x_1 + 2x_2 - 2x_3 \end{cases}$$

Notations

On pose :

- $N = \{$ indices des variables hors base $\}$ avec $(|N| = n)$
- $B = \{$ indices des variables de base $\}$ avec $(|B| = m)$
- A la matrice des coefficients des variables hors base
- b le vecteur des constantes dans les égalités linéaires
- c le vecteur des variables de la fonction objectif
- v la constante de la fonction objectif

$$\begin{cases} z = & v + \sum_{j \in N} c_i x_j \\ x_i = & b_i - \sum_{j \in N} a_{ij} x_j \text{ pour } i \in B \end{cases}$$

$\Longrightarrow (N, B, A, b, C, v)$: la forme standard d'un programme linéaire.

Exemple de forme standard

$$\begin{cases} z = & 28 - \frac{x_3}{6} - \frac{x_5}{6} - \frac{2x_6}{3} \\ x_1 = & 8 + \frac{x_3}{6} + \frac{x_5}{6} - \frac{x_6}{3} \\ x_2 = & 4 - \frac{8x_3}{3} - \frac{2x_5}{3} + \frac{x_6}{3} \\ x_4 = & 18 - \frac{x_3}{2} + \frac{x_5}{6} \end{cases}$$

avec

$$B = \{1,2,4\}\, N = \{3,5,6\}\,, A = \begin{pmatrix} -1/6 & -1/6 & 1/3 \\ 8/3 & 2/3 & -1/3 \\ 1/2 & -1/2 & 0 \end{pmatrix}, b = \begin{pmatrix} 8 \\ 4 \\ 18 \end{pmatrix}, c = \begin{pmatrix} -1/6 \\ -1/6 \\ 2/3 \end{pmatrix}, v = 28$$

Algorithme du simplexe

- Algorithme non polynomial dans le pire cas
- Excellentes performances en pratique
- Mode de calcul semblable à la méthode du pivot de Gauss
- Programme linéaire écrit sous la forme standard
- Très souvent, différents formats de représentation des programmes linéaires sont admis

Algorithme du simplexe : Principe

$\Longrightarrow$ **Maximiser la fonction objectif en maximisant chacune des variables hors base ayant un coefficient positif dans la fonction objectif.**

- extraction des variables de base les unes après les autres
- remplacement dans le reste du programme
- si plus aucune variable n'a de coefficient positif dans la fonction objectif alors il n'est plus possible d'augmenter sa valeur
- mise à 0 des variables de la fonction objectif à cet instant
- calcul de la valeur des variables hors base

$\Longrightarrow$ On a alors la solution du programme linéaire : valeur de la fonction objectif et valeur des variables hors base du programme initial

Exemple d'exécution

Soit le programme canonique suivant :

$$\begin{cases} \text{maximiser} & 3x_1 + x_2 + 2x_3 \\ \text{sous les contraintes} & x_1 + x_2 + 3x_3 \leq 30 \\ 2x_1 & +2x_2 + 5x_3 \leq 24 \\ 4x_1 + & x_2 + 2x_3 \leq 36 \\ x_1, x_2, x_3 & \geq 0 \end{cases}$$

Exemple d'exécution

Voici sa forme standard :

$$\begin{cases} z = & 3x_1 + x_2 + 2x_3 \\ x_4 = & 30 - x_1 - x_2 - 3x_3 \\ x_5 = & 24 - 2x_1 - 2x_2 - 5x_3 \\ x_6 = & 36 - 4x_1 - x_2 - 2x_3 \end{cases}$$

Ce système de contraintes a 3 équations et 6 inconnues. Il y a par conséquent un nombre infini de solution réalisables. Il s'agit de trouver celle qui maximise la fonction objectif.

Exemple d'exécution
Solution de base

- Les solutions de base :
  - mise à 0 des variables à droite du signe = (variable hors base)
  - les variables de base sont donc des constantes des équations linéaires
  - dans l'exemple : les solutions de base sont les suivantes :

$$(\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}, \overline{x_5}, \overline{x_6}) = (0, 0, 0, 30, 24, 36)$$

- La valeur de l'objectif est $z = 0$ et $\overline{x_i} = \forall b_i \in B$. La solution de base n'est pas toujours une solution réalisable. Ceci ne remet pas en question la résolution du programme linéaire par l'algorithme du simplexe.

Exemple d'exécution

Réécriture de l'ensemble des équations et la fonction objectif

$\Longrightarrow$ Recherche d'un ensemble de variables de base différent de celui de la solution de base sans changer le programme linéaire initial :

- réécriture dans une autre forme
- itération sur cette transformation tant que cela est possible (amélioration de la fonction objectif)

Réécriture du programme linéaire

- Choix d'une variable hors base x_e ayant un coefficient positif dans la fonction objectif
- Donner à cette variable x_e la plus grande valeur possible sans qu'aucune contrainte ne soit violée
  - Soit l l'équation du programme linéaire la plus restrictive pour x_e
  - Expression de x_e en fonction de la variable x_l, des autres variables hors base et de la constante
  - x_e devient alors une variable de base
  - x_l devient une variable hors base
- remplacement de x_e par l'expression trouvée au niveau de l'équation l du programme linéaire et dans la fonction objectif

Exemple d'exécution (itération 1)

Choix de la variable x_1

- les autres variables hors base restent à zéro
- on augmente la valeur de x_1 le plus possible, tout en conservant des valeurs positives pour les variables de base
- on ne garde que l'équation imposant la contrainte la plus stricte

On va choisir x_1 pour entrer en base.
Qui va donc en sortir ?

$$x_4 \geq 0 \Leftrightarrow 30 - x_1 \geq 0$$
$$x_5 \geq 0 \Leftrightarrow 24 - 2x_1 \geq 0$$
$$x_6 \geq 0 \Leftrightarrow 36 - 4x_1 \geq 0 \text{ (contrainte la plus stricte !!)}$$

permutation des rôles entre x_1 et x_6 :

- x_1 devient une variable de base
- x_6 quitte la base et devient une variable hors-base !!

Exemple d'exécution (itération 1 - suite)

Expression de x_1 en fonction des autres variables dans l'équation relative à x_6 :

$$x_1 = 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}$$

$\hookrightarrow$ Réécriture de la première équation :

$$x_4 = 30 - x_1 - x_2 - 3x_3$$
$$x_4 = 30 - (9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}) - x_2 - 3x_3$$
$$x_4 = 21 - \frac{3x_2}{4} - \frac{5x_3}{2} - \frac{x_6}{4}$$

Exemple d'exécution (itération 1 - suite)

Réécriture des autres équations et de la fonction objectif :

$$\begin{cases} z = & 27 + \frac{x_2}{4} + \frac{x_3}{2} - \frac{3x_6}{4} \\ x_1 = & 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4} \\ x_4 = & 21 - \frac{3x_2}{4} - \frac{5x_3}{2} - \frac{x_6}{4} \\ x_5 = & 6 - \frac{3x_2}{2} - 4x_3 + \frac{x_6}{2} \end{cases}$$

$\Rightarrow$ Opération de pivot avec x_1 une **variable entrante** et x_6 une **variable sortante**

On cherche alors la valeur de la solution de base en annulant toutes les variables hors base :

- On obtient une valeur de $z = 27$ pour les solutions suivantes :

$$(\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}, \overline{x_5}, \overline{x_6}) = (9, 0, 0, 21, 6, 0)$$

Exemple d'exécution (itération 2)

On choisit d'augmenter la variable x_3 :

On va choisir x_3 pour entrer en base.
Qui va donc en sortir (les variables de base doivent demeurer ≥ 0) ?

$$x_1 \geq 0 \Leftrightarrow 9 - \frac{x_3}{2} \geq 0$$
$$x_4 \geq 0 \Leftrightarrow 21 - \frac{5x_3}{2} \geq 0$$
$$x_5 \geq 0 \Leftrightarrow 6 - 4x_3 \geq 0 \text{ (contrainte la plus stricte !!)}$$

permutation des rôles entre x_3 et x_5 :

- x_3 devient une variable de base
- x_5 quitte la base et devient une variable hors-base !!
- x_3 dans les autres équations ainsi que dans la fonction objectif

Exemple d'exécution (itération 2 - suite)

Écriture du programme linéaire :

$$\begin{cases} z = \frac{111}{4} + \frac{x_2}{16} - \frac{x_5}{8} - \frac{11x_6}{16} \\ x_1 = \frac{33}{4} - \frac{x_2}{16} + \frac{x_5}{8} - \frac{5x_6}{16} \\ x_3 = \frac{3}{2} - \frac{3x_2}{8} - \frac{x_5}{4} + \frac{x_6}{8} \\ x_4 = \frac{69}{4} + \frac{3x_2}{16} + \frac{5x_5}{8} - \frac{x_6}{16} \end{cases}$$

On a alors $z = 111/4$ avec la solution de base :

$$(\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}, \overline{x_5}, \overline{x_6}) = (33/4, 0, 3/2, 69/4, 0, 0)$$

Exemple d'exécution (itération 3)

Augmentation de la valeur de x_2 :

On va choisir x_2 pour entrer en base.
Qui va donc en sortir (les variables de base doivent demeurer ≥ 0) ?

$$x_1 \geq 0 \Leftrightarrow \frac{111}{4} + \frac{x_2}{16} \geq 0$$
$$x_3 \geq 0 \Leftrightarrow \frac{33}{4} - \frac{x_2}{16} \geq 0 \text{ (contrainte la plus stricte !!)}$$
$$x_5 \geq 0 \Leftrightarrow \frac{69}{4} + \frac{3x_2}{16} \geq 0$$

permutation des rôles entre x_2 et x_3 :

- x_2 devient une variable de base
- x_3 quitte la base et devient une variable hors-base !!
- x_2 dans les autres équations ainsi que dans la fonction objectif

Exemple d'exécution (itération 3 - suite)

Écriture du programme linéaire :

$$\begin{cases} z = & 28 - \frac{x_3}{6} - \frac{x_5}{6} - \frac{2x_6}{3} \\ x_1 = & 8 + \frac{x_3}{6} + \frac{x_5}{6} - \frac{x_6}{3} \\ x_2 = & 4 - \frac{8x_3}{3} - \frac{2x_5}{3} + \frac{x_6}{3} \\ x_4 = & 18 - \frac{x_3}{2} + \frac{x_5}{2} \end{cases}$$

Solution du programme linéaire

On annule les valeurs des variables hors base :

$$(\overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4}, \overline{x_5}, \overline{x_6}) = (8, 4, 0, 18, 0, 0) \text{ avec } z_{max} = 28$$

La solution du problème initial est donc :

$$\begin{cases} x_1 = 8 \\ x_2 = 4 \\ x_3 = 0 \end{cases}$$

Flot maximal et programmation linéaire

$$max_{f_{ij},v}\,[F = v]$$

1. **conservation des flux en chaque sommet :**

$$\begin{cases} \sum_{k\in S(s)} f_{sk} - v = 0 \text{ (source } s) \\ -\sum_{i\in P(j)} f_{ij} + \sum_{k\in S(j)} f_{jk} = 0 \ \forall j \neq s,t \\ -\sum_{i\in P(t)} f_{it} + v = 0 \text{ (puits } t) \end{cases}$$

2. **respect des capacités :**

$$f_{ij} \leq c_{ij} \text{ pour toute arête } (i,j) \in \text{ réseau}$$

3. **contrainte de signe :**

$$f_{ij} \geq 0 \text{ pour toute arête } (i,j) \in \text{ réseau}$$

Remarque : les inconnues sont les f_{ij} et la valeur v du flot.

Flot maximal et programmation linéaire

Ecriture matricielle (n sommets et m arêtes)

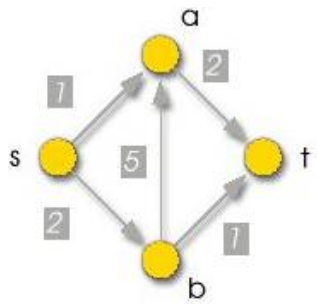
$$max_{f,v}\,[F = v]$$

$$\begin{cases} \mathbf{A}f + \mathbf{v} = & 0 \\ \mathbf{f} & \leq \mathbf{c} \\ \mathbf{f} \geq & 0 \end{cases}$$

A est la matrice d'incidence du graphe, de taille $n \times m$,

$$\mathbf{f} = \begin{pmatrix} (f_{sk})_{k \in S(s)} \\ \vdots \\ f_{ij} \\ \vdots \\ (f_{it})_{i \in P(t)} \end{pmatrix} \in \mathbb{R}^m,\ \mathbf{v} = \begin{pmatrix} -v \\ 0 \\ \vdots \\ 0 \\ +v \end{pmatrix} \in \mathbb{R}^n$$

Flot maximal et programmation linéaire : illustration



A est la matrice d'incidence sommets/arcs du graphe, de taille 4×5,

$$A = \left(\begin{array}{c|ccccc} & (s,a) & (s,b) & (b,a) & (a,t) & (b,t) \\ \hline s & 1 & 1 & 0 & 0 & 0 \\ a & -1 & 0 & -1 & 1 & 0 \\ b & 0 & -1 & 1 & 0 & 1 \\ t & 0 & 0 & 0 & -1 & -1 \end{array}\right) \quad \mathbf{f} = \begin{pmatrix} f_{sa} \\ f_{sb} \\ f_{ba} \\ f_{at} \\ f_{bt} \end{pmatrix} \in \mathbb{R}^5,$$

$$\mathbf{v} = \begin{pmatrix} -v \\ 0 \\ 0 \\ +v \end{pmatrix} \in \mathbb{R}^4$$

Outline

SOLVEUR

Un peu de théorie : Optimisation appliquée (Yadolah Dodge, Sylvie Gonano-Weber et Jean-Pierre Renfer), Statistique et probabilités appliquées, Springer Paris, 2005.

Cours de NICOD Jean-Marc

SOLVEUR

Résolution d'une programmation linéaire avec l'aide du solveur Excel

Exemple

L'entreprise Genco fabrique divers modèles d'appareils électroménagers. Suite à une réunion départementale de divers chefs de services de l'entreprise, il a été convenu d'examiner la possibilité de modifier le programme actuel de fabrication des grille-pains, soit 600 unités de son modèle électronique (QL-500) et 200 unités de son modèle grille-pain/four (QL-700X). L'assemblage se fait essentiellement en deux phases et, par la suite, une vérification (contrôle exhaustif) est effectuée sur toutes les unités. Le tableau suivant donne l'information concernant le nombre d'heures exigées pour fabriquer chaque modèle ainsi que les disponibilités en heures de chaque département .

	Modèles (Nombres d'heures requises)		
Départements	QL-500	QL-700X	Heures disponibles
Assemblage (phase 1)	3	4	4200
Assemblage (phase 2)	1	3	2250
Vérification/Empaquetage	2	2	2600

SOLVEUR

Résolution d'une programmation linéaire avec l'aide du solveur Excel

Étant donné la situation du marché, l'entreprise ne veut pas fabriquer plus de 1100 unités du modèle électronique QL-500.
La contribution au bénéfice du modèle QL-500 est de 66$ l'unité alors que celle du modèle QL-700X est de 84$.

On veut déterminer le programme optimal de fabrication à mettre en œuvre c'est-à-dire celui qui maximiserait les bénéfices.

Variables de décision :
x_1 : le nombre d'unités à fabriquer du modèle QL-500.
x_2 : le nombre d'unités à fabriquer du modèle QL-700X

Les contraintes sont :

C1 : $3x_1 + 4x_2 \leq 4200$ heures (heures disponibles à l'assemblage : phase 1)
C2 : $x_1 + 3x_2 \leq 2250$ heures (heures disponibles à l'assemblage : phase 2)
C3 : $2x_1 + 2x_2 \leq 2600$ heures (heures disponibles : vérification/empaquetage)
C4 : $x_1 \leq 1100$ unités (quantité maximale pour QL-500)
C5 : $x_1 \geq 0$, $x_2 \geq 0$

La fonction économique à **maximiser** est

$Z = 66x_1 + 84x_2$ où Z correspond au bénéfice total ($).

SOLVEUR

Résolution avec EXCEL

Il y a trois principales parties à fournir au solveur d'Excel :

- La cellule à maximiser/minimiser
- La plage de variables de décision (x_1, x_2)
- Les contraintes.

Il y a plusieurs façons de fournir au solveur ces informations. Nous utiliserons dans cet exemple une façon qui se rapproche de la modélisation d'un problème linéaire. Toutefois, vous verrez dans les autres exemples qu'il est parfois plus facile de représenter l'information d'une autre façon.

SOLVEUR

Résolution avec EXCEL

Exemple :

Les cellules B2 et C2 seront les variables du problème (x_1 et x_2).
Chacun des coefficients reliés aux variables pour chaque contrainte est inscrit de **B5 :C8**.
La quantité des ressources est indiquée et le sens de la contrainte. Ce dernier élément est facultatif, il aide seulement comme aide-mémoire au problème.
Il faut indiquer le bénéfice/unité pour chaque variable (**B11 :C11**).

	A	B	C	D	E	F
1	VARIABLES	QL-500	QL-700X			
2	Nombre d'unités					
3						Quantité des
4	CONTRAINTES				sens	ressources
5	Assemblage:phase 1	3	4		<=	4200
6	Assemblage:phase 2	1	3		<=	2250
7	Verifcation et empaquetage	2	2		<=	2600
8	Quantité de QL-500	1	0		<=	1100
9						
10						
11	BÉNÉFICE:	66	84			
12						

SOLVEUR

Résolution avec EXCEL

Première contrainte **: $3x_1 + 4x_2 \leq 4200$** Vous devez calculer l'expression de la partie gauche de l'équation avant d'activer le solveur.
Exemple dans la cellule D5 la formule = $B$2*B5 + $C$2*C5 est inscrite, équivalente à **$3x_1 + 4x_2$** (figure 2).

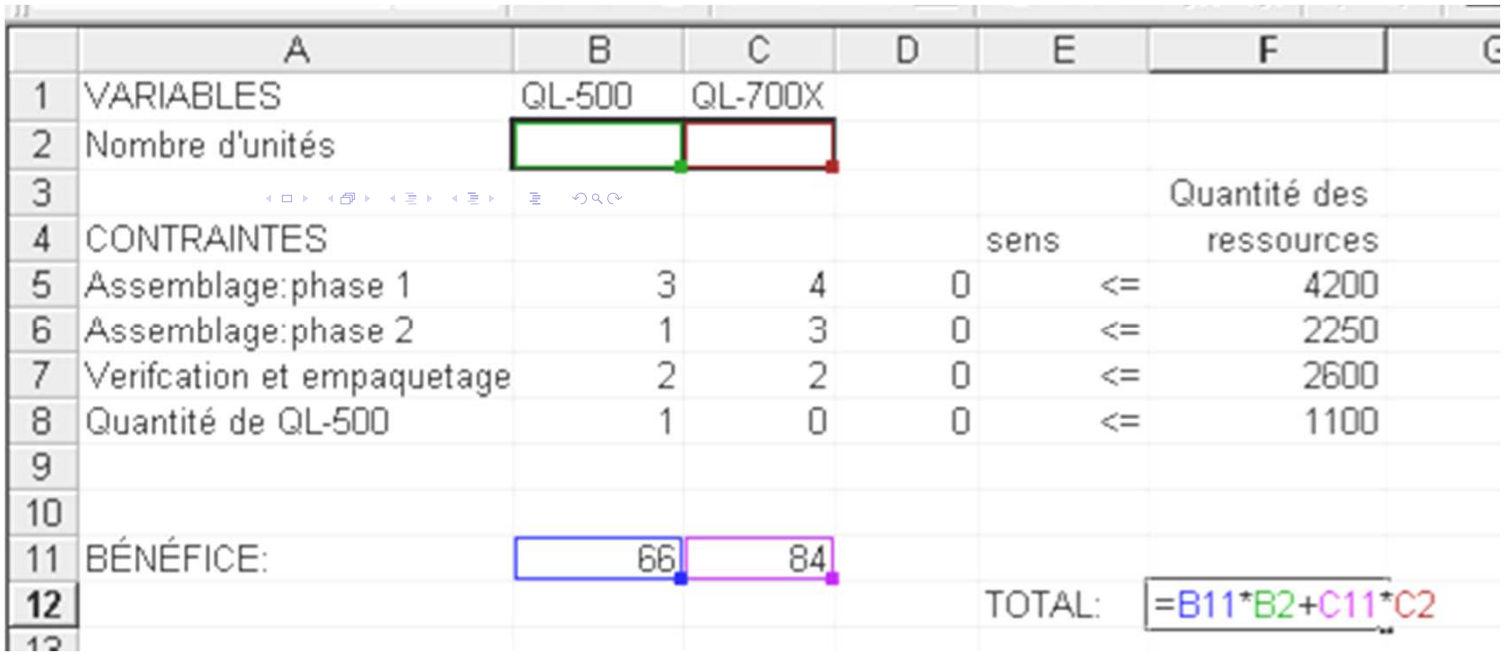
Copiez cette formule pour les autres contraintes.

	A	B	C	D	E	F
1	VARIABLES	QL-500	QL-700X			
2	Nombre d'unités					
3						Quantité des
4	CONTRAINTES				sens	ressources
5	Assemblage:phase 1	3	4	=$B$2*B5+$C$2*C5		
6	Assemblage:phase 2	1	3	0	<=	2250
7	Verifcation et empaquetage	2	2	0	<=	2600
8	Quantité de QL-500	1	0	0	<=	1100
9						
10						
11	BÉNÉFICE:	66	84			
12					TOTAL:	0

SOLVEUR

Résolution avec EXCEL

La formule =B11*B2+C11*C2 est inscrite dans la cellule F12. C'est cette cellule qu'on maximisera car elle correspond à la fonction objectif $66x_1 + 84x_2$

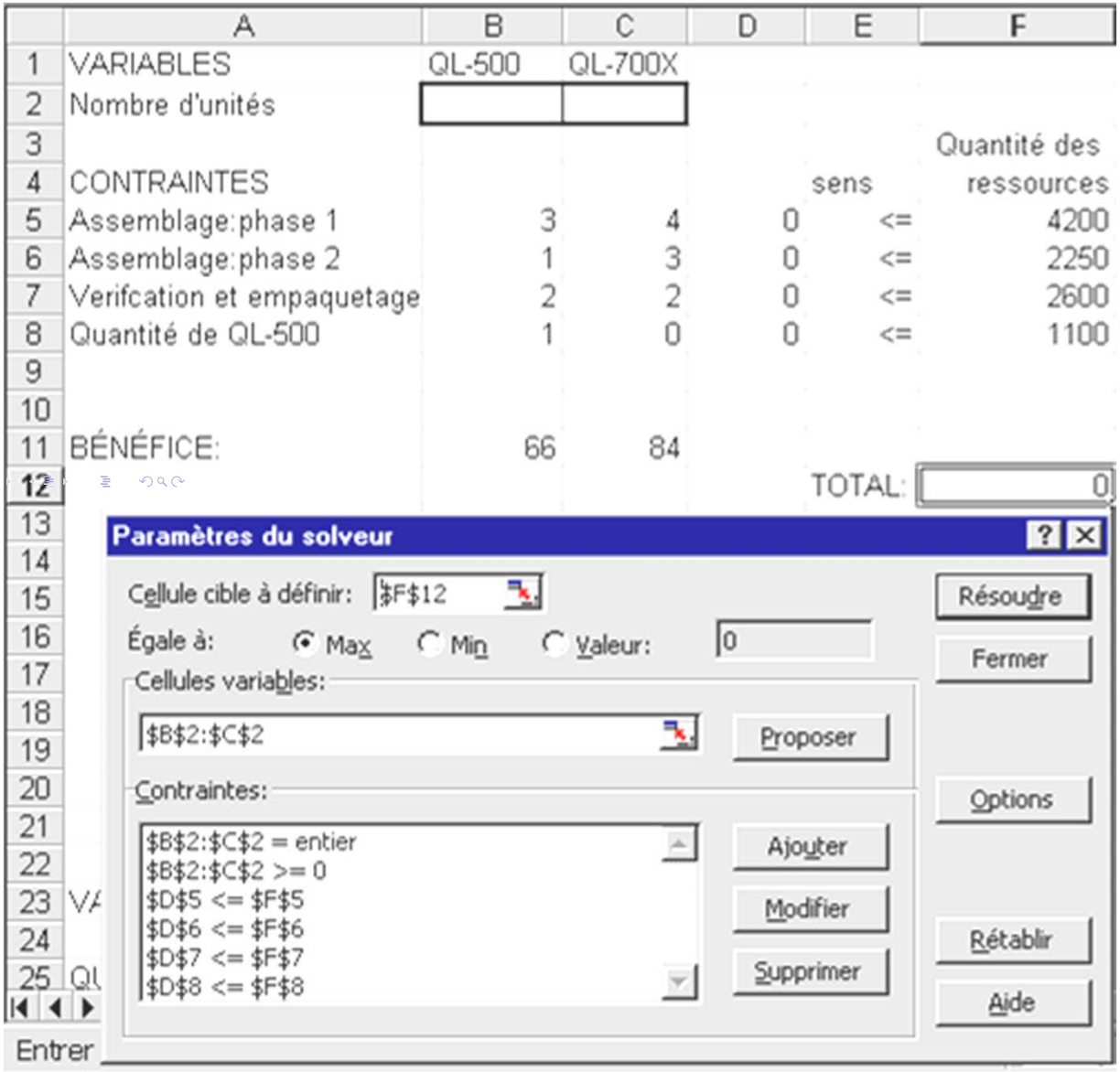


	A	B	C	D	E	F
1	VARIABLES	QL-500	QL-700X			
2	Nombre d'unités					
3						Quantité des
4	CONTRAINTES				sens	ressources
5	Assemblage:phase 1	3	4	0	<=	4200
6	Assemblage:phase 2	1	3	0	<=	2250
7	Verifcation et empaquetage	2	2	0	<=	2600
8	Quantité de QL-500	1	0	0	<=	1100
9						
10						
11	BÉNÉFICE:	66	84			
12					TOTAL:	=B11*B2+C11*C2

SOLVEUR

Résolution avec EXCEL

Entrez les paramètres du solveur



SOLVEUR

Résolution avec EXCEL

Cellule cible à définir: (Exemple: F12) Ceci correspond à l'adresse de la fonction à optimiser.

Égale à: Cochez le type d'optimisation voulu. Le Max est coché car dans cet exemple nous voulons maximiser le bénéfice total ($).

Cellules variables: Sélectionnez l'endroit dans le tableur où les variables se trouvent. Il ne doit pas avoir de cellules vides entre les variables. Les cellules B2:C2 représentent les variables de notre problème, c'est-à-dire celles qu'on désire déterminer.

Contraintes: Vous devez spécifier chacune des contraintes de votre problème.

Il ne faut pas oublier d'entrer les contraintes de non-négativité $x_1 \geq 0$, $x_2 \geq 0$!!

- **Cliquez sur « Ajouter ».**
- **Cellule : Sélectionnez toutes vos variables : (Exemple : B2 :C2)**
- **Inscrivez le sens >=**
- **Contrainte : 0**

SOLVEUR

Résolution avec EXCEL

x_1, x_2 doivent être des entiers afin de ne pas produire des fractions d'unités.

Cliquez sur « Ajouter ».
Cellule : Sélectionnez toutes vos variables : (Exemple : B2 :C2)
Choisissez « ent »

Pour enregistrer les autres contraintes

Cliquez sur "Ajouter".
Exemple : pour la première contrainte : $3x_1 + 4x_2 \leq 4200$ heures

Entrez l'adresse de la cellule contenant la formule : $3x_1 + 4x_2$ équivalente à (=B5*B2+C5*C2). On doit donc entrer D5.
Le sens de l'équation <=.
Le nombre de ressource 4200 ou son adresse F5.

La première contrainte correspond à D5 <= F5.

SOLVEUR

Résolution avec EXCEL

Exemple : pour la dernière contrainte x_1 <= 1100.

Le solveur sépare l'équation en trois.

Le membre gauche de l'équation : c'est-à-dire l'adresse de la cellule contenant la formule B2*B8+C2*C8 donc D8.

Le sens de l'équation : <=

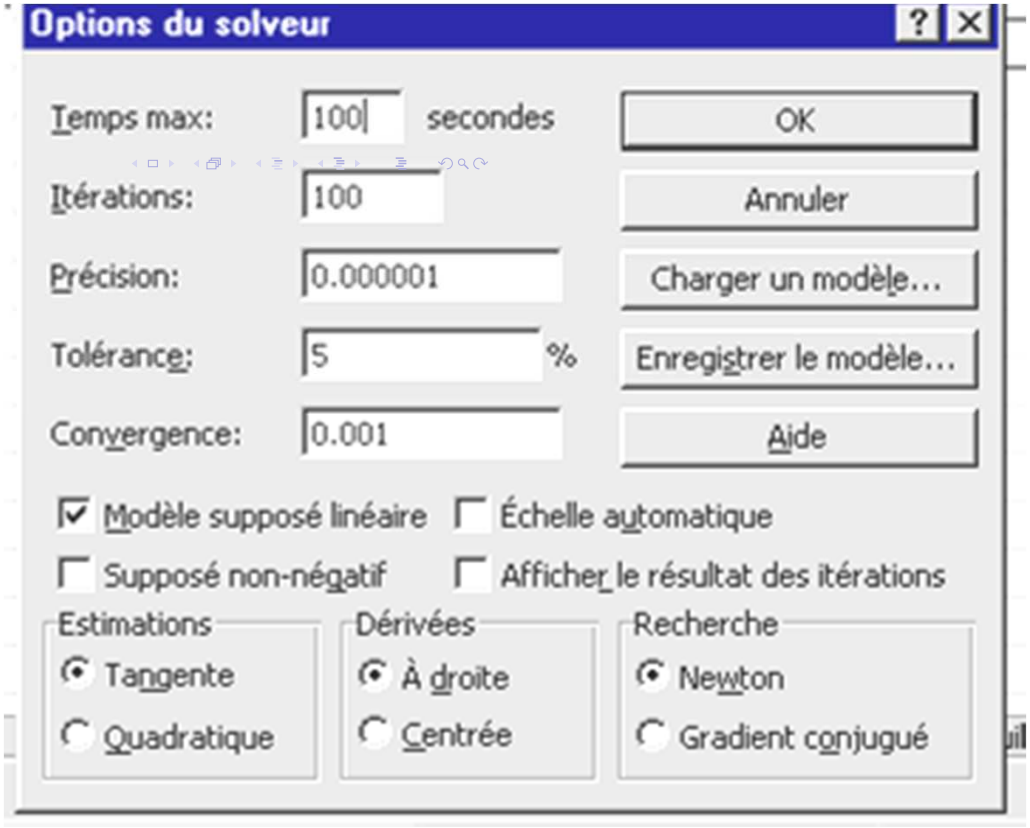
Le membre gauche de l'équation : c'est-à-dire le nombre de ressources 1100 ou son adresse F8

SOLVEUR

Résolution avec EXCEL

Cliquez sur OK lorsque vous avez terminé d'entrer toutes vos contraintes.

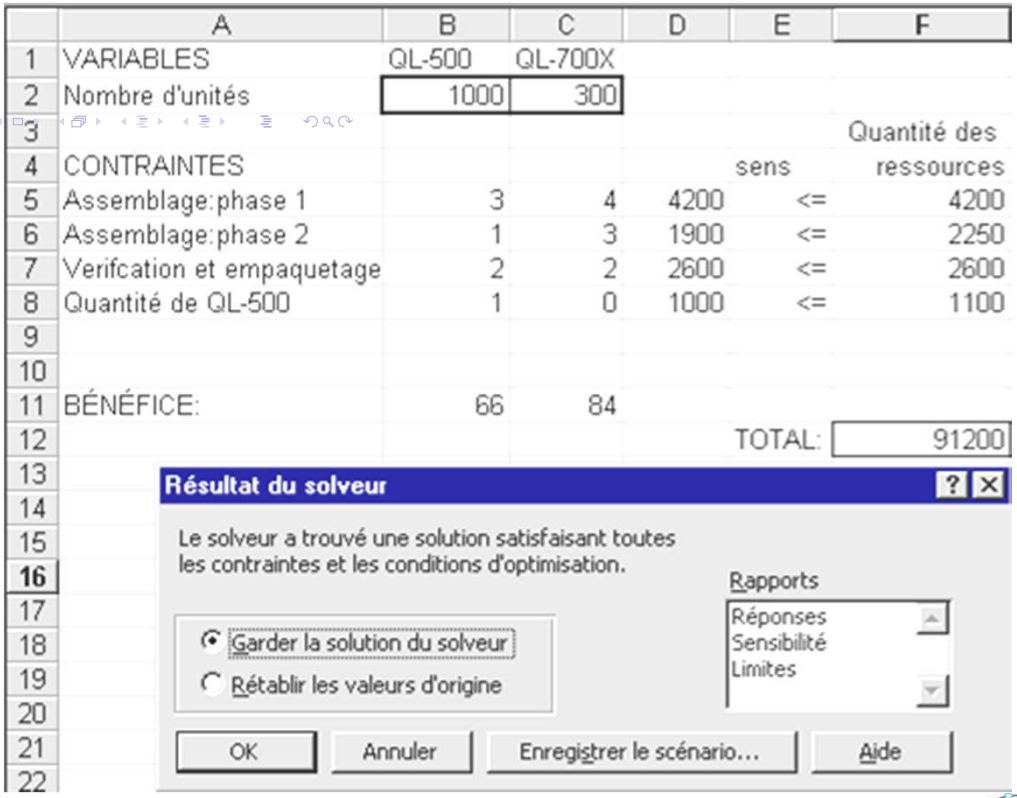
Étant donné que nous voulons résoudre un programme linéaire, il est possible de le spécifier au solveur afin qu'il utilise la méthode adéquate pour résoudre le problème. Cliquez sur "options", cochez "Modèle supposé linéaire" et cliquez sur "OK".



SOLVEUR

Résolution avec EXCEL

Cliquez sur “Résoudre”.
Le solveur a trouvé la solution optimale selon les contraintes.
Production de 1000 QL-500 et de 300 QL-700X
et un bénéfice total de 91200 $.
Le solveur vous demande si vous voulez garder cette solution à l’écran ou revenir à celle de départ. Choisissez garder la solution du solveur .
Appuyez sur “OK”.



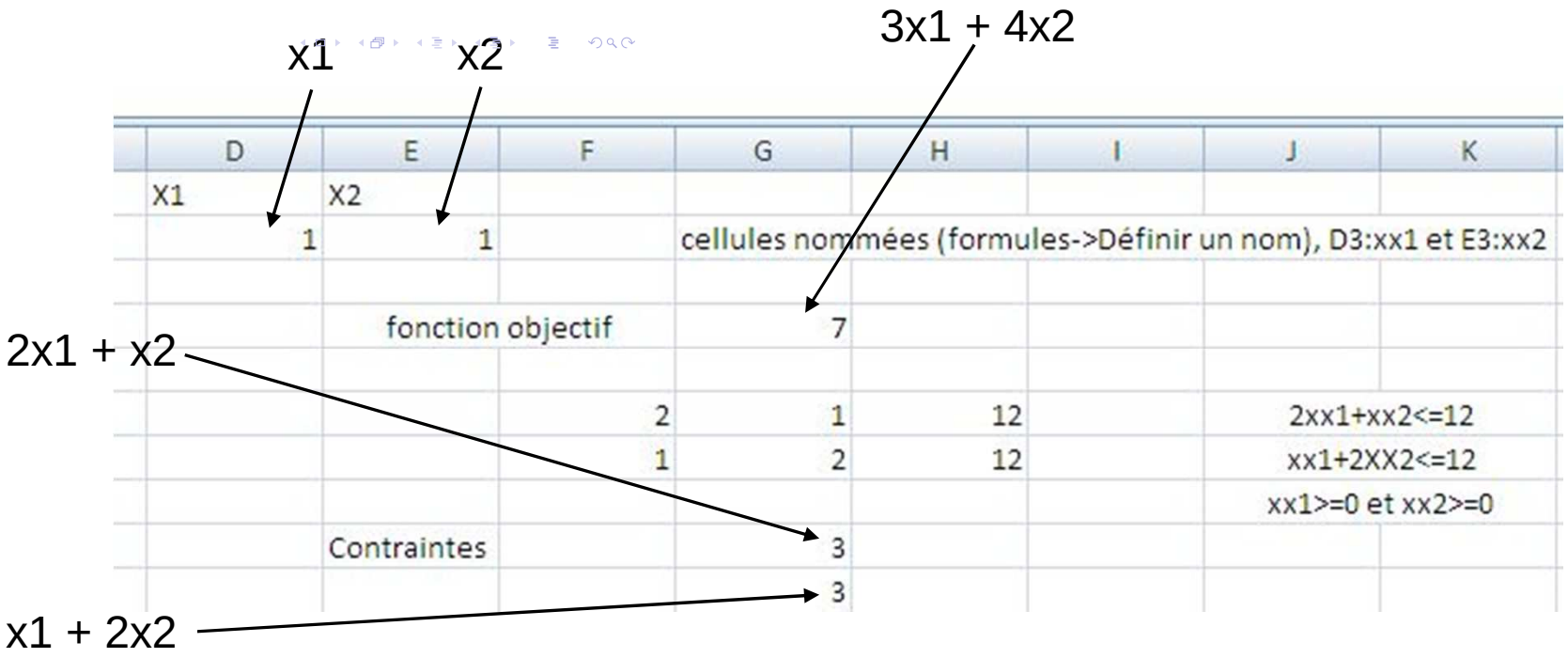
SOLVEUR

Maximiser Z = 3x1 + 4x2

sous les contraintes :

2x1 + x2<=12
x1 + 2x2<=12
et x1, x2 >=0

Résolution via le solveur



D	E	F	G	H	I	J	K
X1	X2						
1	1		cellules nommées (formules->Définir un nom), D3:xx1 et E3:xx2				
	fonction objectif		7				
		2	1	12		2xx1+xx2<=12	
		1	2	12		xx1+2XX2<=12	
						xx1>=0 et xx2>=0	
	Contraintes		3				
			3				

SOLVEUR

Maximiser Z = 3x1 + 4x2
sous les contraintes :

2x1 + x2<=12
x1 + 2x2<=12
et x1, x2 >=0

D	E	F	G	H	I	J	K	L	M
X1	X2								
1	1		cellules nommées (formules->Définir un nom), D3:xx1 et E3:xx2						
	fonction objectif		7						
		2	1	12		2xx1+xx2<=12			
		1	2	12		xx1+2XX2<=12			
						xx1>=0 et xx2>=0			
	Contraintes		3						
			3						

SOLVEUR

Maximiser Z = 3x1 + 4x2
sous les contraintes :

2x1 + x2<=12
x1 + 2x2<=12
et x1, x2 >=0

D	E	F	G	H	I	J	K	L
X1	X2							
4	4		cellules nommées (formules->Définir un nom), D3:xx1 et E3:xx2					
	fonction objectif		28					
		2	1	12		2xx1+xx2<=12		
		1	2	12		xx1+2XX2<=12		
						xx1>=0 et xx2>=0		
	Contraintes		12					
			12					

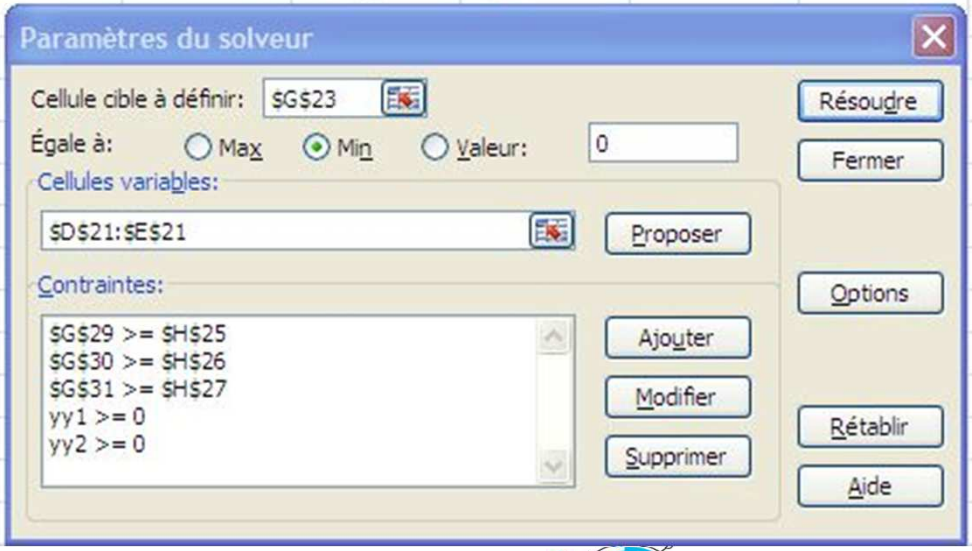
SOLVEUR Minimiser Z = 1000x1 + 1000x2

sous les contraintes :

x1 + 2x2>=12,

x1 + 4x2>=120, 6x1+3x2>=180 et x1, x2 >=0

D	E	F	G	H	I	J	K	L	M
			1000	1000					
YY1	YY2								
1	1		cellules nommées (formules->Définir un nom), D21:yy1 et E21:yy2						
	fonction objectif		2000						
						Minimiser z=1000yy1+1000yy2			
		1	2	90		yy1+2yy2>=90			
		1	4	120		yy1+4yy2>=120			
		6	3	180		6yy1+3yy2>=180			
	Contraintes					yy1>=0 et yy2>=0			
			3						
			5						
			9						



SOLVEUR Minimiser Z = 1000x1 + 1000x2
sous les contraintes :
x1 + 2x2>=12,
x1 + 4x2>=120, 6x1+3x2>=180 et x1, x2 >=0

Affichage des formules

	D	E	F	G	H
19				1000	1000
	YY1	YY2			
	1	1		cellules nommées (formu	
		fonction objectif		=G19*yy1+H19*yy2	
			1	2	90
			1	4	120
			6	3	180
		Contraintes			
				=F25*yy1+G25*yy2	
				=F26*yy1+G26*yy2	
				=F27*yy1+yy2*G27	

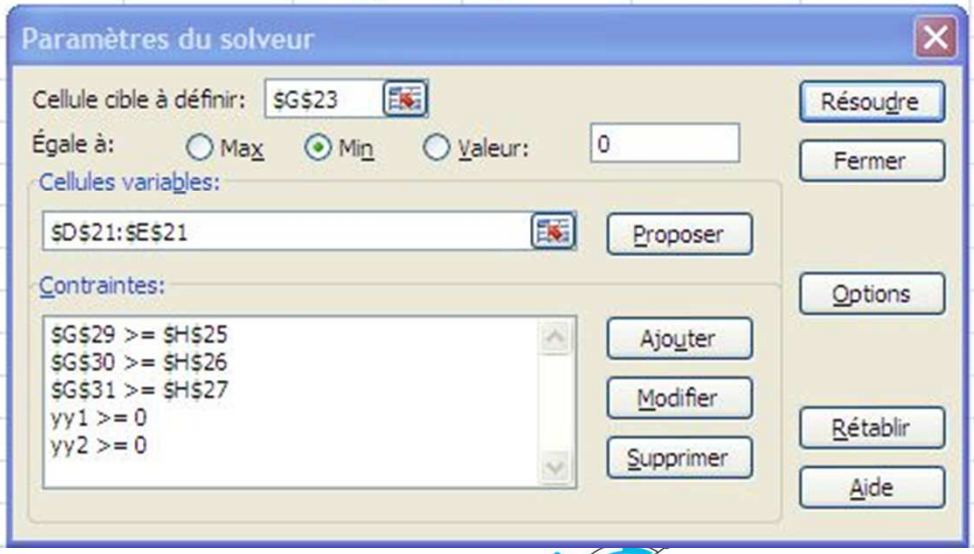
SOLVEUR Minimiser Z = 1000x1 + 1000x2

sous les contraintes :

x1 + 2x2>=12,

x1 + 4x2>=120, 6x1+3x2>=180 et x1, x2 >=0

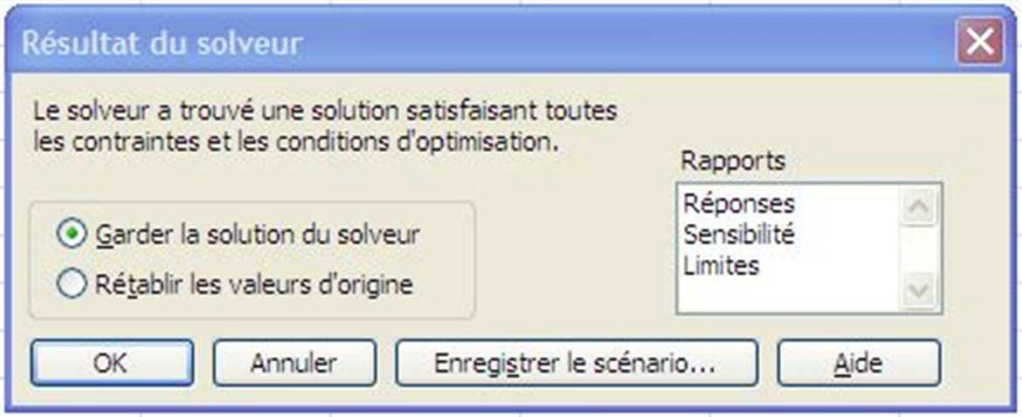
D	E	F	G	H	I	J	K	L	M
			1000	1000					
YY1	YY2								
1	1		cellules nommées (formules->Définir un nom), D21:yy1 et E21:yy2						
	fonction objectif		2000						
						Minimiser z=1000yy1+1000yy2			
		1	2	90		yy1+2yy2>=90			
		1	4	120		yy1+4yy2>=120			
		6	3	180		6yy1+3yy2>=180			
	Contraintes					yy1>=0 et yy2>=0			
			3						
			5						
			9						



SOLVEUR Minimiser Z = 1000x1 + 1000x2
sous les contraintes :

x1 + 2x2>=12,
x1 + 4x2>=120, 6x1+3x2>=180 et x1, x2 >=0

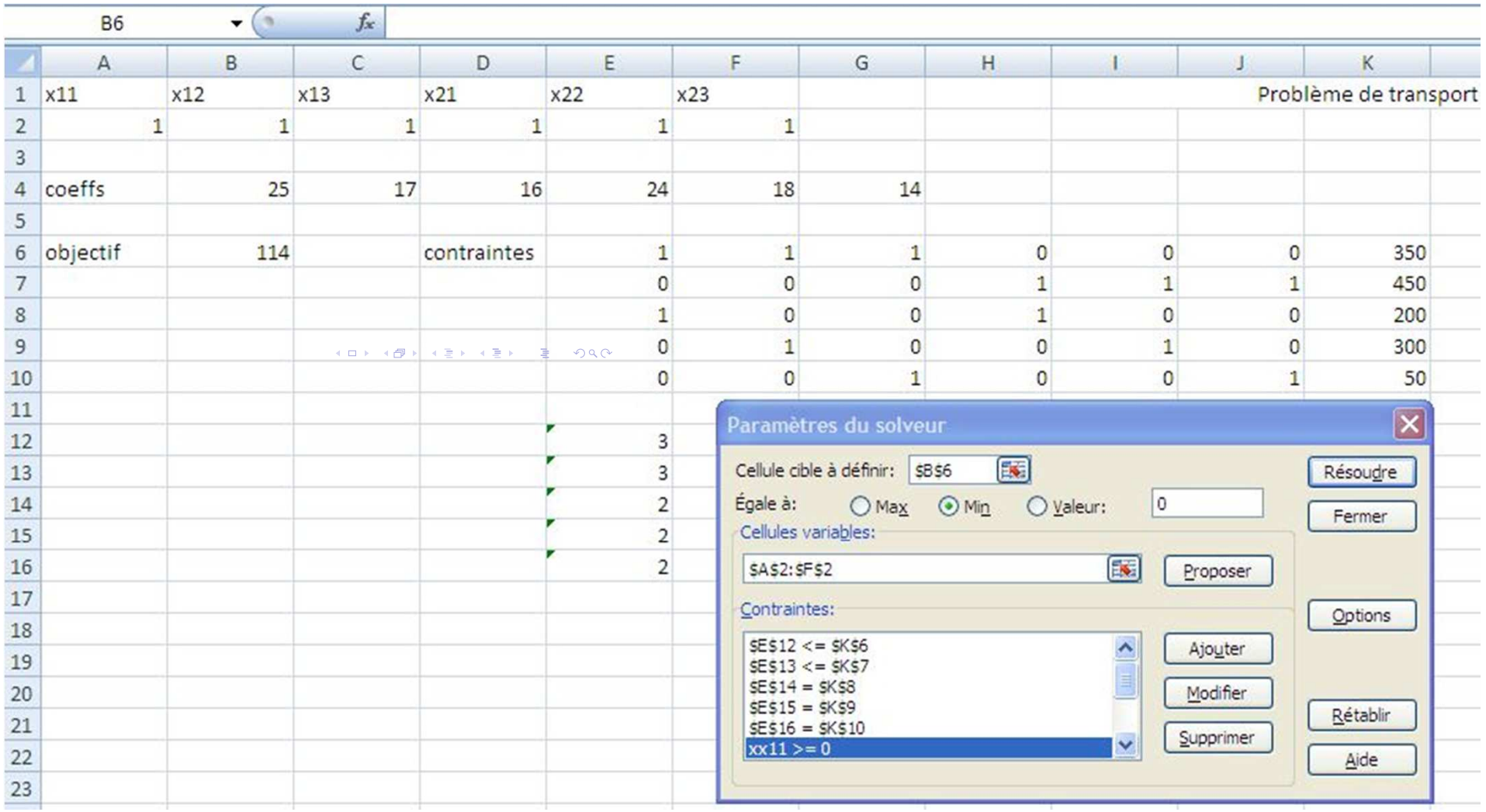
D	E	F	G	H	I	J	K	L	M
			1000	1000					
YY1	YY2								
10	40		cellules nommées (formules->Définir un nom), D21:yy1 et E21:yy2						
	fonction objectif		50000						
						Minimiser z=1000yy1+1000yy2			
		1	2	90		yy1+2yy2>=90			
		1	4	120		yy1+4yy2>=120			
		6	3	180		6yy1+3yy2>=180			
	Contraintes					yy1>=0 et yy2>=0			
			90						
			170						
			180						



SOLVEUR

	A	B	C	D	E	F	G	H	I	J	K
1	x11	x12	x13	x21	x22	x23					Problème de transport
2	1	1	1	1	1	1					
3											
4	coeffs	25	17	16	24	18	14				
5											
6	objectif	114		contraintes	1	1	1	0	0	0	350
7					0	0	0	1	1	1	450
8					1	0	0	1	0	0	200
9					0	1	0	0	1	0	300
10					0	0	1	0	0	1	50
11											
12					3						
13					3						
14					2						
15					2						
16					2						

SOLVEUR



B6

	A	B	C	D	E	F	G	H	I	J	K
1	x11	x12	x13	x21	x22	x23					Problème de transport
2	1	1	1	1	1	1					
3											
4	coeffs	25	17	16	24	18	14				
5											
6	objectif	114		contraintes	1	1	1	0	0	0	350
7					0	0	0	1	1	1	450
8					1	0	0	1	0	0	200
9					0	1	0	0	1	0	300
10					0	0	1	0	0	1	50
11											
12					3						
13					3						
14					2						
15					2						
16					2						
17											
18											
19											
20											
21											
22											
23											

Paramètres du solveur

Cellule cible à définir: $B$6

Égale à: Max Min Valeur: 0

Cellules variables:

$A$2:$F$2

Contraintes:

$E$12 <= $K$6
$E$13 <= $K$7
$E$14 = $K$8
$E$15 = $K$9
$E$16 = $K$10
xx11 >= 0

Résoudre
Fermer
Proposer
Options
Ajouter
Modifier
Supprimer
Rétablir
Aide

SOLVEUR

	A	B	C	D	E	F	G	H	I	J	K
1	x11	x12	x13	x21	x22	x23					Problème de trans
2	0	300	0	200	0	50					
3											
4	coeffs	25	17	16	24	18	14				
5											
6	objectif	10600		contraintes	1	1	1	0	0	0	350
7					0	0	0	1	1	1	450
8					1	0	0	1	0	0	200
9					0	1	0	0	1	0	300
10					0	0	1	0	0	1	50
11											
12					300						
13					250						
14					200						
15					300						
16					50						
17											
18											
19											


That's all Folks!