MASTER EN SCIENCES INFORMATIQUES

Analyse des performances d'algorithmes et application à la comparaison d'algorithmes de flot dans les réseaux de transport

JACQUET, Jean-Marie

Award date:
1984

Awarding institution:
Universite de Namur

ANALYSE DES PERFORMANCES D'ALGORITHMES ET APPLICATION A LA COMPARAISON D'ALGORITHMES DE FLOT DANS LES RESEAUX DE TRANSPORT

Mémoire présenté par

J.M. JACQUET

en vue de l'obtention du grade de
LICENCIE ET MAITRE EN INFORMATIQUE

Année académique : 1983-1984

INTRODUCTION

De tous temps et dans tous les domaines, l'Homme n'a cessé de se surpasser. A des degrés divers, selon les époques, les découvertes nouvelles, en toutes matières, ont créé un monde en mutation permanente bouleversant ce que l'on croyait définitivement établi. Les vingt-cinq dernières années, particulièrement, ont engendré des techniques et des technologies nouvelles.

Pourtant, si au cours de cette période, le matériel informatique a connu une évolution vertigineuse, il faut bien reconnaître que les performances de plus en plus grandes des ordinateurs n'ont pas toujours encouragé à bien réfléchir sur la façon dont on peut concevoir un algorithme "efficace" ni même sur le choix d'un algorithme le plus "performant" d'un ensemble d'algorithmes ayant, par ailleurs, les mêmes prétentions.

Le but de ce travail est à la fois de mettre en évidence l'importance d'une telle approche, d'exposer quelques réflexions personnelles au sujet de l'analyse des performances d'algorithme et de la construction d'algorithmes "efficaces" et d'illustrer ces propos, au travers de la comparaison de sept algorithmes de flot solutionnant le problème du flot maximum. Ce dernier nous a paru intéressant en raison de ses applications multiples : problèmes de distribution, du trafic maritime, des représentants, ...

Décrivons succinctement le contenu de ce mémoire.

Le premier chapitre présente une étude détaillée des différents outils d'analyse de performances d'algorithme. Nous définissons d'abord la notion d'algorithme et prouvons un premier résultat démontrant, à merveille, la difficulté d'une telle démarche : la non-existence

d'un algorithme capable de calculer ou d'estimer le temps d'exécution d'un programme appliqué à une donnée. Ensuite, nous exposons les avantages et inconvénients des mesures pratiques et théoriques et définissons les notions de **complexités** spatiale et temporelle en distinguant les complexités $\mathcal{O}(f)$ et $\Theta(f)$ au plus. Puis, comme il n'existe pas un seul modèle d'analyse applicable à toutes les situations, nous introduisons deux modèles couramment employés. Le premier, la machine à accès aléatoire, modélise les ordinateurs à un seul accumulateur pour lesquels les instructions de programme ne peuvent se modifier. Il en résulte trois modèles très utilisés : le modèle séquentiel, le modèle des circuits logiques et le modèle des arbres de décision. Le deuxième, la machine de Turing, simplifie à l'extrême les ordinateurs actuels mais n'en reste pas moins puissant. Il sert, en outre, de base aux développements très utiles de la théorie de la NP-complétion. Nous introduisons, en fait, deux types de machines de Turing : les machines déterministe et non-déterministe de Turing correspondant respectivement, d'une part, à l'exécution des algorithmes déterministes et à la classe de problèmes P et, d'autre part, à l'exécution des algorithmes non-déterministes et à la classe de problèmes NP. Enfin, le paragraphe I.10. tente de définir les appellations "bon algorithme", "algorithme efficace" et encore "algorithme performant".

Le chapitre II décrit, sans vouloir toutefois donner LA bonne méthode, une méthodologie de recherche d'algorithmes efficaces. Nous y énonçons d'abord quelques principes généraux qui, loin de prétendre constituer une liste exhaustive, présentent quelques points saillants qui ont retenu notre attention. Ensuite, nous montrons comment la théorie de la NP-complétion permet de guider la recherche d'algorithmes performants. Enfin, nous résumons au paragraphe II.3, en un méta-algorithme, notre méthodologie.

Au chapitre III, nous résumons les recherches que nous exposons par ailleurs en [13] et comparons sept méthodes de résolution du problème du flot maximum : méthode de Ford et Fulkerson particularisée à la recherche en profondeur d'abord de chaînes augmentantes, méthode de Ford et Fulkerson particularisée à la recherche en largeur d'abord de chaînes augmentantes, méthode d'Edmonds et Karp, méthode de Dinic, méthode de redistribution de flux, méthode des préflots ainsi qu'une méthode issue de la théorie de la programmation linéaire. A cet effet, nous précisons, tout d'abord, les notions et appellations relatives

au problème traité et définissons trois problèmes qui lui sont proches : le problème du flot b-canalisé maximum, le problème de transfert de Hitchcock et le problème du flot maximum de coût minimum. Ensuite, nous caractérisons brièvement chacune des méthodes précitées et donnons les algorithmes qui en découlent. Le tableau du paragraphe III.7. récapitule les résultats que nous avons obtenus et les tableaux III.8.1. et III.8.2. fournissent des mesures pratiques des temps d'exécution ainsi que, pour les quatre premières, du nombre d'augmentations de flot. Enfin, nous concluons ce travail en établissant, selon notre point de vue, la supériorité de la méthode de Dinic.

Les algorithmes sont écrits dans un pseudo-langage proche du Pascal qui nous semble assez suggestif. Néanmoins, le lecteur qui éprouverait des difficultés à le comprendre peut consulter l'excellent ouvrage de C. Baudoin et B. Meyer ([2]). De même, en ce qui concerne les notions élémentaires de la théorie des graphes, nous le renvoyons à [8].

Nous tenons à exprimer notre particulière reconnaissance à Monsieur le Professeur J. Fichefet, qui nous a permis de réaliser ce mémoire, pour l'aide constante et efficace qu'il nous a apportée. Nos remerciements vont aussi à ceux qui, de près ou de loin, ont collaboré à sa réalisation.

CHAPITRE I

QUELQUES OUTILS D'ANALYSE DE PERFORMANCE D'ALGORITHME

Inventer un algorithme est relativement aisé. En pratique, toutefois, on désire non seulement développer un algorithme correct, eu égard à sa spécification, mais aussi en obtenir un qui soit le plus performant possible. Dès lors, l'objectif de la programmation est bien d'inventer de "bons" algorithmes capables de résoudre un problème donné et de prouver qu'ils sont "bons".

Ce chapitre tente de préciser ce que l'on peut entendre par "algorithme performant", "bon algorithme", "meilleur algorithme" et décrit, sans vouloir cependant paraître exhaustif, les outils d'analyse de performance d'algorithmes les plus utilisés.

Cette démarche s'inscrit tout naturellement dans le cadre de la théorie d'analyse d'algorithme, développée à la suite de Knuth, dont le but est, selon Gentleman, "d'obtenir une compréhension suffisante des valeurs relatives d'algorithmes compliqués de façon à pouvoir conseiller utilement tout qui est désireux d'entreprendre un calcul". On peut, toutefois, élargir cette interprétation à l'étude de tous les aspects de performance du processus de résolution algorithmique d'un problème depuis sa formulation initiale jusqu'à l'interprétation des résultats obtenus en passant par tous les stades de développement de programmes.

Tout ceci sous-entend que l'on ait, tout d'abord, convenu de la définition d'algorithme. Nous adopterons celle ci-après, donnée par H. Leroy dans [15].

I.1. Le traitement de l'information et la notion d'algorithme

Le mot information possède au moins deux acceptions distinctes. Dans le langage courant, il désigne une affirmation au sujet de quelqu'un ou de quelque chose. Dans le jargon informatique, ce qu'on nomme information est plus souvent un nombre, une chaîne de caractères, un état imprimé, un fichier ou encore n'importe quel assemblage de symboles.

La différence essentielle entre les deux est que, dans son premier sens, l'informaion ne peut être que vraie ou fausse et ce, indépendamment de toute pensée autre que la compréhension même de l'affirmation. Ainsi, il suffit d'avoir compris ce que signifie l'affirmation qu'il pleuvra demain pour savoir du même coup, même sans rien connaître à la météorologie, que cette affirmation ne peut être que vraie ou fausse, indépendamment de la volonté et des raisonnements de quiconque. Quelle que soit l'accumulation d'observations par satellites, de théories de raisonnement et de calcul sur ordinateur qui a pu conduire les météorologistes à être convaincus et à annoncer qu'il pleuvra demain, rien de tout cela n'a le moindre pouvoir d'obliger la pluie à tomber demain si, en fait, demain il ne pleut pas. Par contre, dans son second sens, l'information n'affirme rien et ne peut être susceptible d'être vraie ou fausse. Par exemple, le nombre 427 n'affirme rien et il serait vain de se demander s'il est vrai ou faux. Seule pourrait avoir un sens, une affirmation au sujet de ce nombre.

Cette ambiguïté du mot information entraîne une ambiguïté de même nature en ce qui concerne la notion de traitement de l'information. La confusion est même encore aggravée par la place énorme que tiennent aujourd'hui les machines dans cette activité.

Pouvoir définir au juste ce qu'est le traitement de l'information revient tout d'abord à répondre à la question: à quoi sert le traitement de l'information? Car, il faut bien que cette activité que l'on peut appeler une industrie, et même une industrie lourde, ait une finalité extérieure à elle-même.

En admettant que le traitement ne puisse produire que des informations, celles-ci ne pourront manifestement servir qu'à orienter des choix, des décisions, des actions. Il faut évidemment pour cela qu'elles soient des informations dans la première acception du terme à savoir des affirmations au sujet de certaines réalités susceptibles d'être vraies ou fausses et on souhaite évidemment qu'elles soient vraies. Par exemple, si on apprend qu'il y a 427 pièces en magasin alors qu'il en faut au moins 600, il est clair qu'il va falloir faire quelque chose.

Le but de tout traitement est donc de produire des affirmations vraies selon un critère de vérité extérieur au traitement lui-même. Cependant, de tels jugements de vérité ne peuvent être tirés que d'autres jugements de vérité. Le traitement de l'information apparaît donc comme un enchaînement de jugements de vérité se déduisant les uns des autres. C'est ce qu'on appelle encore un raisonnement.

Comme on n'a pas l'éternité devant soi, notamment quand il faut prédire aujourd'hui, le temps qu'il fera demain, tous les moyens sont bons pour simplifier les raisonnements à condition, bien sûr, de ne pas en changer les conclusions. L'un d'entre eux est le calcul.

Alors qu'un raisonnement part de prémisses qui sont des jugements de vérité, en déduit, par des arguments de vérités, d'autres jugements de vérité et aboutit à des conclusions qui sont elles-mêmes des jugements de vérité, un calcul reçoit comme données des objets formels sans signification (et donc sans vérité ni fausseté), soumet ces objets à des manipulations sans aucun rapport avec la vérité ou la fausseté et produit, par conséquent, comme résultats d'autres objets formels sans signification, ni vérité, ni fausseté. Les manipulations qui constituent le calcul sont entièrement déterminées par des règles précises, formant ce qu'on appelle un algorithme, auxquelles il faut obéir sans se préoccuper de savoir pourquoi il faut obéir à ces règles plutôt qu'à d'autres. Pour cette raison, le calcul peut être mécanisé.

Le calcul permet de simplifier les raisonnements en réalisant une économie considérable d'arguments et de jugements de vérité. Ainsi, face au problème de calculer la somme des deux nombres 427 et 856, on pourrait théoriquement effectuer un raisonnement "sur mesure" qu'on ne fera qu'amorcer tellement il serait long et fastidieux.

Tout d'abord, les notations "427" et "856" sont des représentations abrégées des expressions

$$(4 \times 10 + 2) \times 10 + 7 \quad \text{et} \quad (8 \times 10 + 5) \times 10 + 6 .$$

Dès lors, résoudre le problème revient à trouver une expression sous forme standard qui ait la même valeur que l'expression

$$((4 \times 10 + 2) \times 10 + 7) + ((8 \times 10 + 5) \times 10 + 6) .$$

En faisant appel à la commutativité et à l'associativité de l'addition, on peut se convaincre (jugement de vérité) qu'elle a la même valeur que

$$(4 \times 10 + 2) \times 10 + (8 \times 10 + 5) \times 10 + (7 + 6)$$

et comme on sait (autre jugement de vérité) que l'expression $7 + 6$ a la même valeur que l'expression $10 + 3$, on est ramené à l'expression

$$(4 \times 10 + 2) \times 10 + (8 \times 10 + 5) \times 10 + 10 + 3.$$

En continuant de la sorte, on aboutirait finalement à l'expression sous forme standard

$$((1 \times 10 + 2) \times 10 + 8) \times 10 + 3$$

ou, en abrégé, "1283".

Comme chacun le sait, il est beaucoup plus économique de centraliser tous ces arguments de vérité dans la construction d'un algorithme dont l'exécution aveugle, sans recours à aucune justification, permettra d'obtenir rapidement la somme de deux nombres quelconques.

Toutefois, si le calcul permet d'éviter la répétition fastidieuse d'un grand nombre d'arguments de vérité, il ne les élimine pourtant pas totalement. De fait, pour qu'il serve à quelque chose, il faut au minimum qu'on puisse affirmer qu'en appliquant tel algorithme à "427" et "856" on a obtenu "1283" comme résultat. D'une part, cette affirmation n'est pas tautologique en ce sens qu'elle pourrait être fausse dans la réalité : on peut s'être trompé de données ou d'algorithmes ou en avoir mal appliqué les règles. D'autre part, le calcul n'a pu produire cette affirmation : il n'a rien produit d'autre que l'objet formel 1283 qui n'affirme rien du tout.

Il en est ainsi en toute généralité. L'affirmation que l'exécution de tel programme avec telles données a produit tels résultats

n'est jamais tautologique et n'est jamais produite par l'exécution du programme. C'est cette affirmation qui devra être utilisée dans la suite des traitements c'est-à-dire des raisonnements qui en tireront les conséquences utiles.

L'algorithme (ou le programme) apparaît donc comme une description précise et rigoureuse d'une suite d'opérations dont l'exécution aveugle permet d'obtenir, si elle se termine, la solution d'une classe de problèmes.

Une question se pose alors tout naturellement. Etant donné une telle classe, existe-t-il un algorithme capable de la résoudre? La théorie de la calculabilité permet d'y répondre et, par suite, offre une partition de l'ensemble des classes de problèmes en classes de problèmes calculables pour lesquelles la réponse est positive et en classes de problèmes non calculables pour lesquelles elle est négative. Notre but n'est nullement de la développer ici. Toutefois, elle apporte un premier résultat à la quête d'outils de mesure d'algorithmes : la non-existence d'algorithmes capables de calculer ou d'estimer le temps d'exécution d'un programme appliqué à certaines données.

I.2. La non-existence d'un algorithme capable de calculer ou d'estimer le temps d'exécution d'un programme

Afin de démontrer ce premier résultat, rappelons tout d'abord que la théorie de la calculabilité prouve que le problème de l'arrêt est en toute généralité insoluble algorithmiquement. En clair, cela signifie qu'il n'existe pas d'algorithme général capable de déterminer si l'exécution A(X) d'un algorithme A appliqué à une donnée X se termine ou non.

Il s'ensuit qu'il n'existe pas d'algorithme capable de calculer précisément le temps d'exécution d'un algorithme appliqué à une donnée. De fait, un tel algorithme détermine aussi, pour tout algorithme A

et toute donnée X, si l'exécution A(X) se termine : il suffit de l'appliquer à A et à X et de voir si le temps d'exécution de A à X est fini ou non.

Mais on peut encore aller plus loin et démontrer qu'il n'existe pas d'algorithme capable de déterminer, si l'exécution A(X) se termine, une borne supérieure sur ce temps d'exécution. Pour le vérifier, considérons l'exécution d'un algorithme comme l'exécution d'une succession d'opérations discrètes appelées pas de programme. Nous supposerons qu'on ait convenu d'une définition précise de cette notion. Quelqu'arbitraire qu'elle soit, la seule chose qui compte pour la suite est qu'un pas de programme soit une opération qui se termine toujours. Dans ces conditions, pour tout algorithme A et toute donnée X, désignons par $\ell(A,X)$ le nombre total de symboles qui figurent dans la donnée X et dans la représentation de A. Bien sûr, pour tout entier ℓ_o, le nombre de couples (A,X) tels que $\ell(A,X) = \ell_o$ est fini. En particulier, le nombre de couples (A,X) tels que $\ell(A,X) = \ell_o$ et tels que l'exécution A(X) de l'algorithme A à la donnée X se termine est fini. Comme chacun d'entre eux se termine après avoir exécuté un certain nombre de pas de programme, il existe un nombre $\theta(\ell_o)$ majorant ces nombres de pas. Si aucun des processus A(X) tels que $\ell(A,X) = \ell_o$ ne se termine, nous posons $\theta(\ell_o) = 0$. Posons, en outre, pour tout programme A et toute donnée X, $t(A,X) = \theta(\ell(A,X))$. Comme on le vérifie de suite, cette fonction possède la propriété que si le processus A(X) ne s'est pas terminé au bout de l'exécution de t(A,X) pas de programme, il ne se terminera jamais. Pour conclure, il reste bien entendu à prouver que la fonction t n'est pas calculable. De fait, sinon, l'algorithme suivant permettrait de déterminer, pour tout algorithme A et toute donnée X, si l'exécution A(X) se termine :

- *calculer $n = t(A,X)$ en utilisant l'algorithme de calcul de t,*
- *simuler l'exécution de A(X) en comptant le nombre de pas de programme exécutés,*
- *si n pas de programme ont été exécutés sans que l'exécution soit terminée, en conclure qu'elle ne se terminera jamais ; sinon, affirmer qu'elle se termine.*

D'où une contradiction avec le rappel précédent.

Remarquons, en outre, que la propriété de la fonction t soulignée ci-dessus reste valable pour toute fonction t' telle que, pour tout algorithme A et toute donnée X, $t'(A,X) \geq t(A,X)$. Par suite, toute fonction majorant t n'est pas calculable non plus.

Il n'existe donc pas d'algorithme capable de calculer le temps d'exécution d'un algorithme appliqué à une donnée ni même d'algorithme pour déterminer, a priori, quand il est fini, une borne sur ce temps d'exécution. Insistons ici sur le fait que ceci n'implique nullement qu'il soit impossible de déterminer une telle borne. Simplement, il n'existe pas de procédé uniforme qui en soit capable : la détermination d'une telle borne impliquera donc un raisonnement particulier à chaque situation.

I.3. Mesures théoriques et pratiques

Bien qu'une grande variété de critères puissent servir de base à l'analyse d'algorithme, les mesures du temps et de l'espace mémoire requis par son exécution retiennent l'attention de la grande majorité des informaticiens. Nous partageons ce point de vue et nous limitons, dans la suite, à ces deux seuls critères.

Néanmoins, on peut envisager les choses de deux façons différentes. D'une part, ces deux grandeurs peuvent être obtenues, de façon très pratique, par des mesures exactes de temps de calcul et de la taille mémoire nécessaires à l'exécution de l'algorithme étudié sur une machine particulière à partir de données particulières. Cette façon de procéder offre, outre l'avantage de sa simplicité de réalisation, la possibilité d'obtenir facilement des mesures moyennes ainsi que les écarts-types s'y rapportant. Toutefois, les résultats obtenus dépendent fortement de la machine particulière utilisée et par suite de ses caractéristiques intrinsèques. Dès lors, la comparaison de deux algorithmes sur deux machines différentes peut conduire à des conclusions différentes.

D'autre part, ces deux grandeurs peuvent être évaluées, plus théoriquement, par l'analyse même de l'algorithme. Les mesures ainsi

obtenues sont moins précises mais permettent toutefois, généralement une meilleure compréhension des coûts temporel et spatial engendrés par l'exécution de l'algorithme analysé, notamment de son évolution avec la taille du problème.

Comme l'implémentation de la première technique est immédiate, nous restreindrons dans la suite notre attention aux différentes approches de la seconde.

I.4. Complexité des algorithmes

La notion de complexité des algorithmes permet de donner un sens précis à l'idée intuitive selon laquelle un algorithme est caractérisé par un coût. Nous l'avons déjà dit, par coût, on entend généralement, le temps d'exécution et la quantité de mémoire requise par l'exécution de l'algorithme traité. Il leur correspond respectivement les notions de complexité temporelle et de complexité spatiale. Nous ne développerons toutefois dans ce chapitre que les notions relatives au temps d'exécution, celles relatives à l'emplacement mémoire étant obtenues, mutatis mutandis, à partir des premières.

Il convient ici encore de distinguer les notions de complexité temporelle pratique et de complexité temporelle théorique. La première consiste en une mesure plus précise des temps de calcul et des tailles de mémoire pour un modèle de machine bien défini alors que la seconde donne des ordres de grandeurs des coûts de façon plus indépendante des conditions pratiques de l'exécution de l'algorithme.

Introduisons ces notions à l'aide d'un exemple simple. Pour ce faire, proposons-nous d'écrire un programme déterminant pour tout nombre entier supérieur à 1 son plus grand diviseur autre que lui-même. L'algorithme suivant convient. Le nombre n y représente l'entier à traiter et l'entier *pgd* fournit, après exécution, le diviseur cherché.

ALGORITHME I.4.1.

```
|pgd := n - 1;
|Tant que (n mod i) ≠ 0 répéter
| |pgd := pgd - 1.
```

Un autre algorithme possible se base sur la remarque que le résultat cherché s'écrit sous la forme n/m où m est le plus petit diviseur de n supérieur à 1. On obtient alors

ALGORITHME I.4.2.

```
|m := 2;
|Tant que m < √n et (n mod m ≠ 0) répéter
| |m := m+1;
|Si n mod m = 0
| |alors pgd := n/m
| |sinon pgd := 1.
```

Pour comparer les temps d'exécution de ces deux algorithmes, écrivons le premier sous la forme

```
|i1;
|Tant que c1 répéter
| |i2
```

et le second sous la forme

```
|i3;
|Tant que c2 répéter
| |i4;
|Si c3
| |alors i5
| |sinon i6
```

et désignons respectivement par $t(c_1)$, $t(c_2)$, $t(c_3)$, le temps d'exécution des conditions c_1, c_2, c_3 (en comptant le cas échéant, le temps nécessaire aux branchements sous-jacents) et par $t(i_j)$ le temps nécessaire à l'exécution de l'instruction i_j ($j = 1,\ldots,6$). Le temps d'exécution du premier algorithme vaut donc

$$T_1 = t(i_1) + (\alpha+1)t(c_1) + \alpha t(i_2)$$

et le temps d'exécution du second, au plus,

$$T_2 = t(i_3) + (\beta + 1)t(c_2) + \beta t(i_4) + t(c_3) + \max\{ t(i_5), t(i_6) \}$$

où α et β représentent le nombre d'itérations des boucles répétitives *Tant que ...* dans le premier et second algorithme. On vérifie de suite que l'on a

$$\alpha = n - 2 \quad \text{et} \quad \beta = \lceil \sqrt{n} \rceil - 2$$

où $\lceil \sqrt{n} \rceil$ désigne l'entier immédiatement supérieur à $\sqrt{n}$ et, par suite, en introduisant les constantes appropriées a, a', b, b'

$$T_1 = a + bn$$

$$T_2 = a' + b'\sqrt{n}$$

Nous dirons que la complexité temporelle de l'algorithme I.4.1. est $\mathcal{O}(n)$ au plus, alors que celle du second est $\mathcal{O}(\sqrt{n})$ au plus.

Détailler la complexité temporelle pratique de ces deux versions conduirait à calculer les constantes a, a', b, b' propres à une machine donnée et dépendant des temps des instructions de base. La complexité temporelle théorique semble beaucoup plus intéressante ici. En effet, elle montre que plus n est grand, plus les termes bn et $b'\sqrt{n}$ sont dominants. Par suite, pour n grand, de l'ordre de 10^{10}, par exemple, le premier prendrait un temps prohibitif sur les machines actuelles en comparaison du second.

Plus généralement, étant donné un algorithme A, il existe presque toujours un paramètre, soit n, caractérisant la taille des données auxquelles A s'applique dans chaque cas. Soit donc T(n) le temps d'exécution de l'algorithme A exprimé en fonction de n, soit en outre f une fonction du paramètre n.

L'algorithme A est de <u>complexité temporelle théorique</u> $\mathcal{O}(f(n))$ <u>au plus</u> s'il existe une constante C et une valeur n_0 du paramètre n telles que, $T(n) \leqslant Cf(n)$, pour tout $n \geqslant n_0$. Il est de <u>complexité temporelle théorique</u> $\mathcal{O}(f(n))$ si la limite

$$\lim_{n \to +\infty} \frac{T(n)}{f(n)}$$

existe, est finie et diffère de 0.

Comme on le vérifie de suite, la complexité temporelle $\mathcal{O}(f(n))$ implique la complexité temporelle théorique $\mathcal{O}(f(n))$ au plus.

Toutefois, l'assertion inverse n'est pas vérifiée.

Nous avons cru bon de mettre en évidence cette différence. L'interprétation en est aisée. La complexité temporelle théorique $\mathcal{O}(f(n))$ implique que pour ε arbitrairement petit et pour n suffisamment grand, le temps d'exécution T(n) ne diffère pas de

$$\left(\lim_{n \to +\infty} \frac{T(n)}{f(n)} \right) f(n)$$

de plus de ε. Ainsi l'algorithme suivant de multiplication de deux motrices A, B de dimension n x n

```
Pour i variant de 1 à n effectuer
  Pour j variant de 1 à n effectuer
    somme := 0;
    Pour k variant de 1 à n effectuer
      somme := somme + a(i,k)xb(k,j);
    c(i,j) := somme
```

est de complexité temporelle théorique $\mathcal{O}(n^3)$. Quelle que soit la façon dont on raisonne, on n'arrivera pas à démontrer que le terme dominant de T(n) diffère de n^3, si ce n'est pas une constante multiplicative. En outre, dans tous les cas de figure de données, le temps d'exécution restera proportionnel à n^3. Par contre, la complexité temporelle théorique $\mathcal{O}(f(n))$ au plus implique seulement l'existence d'une constante C et d'une valeur n_0 de n telles que

$$T(n) \leqslant cf(n), \quad \forall n \geqslant n_0$$

Dans ce cas, rien ne s'oppose à la découverte d'une borne supérieure plus stricte et, par suite, à l'établissement d'une complexité temporelle théorique meilleure. Ceci reflète essentiellement deux situations. Dans la première, les connaissances actuelles n'ont pu permettre l'établissement d'une inégalité plus stricte et rien ne s'oppose à ce qu'on y arrive dans le futur. Dans la deuxième, l'inégalité correspond aux comportements de l'algorithme dans les cas les plus défavorables. Dans cette situation, il est fort probable qu'il s'améliore dans certains cas de données plus favorables. Ainsi en est-il de certains algorithmes de tri dont le temps d'exécution est considérablement décru si les données entrées sont partiellement triées à

l'origine. Ainsi en est-il aussi, comme nous l'établissons en annexe (cfr. [13]), de l'algorithme de Dinic dont les performances sont nettement accrues si on l'applique à des réseaux de transport de capacités unitaires.

On peut tenter d'appréhender cette situation en introduisant la notion de complexité temporelle moyenne théorique définie de même que la notion de complexité temporelle théorique si ce n'est qu'on y remplace $T(n)$ par le temps d'exécution moyen de l'algorithme. Ceci n'a toutefois de sens que si l'on peut faire des hypothèses probabilistes sur la répartition des données, ce qui est loin d'être toujours le cas.

Notons, enfin, que l'expérience enseigne qu'on peut généralement associer à tout algorithme une situation particulièrement défavorable conduisant l'algorithme à des performances catastrophiques. A notre avis, la détection d'un tel cas ne doit pas automatiquement entraîner le rejet de l'algorithme ; une telle attitude ne pouvant, selon nous, résulter que d'une analyse suffisamment poussée de ces cas pathologiques.

Ces définitions se généralisent aisément au cas où le temps d'exécution dépend de plusieurs paramètres. Soient donc, $T(n_1,\ldots, n_m)$ le temps d'exécution de l'algorithme A et $f(n_1,\ldots, n_m)$ une fonction des m paramètres $n_1,\ldots, n_m$.

L'algorithme A est de <u>complexité temporelle théorique</u> $\mathcal{O}(f(n_1,\ldots, n_m)$ <u>au plus</u> si, pour tout $i\in\{1,\ldots, m\}$, il existe une constante $C(n_1,\ldots, n_{i-1}, n_{i+1},\ldots, n_m)$-éventuellement fonction de ces paramètres sauf de n_i- et une valeur $n_{i,o}$ du paramètre n_i telle que

$$T(n_1,\ldots, n_m) \leqslant C(n_1,\ldots,n_{i-1}, n_{i+1},\ldots n_m)\ f(n_1,\ldots, n_m)$$

pour tout $n_i \geqslant n_{i,0}$. <u>Il est de complexité temporelle théorique</u> $\mathcal{O}(f(n_1,\ldots,n_m))$ si, pour tout $i\in\{1, \ldots ,m\}$, la limite

$$\lim_{n_i\to+\infty} \frac{T(n_1, \ldots ,n_m)}{f(n_1, \ldots ,n_m)}$$

existe, est finie et diffère de 0.

Bien sûr, au cas où le temps d'exécution dépend de plusieurs paramètres, les deux premières notions restent valables. Ainsi, l'algorithme A peut-il être de complexité temporelle théorique $\mathcal{O}(f(n_1))$ et $\mathcal{O}(g(n_2))$. Il convient ici d'attirer l'attention sur le fait suivant : un algorithme peut très bien avoir une bonne complexité en terme d'un paramètre et une médiocre en terme d'un autre. Le choix du paramètre adéquat est donc primordial. Nous ne connaissons malheureusement aucune théorie guidant ce choix.

Cette approche théorique peut être critiquée parce qu'elle néglige les constantes de proportionalité pour ne retenir que les comportements asymptotiques. Ainsi, étant donné deux algorithmes A_1 et A_2 résolvant un même problème et de temps d'exécution respectifs $T_1(n)$ et $T_2(n)$, il pourrait sembler à première vue que

- les deux algorithmes sont aussi efficaces l'un que l'autre parce qu'ils sont de même complexité temporelle théorique, par exemple, $\mathcal{O}(n^2)$, alors qu'un examen plus approfondi révèle que $T_1(n) = 100\ n^2 + 2000\ n + 100$ et $T_2(n) = 300\ n^2 + 1000\ n + 900$ et donc que l'égalité $T_1(n) = T_2(n)$ n'est vérifiée que pour les valeurs 1 et 4 de n
- l'algorithme A_1 est plus efficace que l'algorithme A_2 parce que, par exemple, le premier est de complexité temporelle théorique $\mathcal{O}(n)$ et le second de complexité temporelle théorique $\mathcal{O}(n^2)$ alors qu'un examen plus approfondi montre que $T_1(n)$ est strictement supérieur à $T_2(n)$ pour toutes les valeurs de n considérées dans la pratique. C'est notamment le cas si $T_1(n) = 2000\ n$ et $T_2(n) = n^2 + n$ pour lequel on a $0 < n < 1999 \Longrightarrow T_2(n) < T_1(n)$.

Il ne manque cependant pas d'arguments pour justifier l'emploi de la notion de complexité théorique. D'une part, contrairement aux mesures pratiques et à la complexité pratique, elle donne des ordres de grandeur des coûts d'exécution des algorithmes sans s'attacher à un modèle particulier d'ordinateur et sans obliger à exécuter réellement ces algorithmes sur un ordinateur particulier. Or, comme l'illustrent les deux tables suivantes, ces ordres de grandeur sont particulièrement intéressants.

Complexité Temporelle	Taille du problème n			
	10	10^2	10^3	10^4
$\log_2 n$	3,3	6,6	10	13,3
n	10	10^2	10^3	10^4
$n \log_2 n$	$0,33 \times 10^2$	$0,7 \times 10^3$	10^4	$1,3 \times 10^5$
n^2	10^2	10^4	10^6	10^9
n^3	10^3	10^6	10^9	10^{12}
$2n$	1024	$1,3 \times 10^{30}$	10^{100}	10^{100}

TABLE I.4.1.

Complexité Temporelle	Taille maximum du problème pour une exécution en		
	1 sec	1 min	1 heure
$\log_2 n$	2^{10^6}	$2^{6 \times 10^7}$	$2^{3,6 \times 10^9}$
n	10^6	6×10^7	$3,6 \times 10^9$
$n \log_2 n$	62746	$2,8 \times 10^6$	$1,3 \times 10^8$
n^2	10^3	7746	60000
n^3	10^2	$3,9 \times 10^2$	$1,5 \times 10^3$
2^n	20	25	32

TABLE I.4.2.

La première illustre la croissance des différentes fonctions de complexité en fonction de la taille du problème. La seconde fournit les tailles maximum des problèmes pouvant être résolus par les algorithmes de complexité correspondante en une limite de temps spécifiée. Nous supposons qu'une opération s'effectue en 10^{-6} sec. On remarque, dans les deux cas, que le comportement des algorithmes de complexité logarithmique est le meilleur alors que le comportement des algorithmes de complexité exponentielle est le moins performant.

D'autre part, elle constitue une notion fondamentale si l'on désire tenir compte des progrès technologiques. Le tableau ci-dessous indique les tailles maximales des données que permettraient de traiter, dans le même temps, les algorithmes A_1 à A_6 sur une nouvelle génération d'ordinateurs sous les hypothèses que celle-ci est 10, 100 ou 1000 fois plus rapide que celle correspondant à la troisième colonne.

Algorithme	Complexité Temporelle	Taille Maximale	Génération 10 fois plus rapide	Génération 100 fois plus rapide	Génération 1000 fois plus rapide
A_1	$\log_2 n$	n_1	n_1^{10}	n_1^{100}	n_1^{1000}
A_2	n	n_2	$10\ n_1$	$100\ n_1$	$1000\ n_1$
A_3	$n \log_2 n$	n_3	$\simeq 10 n_3$ pour $n_3 >>$	$\simeq 100\ n_3$ pour $n_3 >>$	$\simeq 1000\ n_3$ pour $n_3 >>$
A_4	n^2	n_4	$\sqrt{10}\ n_4$	$10\ n^4$	$\sqrt{100}\ n^4$
A_5	n^3	n_5	$\sqrt[3]{10}\ n_5$	$\sqrt[3]{100}\ n_5$	$10\ n^5$
A_6	2^n	n_6	$n_6 + \log_2 10$	$n_6 + \log_2 100$	$n_6 + \log_2 1000$

TABLE I.4.3.

On remarque à nouveau que l'algorithme A_1 (respectivement A_6) de complexité temporelle théorique $\mathcal{O}(\log_2 n)$ (resp. $\mathcal{O}(2^n)$) assure l'augmentation maximum (resp. minimum) de taille.

Ceci tend encore à prouver le fait suivant. Si la recherche d'un algorithme efficace destiné à être exécuté sur des machines lentes semble primordiale, contrairement à ce qu'on pourrait croire, les progrès technologiques la rendent encore plus importante, notamment du point de vue de la complexité temporelle théorique, les coefficients intervenant dans la complexité pratique devenant au contraire moins fondamentaux.

Enfin, elle est à la base de la théorie très intéressante de la NP-complétion que nous esquissons au chapitre II.

Toutefois, afin de l'exposer et de poursuivre notre dissertation sur les performances d'algorithmes, il convient de spécifier un modèle théorique de machine. Malheureusement, il n'existe pas un seul modèle applicable à toutes les situations. Une des difficultés principales consiste en la taille des mots des ordinateurs. Par exemple, si l'on accepte qu'un tel mot puisse contenir un entier de taille arbitrairement grande, le problème entier peut être encodé en un seul entier dans un seul mot. Par contre, si les mots sont supposés finis, il est nécessaire de considérer le problème de stockage des entiers arbitrairement grand au même titre que les autres, ce qui est souvent oublié lorsqu'on traite des problèmes de faible taille.

Pour chaque type de problème, nous devons donc sélectionner un modèle approprié qui reflètera autant que possible l'exécution des algorithmes sur les ordinateurs réels. Dans les paragraphes suivants, nous décrivons deux de ces modèles. Le premier appelé machine d'accès aléatoire fournit un modèle général duquel en découlent trois autres particulièrement intéressants. Le second, la machine de Turing servira de base aux développements de la théorie de la NP-complétion.

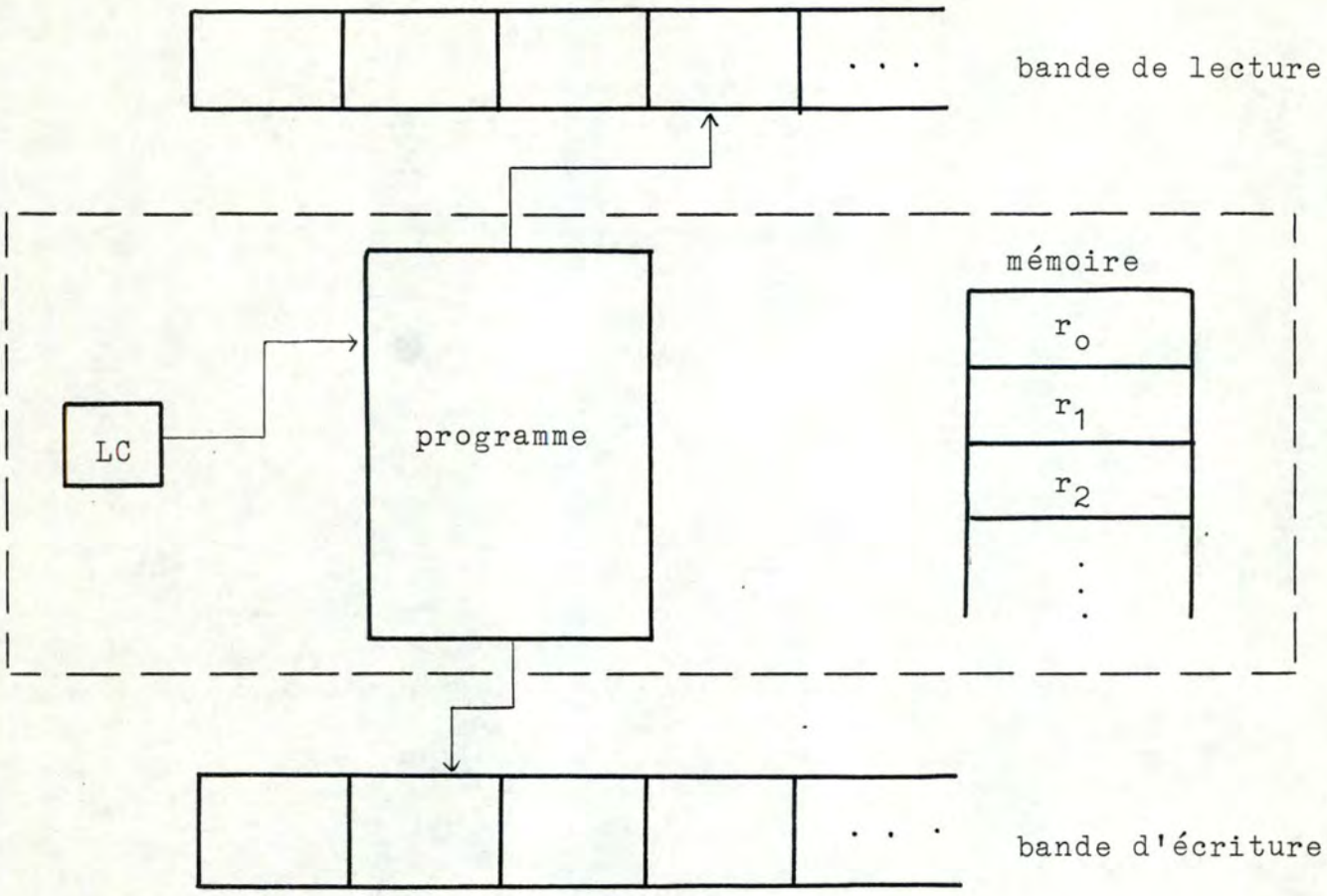
I.5. La machine à accès aléatoire

La machine à accès aléatoire modélise les ordinateurs à un seul accumulateur pour lesquels les instructions de programme ne peuvent se modifier.

Elle consiste en

- une bande de lecture formée d'une suite infinie (vers la droite) de carrés dont chacun contient soit le symbole blanc, soit un nombre entier éventuellement négatif. Il lui est associé une tête de lecture qui, après la lecture du contenu d'un carré, se déplace automatiquement d'un carré vers la droite.
- une bande d'écriture décomposée, de même, en une infinité de carrés contenant tous initialement le symbole blanc. Après l'exécution d'une instruction d'écriture, un entier est imprimé dans la case située en dessous de la tête d'écriture qui lui est associée, puis cette tête se déplace d'un carré vers la droite. Une fois qu'un symbole a été écrit, il ne peut être changé.
- un programme composé d'instructions auxquelles la machine accède grâce à un compteur noté LC (pour Location Counter).
- une mémoire constituée d'une suite (r_i) éventuellement infinie de registres capables de mémoriser un entier de taille arbitrairement grande. Le premier r_o, appelé accumulateur, joue un rôle particulier : tous les calculs s'y effectuent.

La figure suivante la représente.



Comme le programme n'est pas stocké en mémoire, il ne peut se modifer. Nous supposerons qu'il consiste en une simple suite d'instructions dont la nature exacte importe peu pour autant qu'elles ressemblent à des instructions habituellement utilisées sur ordinateur réel. Ainsi, nous pouvons toujours supposer l'existence d'instructions arithmétiques, d'entrée-sortie, d'adressage indirect et de branchement. La liste I.5.1 donne un exemple d'un tel ensemble d'instructions dont le format est toujours constitué de deux parties

- le code opératoire
- l'adresse, elle-même, quand elle est non vide, composée d'une opérande ou d'une étiquette.

Chaque opérande est nécessairement l'une des trois formes suivantes

1) = i, indiquant l'entier i lui-même
2) i, (où i est positif) indiquant le contenu du registre i

3) *i, indiquant l'adressage indirect, (l'opérande est, dans ce cas, le contenu du registre r_j, j étant l'entier représenté dans le registre r_i. Par convention, la machine s'arrête si j est strictement négatif).

	Code opératoire	Adresse
1.	LOAD	Opérande
2.	STORE	Opérande
3.	ADD	Opérande
4.	SUB	Opérande
5.	MULT	Opérande
6.	DIV	Opérande
7.	READ	Opérande
8.	WRITE	Opérande
9.	JUMP	Etiquette
10.	JGTZ	Etiquette
11.	JZERO	Etiquette
12.	HALT	

LISTE I.5.1.

Ces instructions sont familières à tout qui a programmé en langage ASSEMBLER. Comme dit ci-dessus, on peut, bien sûr, en adjoindre d'autres pour autant qu'elles restent des instructions de base d'ordinateur.

Décrivons à présent l'exécution d'un programme quelconque formé des instructions de la liste I.5.1. Pour ce faire, désignons pour tout $i \in \mathbb{N}$, par c(i) l'entier mémorisé dans le registre r_i et définissons la valeur v(a) de l'opérande a par les relations

$v(=i) = i$

$v(i) = c(i)$

$v(*i) = c(c(i))$.

Initialement, on a, pour tout $i \in \mathbb{N}$, $c(i) = 0$. En outre, le compteur LC pointe vers la première instruction du programme à exécuter et les têtes de lecture et d'écriture vers le premier carré de leur bande respective.

L'exécution du programme se déroule alors conformément au tableau I.5.2.. Les instructions non définies telles que STORE = i sont considérées comme équivalentes à l'instruction HALT. De même, toute division par zéro arrête la machine.

INSTRUCTION		SIGNIFICATION
1.	LOAD a	c(o) := v(a) ; incrémenter LC d'une unité
2.	STORE i	c(i) := c(o) ; incrémenter LC d'une unité
	STORE *i	c(c(i)) := c(o) ; incrémenter LC d'une unité
3.	ADD a	c(o) := c(o) + v(a) ; incrémenter LC d'une unité
4.	SUB a	c(o) := c(o) - v(a) ; incrémenter LC d'une unité
5.	MULT a	c(o) := c(o) * v(a) ; incrémenter LC d'une unité
6.	DIV a	c(o) := c(o) div v(a) ; incrémenter LC d'une unité
7.	READ i	c(i) := symbole se trouvant sous la tête de lecture ; décaler cette tête d'une case à droite ; incrémenter LC d'une unité
	READ *i	c(c(i)) := symbole se trouvant sous la tête de lecture ; décaler cette tête d'une case à droite ; incrémenter LC d'une unité
8.	WRITE a	imprimer v(a) dans la case située en dessous de la tête d'écriture ; décaler cette tête d'une case à droite et incrémenter LC d'une unité
9.	JUMP b	positionner le compteur LC sur l'instruction libellée par b
10.	JGTZ b	positionner le compteur LC sur l'instruction libellée par b si c(0) > 0, sinon, incrémenter ce compteur d'une unité
11.	JZERO b	positionner le compteur LC sur l'instruction libellée par b si c(0) = 0, sinon incrémenter ce compteur d'une unité
12.	HALT	arrêter l'exécution

Tableau I.5.2.

Comme on le constate de suite, l'exécution des huit premières instructions a pour effet d'incrémenter le compteur LC d'une unité. Par suite, les instructions de tout programme sont exécutées séquentiellement jusqu'à la rencontre d'une des instructions JUMP ou HALT, de l'instruction JGTZ quand le contenu de l'accumulateur est plus grand que 0 ou encore de l'instruction JZERO quand ce contenu est nul.

Afin d'être complet, remarquons que tout programme définit une application de l'ensemble des contenus possibles de la bande de lecture dans l'ensemble des contenus possibles de la bande d'écriture. Comme l'exécution d'un programme peut ne pas se terminer, cette application est partielle. De plus, elle est sujette aux deux interprétations suivantes. Si le programme P a pour spécification de lire les n entiers $x_1, \ldots, x_n$ figurant sur la bande de lecture et d'écrire un seul entier y, on peut affirmer qu'elle calcule la fonction $f(x_1, \ldots, x_n) = y$. Par ailleurs, si sa spécification est de lire une chaîne sur la bande de lecture et de produire sur la bande d'écriture l'un des deux résultats 0 ou 1, elle constitue un accepteur de langage, le langage accepté étant, selon les conventions, le langage constitué des chaînes pour lesquelles P produit 0 ou 1.

En guise d'exemple, proposons-nous d'écrire un programme dont la spécification est de calculer la fonction définie sur $\mathbb{Z}$ par les relations

$$f(n) = \begin{cases} n^n & \text{si } n \geq 1 \\ 0 & \text{sinon} \end{cases}$$

L'algorithme suivant convient. Le symbole n représente l'entier en lequel on veut calculer la valeur de f et le nombre res fournit, après exécution, cette valeur.

```
x := n;
si x > 0
  alors
    res := 0
  sinon
    x_aux := x;
    cpteur := x - 1;
    tant que cpteur > 0 effectuer
      x_aux := x_aux * x ;
      cpteur := cpteur - 1;
    res := x_aux.
```

Le programme ci-dessous lui correspond. Les variables *x*, *x_aux* et *cpteur* sont respectivement mémorisées dans les registres r_1, r_2 et r_3. Afin de rendre évidente cette correspondance, nous avons négligé certaines optimisations.

programme	algorithme
READ 1	*x := n*
LOAD 1 JGTZ pas WRITE =0	*si x > 0* *alors* *res := 0*
JUMP endif	
pas : LOAD 1 STORE 2	*sinon* *x_aux := x*
LOAD 1 SUB =1 STORE 3	*cpteur := x - 1*
while : LOAD 3 JGTZ continue JUMP endwhile	*tant que cpteur > 0* *effectuer*
continue : LOAD 2 MULT 1 STORE 2	*x_aux := x_aux * x*
LOAD 3 SUB =1 STORE 3	*cpteur := cpteur - 1*
JUMP while	
endwhile : WRITE 2	*res := x_aux*
endif : HALT	

Déterminons, à présent, les coûts occasionnés par l'exécution d'un programme. A nouveau, nous ne considérerons comme coûts que le temps et l'espace nécessaires à cette exécution.

Nous définissons la première comme la somme des temps d'exécution des instructions qui composent le programme. Dès lors, il reste pour la calculer à définir le temps requis par l'exécution des instructions de la liste I.5.1.. Deux coûts peuvent être retenus. Le premier, appelé coût uniforme, impose que chaque instruction s'exécute en une unité de temps. Le second, le coût logarithmique, tient compte de ce que le temps d'exécution d'une instruction est proportionnel à la longueur de son opérande. Pour ce faire, au décodage de chaque opérande est associé le coût défini par la table ci-dessous.

opérande	coût
=i	$\ell(i)$
i	$\ell(i) + \ell(c(i))$
*i	$\ell(i) + \ell(c(i)) + \ell(c(c(i)))$

où, pour tout entier i, $\ell(i)$ vaut

$$\ell(i) = \begin{cases} \lceil \log |i| \rceil + 1 & \text{si } i \neq 0 \\ 1 & \text{si } i = 0, \end{cases}$$

$\lceil x \rceil$ désignant le plus petit entier supérieur ou égal à x. Le temps d'exécution de chaque instruction de la liste I.5.1. est alors défini par le tableau suivant où t(a) est le coût de l'opérande a.

instruction	coût
LOAD a	t(a)
STORE i	$\ell(c(0)) + \ell(i)$
STORE *i	$\ell(c(0)) + \ell(i) + \ell(c(i))$
ADD a	$\ell(c(0)) + t(a)$
SUB a	$\ell(c(0)) + t(a)$
MULT a	$\ell(c(0)) + t(a)$
DIV a	$\ell(c(0)) + t(a)$
READ i	ℓ(entrée) + $\ell(i)$

READ *i	ℓ(entrée) + ℓ(i) + ℓ(c(i))
WRITE a	t(a)
JUMP b	1
JGTZ b	ℓ(c(0))
JZERO b	ℓ(c(0))
HALT	1

L'espace mémoire nécessaire à l'exécution d'un programme fait aussi l'objet d'une mesure uniforme ou logarithmique. Dans la première optique, cet espace équivaut à la somme des espaces des registres auxquels accède le programme, chaque registre requérant une unité d'espace. Dans la seconde, il reste égal à la somme des espaces des registres auxquels accède le programme mais chaque registre r_i requiert cette fois l'espace $\ell(x_i)$ où x_i est l'élément de $\mathbb{Z}$ le plus grand en valeur absolue stocké durant l'exécution dans r_i.

Cela étant, les complexités temporelles et spatiales se définissent de même que ci-avant.

Il va de soi qu'un programme peut avoir une complexité temporelle ou spatiale totalement différente selon que l'on retient pour la calculer les coûts uniformes ou logarithmiques. La mesure à considérer dépend des situations. S'il est raisonnable de supposer que chaque nombre rencontré pendant l'exécution du programme peut être stocké dans un mot d'ordinateur, les premiers coûts cités doivent être considérés. Dans le cas contraire, les seconds semblent mieux appropriés à une analyse réelle.

En appliquant ce qui précède au programme ci-avant, on obtient, comme on le vérifie de suite,

	coûts uniformes	coûts logarithmiques
complexité temporelle	$\mathcal{O}(n)$	$\mathcal{O}(n^2 \log n)$
complexité spatiale	$\mathcal{O}(1)$	$\mathcal{O}(n \log n)$

Dans cet exemple, l'optique des coûts uniformes est réaliste seulement si un mot mémoire peut contenir un entier aussi grand que n^n. Dans le cas contraire, il convient de retenir les coûts logarithmiques. Toutefois, ces derniers sont sujets à critique car ce qui précède suppose que deux entiers i et j peuvent être multipliés en un temps de l'ordre de $\mathcal{O}(\ell(i) + \ell(j))$ ce qui n'est pas toujours possible.

On remarquera, en outre, que les complexités temporelle et spatiale calculées sur base des coûts uniformes s'identifient aux mêmes complexités de l'algorithme associé, calculées plus intuitivement selon le paragraphe I.4..

Les trois paragraphes qui suivent introduisent trois modèles découlant de la machine à accès aléatoire. Tous abstraient certaines de ses caractéristiques et en ignorent d'autres. La justification de cette façon de procéder réside dans le fait que les instructions ignorées représentent au plus une fraction constante du coût de tout algorithme résolvant les problèmes auxquels le modèle s'applique.

I.6. Le modèle séquentiel

La résolution de nombreux problèmes implique uniquement la construction de programmes dans lesquels les instructions de branchement ne sont utilisées que dans le seul but de provoquer la répétition d'une suite d'instructions un nombre de fois proportionnel à la taille n du problème. Ainsi en est-il de la multiplication de deux matrices de dimension n x n grâce à l'algorithme de la page 11 ou encore de l'évaluation du polynôme $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ par la règle d'Horner (évaluant le polynôme par l'égalité

$$p(x) = (\ldots ((a_n x + a_{n-1})x + a_{n-2})x + \ldots)x + a_0 \quad)$$

Dans un cas comme dans l'autre et plus généralement pour

tout programme de ce type, il est toujours possible d'éliminer les instructions de branchement en duplicitant les instructions à répéter un nombre approprié de fois. Ainsi, l'algorithme

```
| y := a_n;
| Pour i variant de 1 à n effectuer
|  | y := y * x + a_(n-i).
```

traduisant la règle de Horner devient respectivement pour n = 1, n = 2 et n = 3

```
| y := a_1;
| y := y * x + a_0.
```

```
| y := a_2;
| y := y * x + a_1;
| y := y * x + a_0.
```

```
| y := a_3;
| y := y * x + a_2;
| y := y * x + a_1;
| y := y * x + a_0.
```

Ceci se justifie par le fait que, pour ces programmes, le coût occasionné par les instructions de branchement qui y apparaissent ne dépasse pas une fraction constante du coût total du programme. Il en est souvent de même pour les instructions de lecture. Nous les négligeons aussi en supposant que les données se trouvent en mémoire au début de l'exécution du programme. En plus, comme un tel programme se termine nécessairement, l'instruction HALT est superflue. En outre, pour la même raison, il ne peut accéder qu'à un nombre fini de registres que l'on peut dès lors désigner par un nom symbolique. Cette remarque nous dispense encore de l'instruction d'écriture si l'on prend la précaution de réserver certaines adresses symboliques aux résultats. Enfin, comme l'effet des adressages indirects peut être déterminé une fois la taille du problème connue, les seules instructions nécessaires au modèle séquentiel sont STORE, ADD, SUB, MULT, DIV. Il s'ensuit, si l'on convient de remplacer les séquences du type

```
LOAD a
ADD b
STORE c
```

par l'instruction c := a + b, que tout programme de ce modèle peut s'écrire dans le pseudo-langage composé des instructions élémentaires de la forme

$$\delta := \alpha \gamma \beta$$

où α, β, δ désignent des noms symboliques et γ représente une des opérations arithmétiques ADD, SUB, MULT, DIV.

Dans ce modèle, la complexité temporelle théorique d'un programme est déterminée par le nombre d'opérations élémentaires exécutées et sa complexité spatiale théorique par le nombre de registres utilisés. Ainsi, les complexités temporelle et spatiale du programme d'évaluation d'un polynôme d'après la règle d'Horner sont respectivement $\mathcal{O}(2n)$ et $\mathcal{O}(n+4)$. De fait, en particularisant au cas n = 3, on peut l'écrire comme suit

$y := a_3 \times x;$
$y := y + a_2;$
$y := y \times x;$
$y := y + a_1;$
$y := y \times x;$
$res := y + a_0.$

Le registre nommé *res* fournit le résultat de l'évaluation.

L'intérêt de ce modèle est double. D'une part, il modélise parfaitement la comparaison ou l'analyse d'algorithmes qui consistent à compter le nombre d'instructions arithmétiques exécutées. La comparaison des algorithmes V.2.1.3. et V.2.3.1. en annexe constitue un exemple particulièrement éloquent d'une telle démarche. D'autre part, il permet d'introduire la notion de borne inférieure très utile dans la recherche de l'algorithme le plus performant.

Détaillons ce dernier point. La construction d'un algorithme capable de résoudre un problème soulève parfois la question de savoir combien d'opérations arithmétiques minimum sont requises pour le résoudre c'est-à-dire, en fait, le problème de déterminer une borne inférieure du nombre d'opérations arithmétiques que tout algorithme qui le résout vérifie. Ainsi, A.V. Aho, J.E. Hopcroft,

et J.D. Ullman établissent en [1] que l'évaluation d'un polynôme de degré n représenté par ses coefficients requiert au moins n multiplications. Ce résultat est tout particulièrement intéressant si l'on tente de construire un algorithme ayant cette spécification et minimisant le nombre de multiplications.

Il convient ici de noter qu'une telle borne n'a de sens que si le champ mathématique sur lequel travaillent ces opérations arithmétiques a été précisé. Par exemple, le calcul de $x^2 + y^2$ dans le champ des réels exige, comme on s'en doute, au moins deux multiplications, même si les multiplications par une constante sont négligées. Par contre, ce même calcul requiert une seule multiplication dans le champ des complexes à savoir $(x + iy)(x - iy)$. Cette distinction n'est néanmoins qu'académique car le champ communément utilisé est celui des réels.

I.7. Le modèle des circuits logiques

Le modèle précédent repose manifestement sur les coûts uniformes, qui, comme nous l'avons déjà dit, ne sont appropriés que dans le cas où les quantités manipulées sont de taille raisonnable . Il est possible de le modifier de façon à tenir compte de coûts logarithmiques. Le modèle obtenu, que nous appelons modèle des circuits logiques, ressemble essentiellement au modèle séquentiel si ce n'est

- que toutes les variables manipulées ne peuvent prendre que les valeurs 0 ou 1. (En ce sens, ce sont des bits)
- les opérations élémentaires utilisées sont logiques plutôt qu'arithmétiques : nous utiliserons l'opération
  - $\wedge$ pour l'opération logique et
  - $\vee$ pour l'opération logique ou
  - $\oplus$ pour l'opération logique ou exclusif
  - $\neg$ pour la négation.

Les complexités théoriques temporelles et spatiales se définissent de même que dans le modèle précédent.

On remarquera que, vu le choix des coûts employé, toute opération arithmétique sur les entiers i et j s'effectue en au moins $\ell(i) + \ell(j)$ pas. En fait, les meilleurs algorithmes connus de multiplication et de division d'entiers s'exécutent en plus de $\ell(i) + \ell(j)$ pas.

L'intérêt de ce modèle se situe à deux niveaux. D'une part, il est très utile pour exprimer les opérations de base telles que les opérations arithmétiques considérées comme primitives dans d'autres modèles. Par exemple, la multiplication de deux entiers, tous deux représentés par n bits, s'effectue en un pas de programme dans le modèle séquentiel alors que le meilleur résultat démontré dans ce modèle est de l'ordre de $(n \log n \log\log n)$ pas. D'autre part, il s'applique très bien aux circuits logiques ainsi qu'au calcul des fonctions booléennes. L'exemple suivant est particulièrement probant. On y tente de calculer la fonction de $\{0,1 \times 0,1\}$ dans $\{0,1\}$ définie par

$$f((x_1,y_1),(x_2,y_2)) = \begin{cases} 1 & \text{si } x_1 > y_1 \text{ ou si } x_1 = y_1 \text{ et } x_2 > y_2 \\ 0 & \text{sinon.} \end{cases}$$

Afin de construire le programme qui le calcule, remarquons qu'on a, en notant par $\bar{x}$, $x.y$ et $x + y$ respectivement les valeurs $\neg x$, $x \wedge y$, $x \vee y$,

$$x_1.\bar{y}_1 = 1 \Longleftrightarrow x_1 \quad y_1$$

et

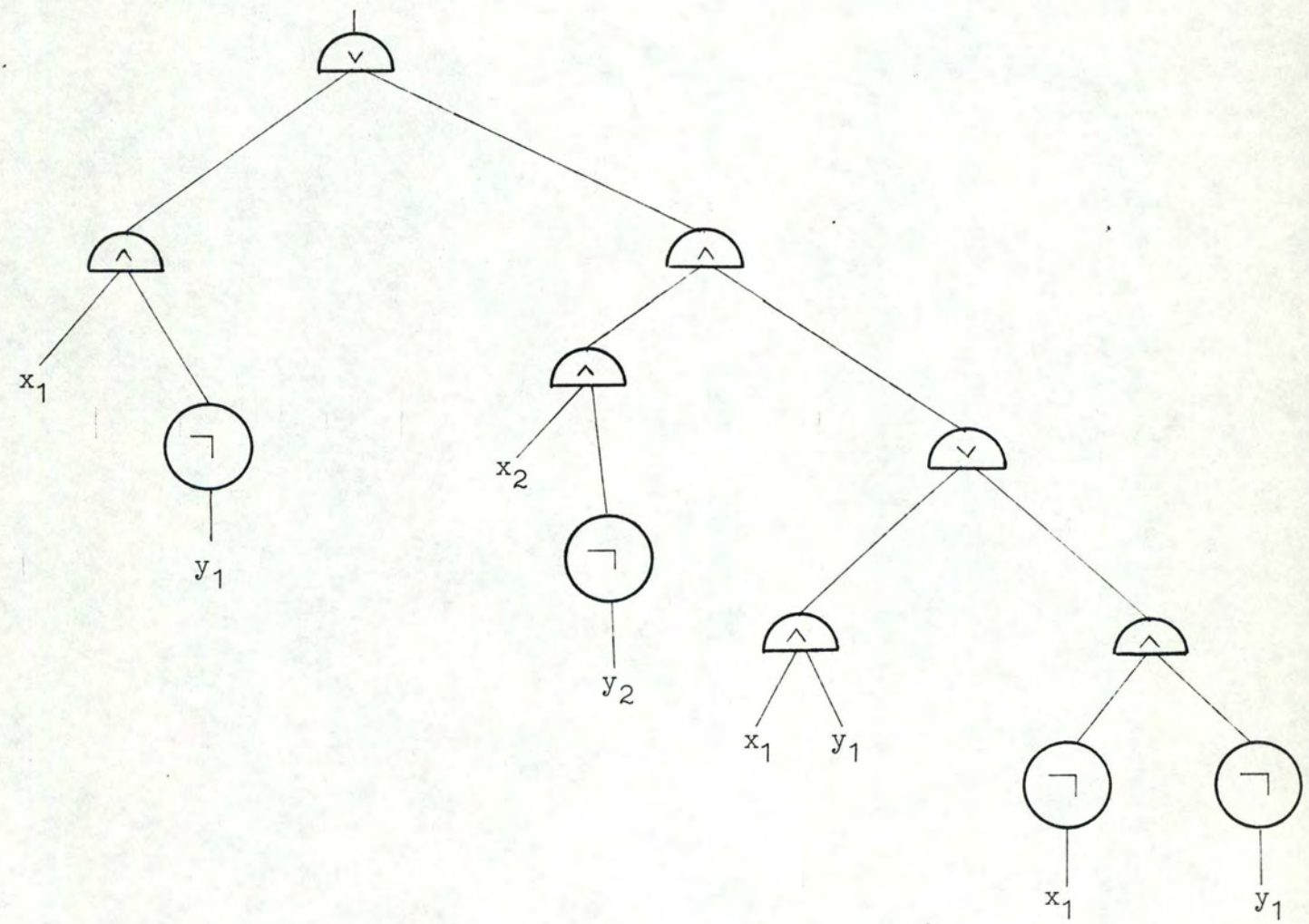
$$x_1.y_1 + \bar{x}_1.\bar{y}_1 = 1 \Longleftrightarrow x_1 = y_1 \ .$$

Dès lors, il vient

$$f((x_1,y_1),(x_2,y_2)) = x_1.\bar{y}_1 + (x_1.y_1 + \bar{x}_1.\bar{y}_1).x_2.\bar{y}_2 \ .$$

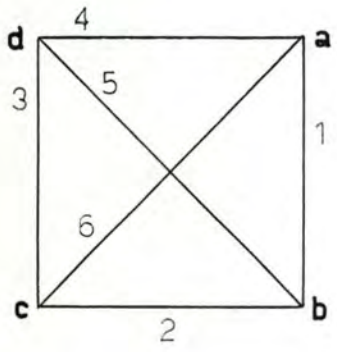
Le programme et le circuit suivants en découlent de suite

```
z := x̄1
w := ȳ1 .
u := z.w;
v := x1.y1;
y := u+v;
y := y.x2;
z := ȳ2;
y := y.z;
u := x1.w;
y := y+u.
```

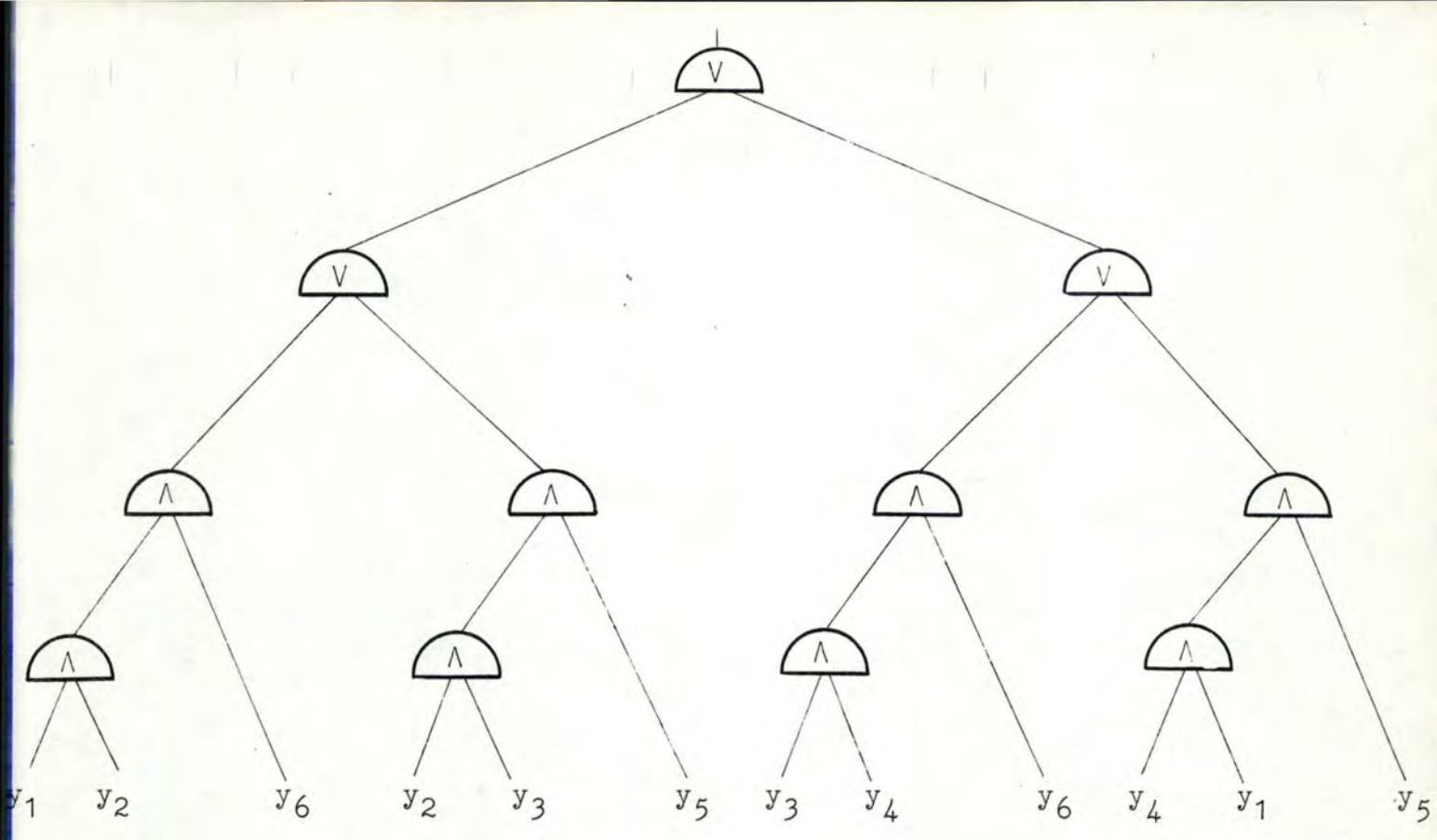


Remarquons, pour conclure, que ce modèle d'aspect fort théorique permet de résoudre certains problèmes de graphe. Par le programme suivant, nous nous proposons de résoudre le problème de la k - clique particularisé aux graphes à quatre dimensions et au cas où k = 3. Dans ces conditions, il s'agit de déterminer si un graphe donné de quatre sommets contient un sous-ensemble de trois sommets tel que toute paire de ses sommets soit reliée par une arête du graphe.

Ce problème équivaut à la recherche d'une fonction booléenne dont les arguments ont pour valeurs 0 et 1. De fait, en numérotant, conformément au graphe ci-dessous, de 1 à 6 les arêtes susceptibles d'exister entre les paires de sommets du graphe et en posant $y_i = 1$, $i \in \{1,\ldots,6\}$, si l'arête correspondante existe, $y_i = 0$ sinon, résoudre le problème revient à déterminer si l'un des ensembles {a,b,c}, {b,c,d}, {c,d,a}, {d,a,b}, possède la propriété que toute paire de sommets qui le composent est reliée par une arête. Or, ceci revient encore à prouver qu'un produit de trois variables y_i vaut 1. Par exemple, l'ensemble {a,b,c} vérifie la propriété ci-dessus si et seulement si on a $y_1 \cdot y_2 \cdot y_6 = 1$.



Il s'ensuit que le programme symbolisé par le circuit logique suivant résout le problème.



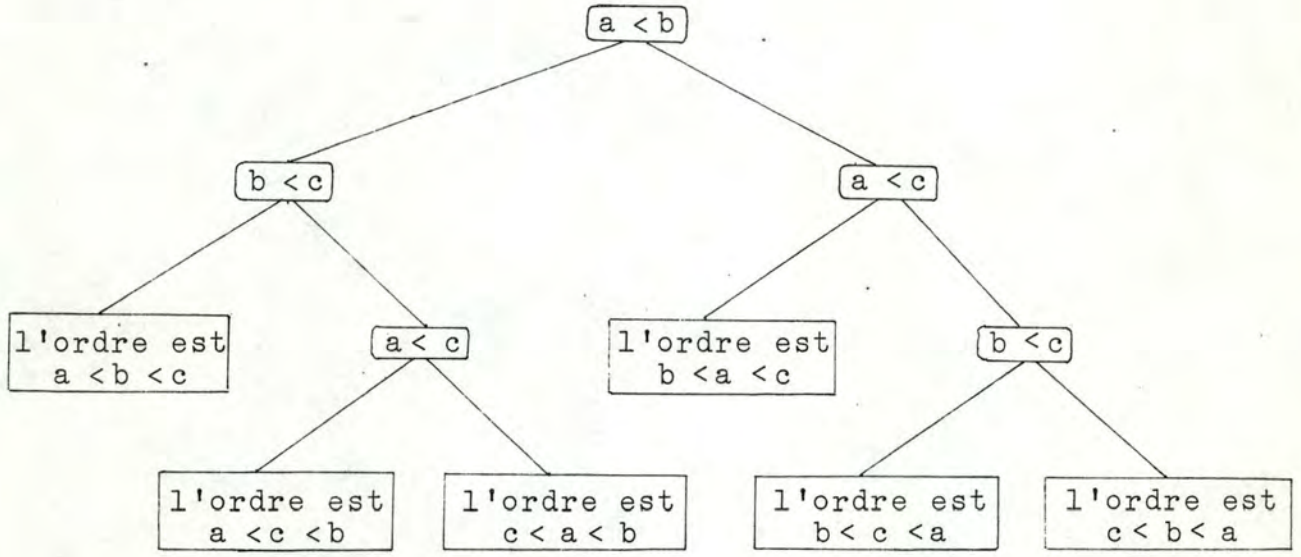
I.8. Le modèle des arbres de décision

Les deux modèles précédents ne considèrent que les pas de programme impliquant un calcul arithmétique ou logique et ignorent totalement les instructions de branchement. Pourtant, il semble réaliste, dans le traitement de certains problèmes, de considérer le nombre d'instructions de branchement comme une mesure primordiale de la complexité. C'est le cas notamment des programmes de tri pour lesquels les données sont identiques aux résultats si ce n'est qu'ils apparaissent dans un ordre différent.

Le modèle présenté dans ce paragraphe tient compte de ce fait et considère comme pas de programme les deux branches résultant d'une comparaison entre deux quantités. Les programmes y sont représentés sous forme d'arbre binaire appelés encore arbres de décision ; chacun des sommets y symbolise, en effet, une décision. Son parcours

s'effectue comme suit. Tout d'abord, le test représenté par la racine courante est effectué, puis, suivant le résultat obtenu, un de ses fils est examiné. Le résultat final apparaît quand une des feuilles de l'arbre est atteinte.

La figure suivante illustre cette notion d'arbre de décision. Elle décrit un programme dont la spécification est de trier trois nombres a, b, c. Les tests y sont encerclés ; pour chacun, la branche de gauche correspond au cas où il s'avère positif, la branche de droite au cas contraire.



Dans ce modèle, la complexité temporelle d'un algorithme est la hauteur de l'arbre de décision - c'est-à-dire le nombre maximum de comparaisons situées sur un chemin allant de sa racine à une de ses feuilles - considéré comme une fonction de la taille du problème traité.

Remarquons que le nombre de sommets d'un arbre de décision peut excéder de beaucoup sa hauteur. Ainsi, un arbre de décision associé à un programme de tri de n nombres a au moins n! feuilles alors que sa hauteur est de l'ordre de n log n.

L'intérêt porté à ce modèle provient du fait qu'il s'applique particulièrement bien aux programmes que l'on peut représenter sous forme d'arbres de décision. Les exemples les plus éloquents sont, sans doute, les programmes "Branch and Bound", de tri et encore de recherche dans des tables triées.

I.9. Les machines de Turing

Ce paragraphe décrit deux machines ayant accès à une masse d'information potentiellement illimitée. Nous les appelons machine déterministe de Turing et machine non déterministe de Turing en l'honneur de A.M. Turing qui les introduisit en 1938 dans son article [21] mondialement connu. Bien qu'on puisse les considérer comme l'expression la plus élémentaire de la notion de machine, elles n'en restent pas moins aussi puissantes que les machines ordinaires et, jusqu'à ce jour, aucun exemple d'algorithme n'ayant son équivalent sur ces machines n'est connu, ce qui renforce la thèse de Church selon laquelle tout procédé uniforme peut être réalisé sur une machine de Turing.

Ces machines sont dédiées à la résolution de problèmes de décision c'est-à-dire de problèmes dont les solutions sont booléennes (oui ou non). Ceci ne restreint en rien leur généralité car, comme on le vérifie de suite, tout problème peut toujours se formuler sous une telle forme.

A toutes fins utiles, rappelons, avant de les exposer, quelques notions relatives à la théorie des langages. Dans la suite de ce travail, nous appelons

- alphabet, un ensemble fini de symboles,
- chaîne de l'alphabet Σ une suite finie d'éléments de Σ
- longueur d'une chaîne, le nombre d'éléments qui la composent,
- langage sur l'alphabet Σ, un ensemble de chaînes de cet alphabet.

De plus, les notations

Σ, Σ^*, L

désignent respectivement un alphabet, l'ensemble des chaînes de l'alphabet Σ et un langage.

La notion de schéma d'encodage établit une correspondance entre les problèmes de décision et les langages. Nous définirons un schéma d'encodage e relatif à l'alphabet Σ et à la classe de problèmes π comme une application qui, à tout problème de π, associe une chaîne de Σ^*. Comme on le vérifie de suite, ceci partitionne l'ensemble Σ^* en trois classes de chaînes

1) l'ensemble des chaînes qui ne correspondent à l'encodage d'aucun problème de π,

2) l'ensemble des chaînes correspondant à l'encodage de problèmes de π dont la solution est positive,

3) l'ensemble des chaînes correspondant à l'encodage de problèmes de π dont la solution est négative.

La deuxième joue un rôle particulier. Nous l'appelons langage associé à π et à e et la notons $L(\pi,e)$.

Il convient ici de remarquer qu'on peut associer en toute généralité à toute classe de problèmes plus d'un schéma d'encodage. Cette remarque nous obligera dans la suite à observer, chaque fois qu'un nouveau concept est introduit en terme de langage, que la propriété associée est indépendante du codage utilisé. Ceci sera toujours vérifié tant que nous nous limiterons à utiliser des schémas d'encodage "raisonnables".

Dans cette optique, la propriété envisagée pourra être considérée comme propre à la classe de problèmes et indépendante d'un schéma d'encodage. Cette façon de faire requiert évidemment la possibilité d'exhiber, à la demande, un schéma d'encodage particulier e tel que la propriété soit vérifiée pour $L(\pi,e)$. Notons encore qu'en procédant de la sorte, nous perdons toute idée précise de la longueur des entrées. Comme il s'agit d'une notion fondamentale pour la complexité des algorithmes, nous supposerons qu'est associée à chaque classe de problèmes de décision π une fonction indépendante de l'encodage Length : $\pi \to \mathbb{N}$ polynomialement reliée

aux longueurs des entrées qu'on pourrait obtenir à l'aide d'un schéma d'encodage raisonnable. Cette dernière restriction équivaut encore à imposer pour tout schéma d'encodage raisonnable e de π, l'existence de deux polynômes p et p' telle que tout problème I de π et toute chaîne x qui l'encode sous e, vérifient les inégalités

$$\text{Length}(I) \leqslant p(|x|)$$

et

$$|x| \leqslant p'(\text{Length}(I))$$

où $|x|$ désigne la longueur de x.

Reste à présent à convenir du caractère raisonnable d'un schéma d'encodage. L'acception généralement admise l'identifie aux deux notions de concision et de décodabilité.

Développons-les. La première impose que les problèmes soient décrits aussi brièvement qu'ils le sont dans la pratique. Par suite, il est exclu d'allonger artificiellement les données d'un problème pour essayer, par exemple, de convertir un algorithme de complexité exponentielle en un algorithme de complexité polynomiale. La seconde exige qu'à toute variable de l'énoncé générique décrivant la classe de problèmes traitée puisse être associé un algorithme de complexité temporelle polynomiale capable d'extraire sa valeur d'un problème particulier de la classe.

Bien qu'en général satisfaisante, cette pseudo-définition ne fournit toutefois pas de définition formelle de "schéma d'encodage raisonnable". Alors même que la majorité des informaticiens pourrait s'accorder sur le caractère raisonnable d'un schéma d'encodage particulier, l'absence d'une telle définition peut paraître inconfortable. Une façon élégante de résoudre cette difficulté serait d'exiger que les énoncés génériques des classes de problèmes soient formés à partir d'une collection déterminée d'objets de base. Nous n'imposerons pas une telle contrainte ici. Toutefois, la notion de schéma d'encodage utilisée dans la suite s'en inspire. Elle réfère à la description suivante.

Soit l'alphabet $\Sigma = \{0, 1, -, [,], (,), ,\}$. Définissons d'abord récursivement les chaînes structurées par les règles suivantes

(1) la représentation binaire d'un entier k considéré comme chaîne de 0 et de 1 (précédée du signe - si k est négatif) est une chaîne structurée représentant cet entier

(2) si x est une chaîne structurée représentant l'entier k, alors [x] est une chaîne structurée. Elle peut être utilisée comme un nom (par exemple, pour un sommet dans un graphe, un ensemble d'éléments)

(3) si les chaînes structurées $x_1,\ldots,x_m$ représentent les objets $X_1,\ldots,X_m$, alors $(x_1,\ldots,x_m)$ est une chaîne structurée ; elle représente la suite $\langle X_1,\ldots,X_m\rangle$

La dernière permet, connaissant la représentation de chaque objet intervenant dans un énoncé générique d'une classe de problèmes, de construire la représentation de cet énoncé. Dès lors, pour dériver un schéma d'encodage d'un problème de décision particulier spécifié dans ce format standart, il reste à fournir la représentation de chaque type d'objet. Nous ne considérerons ici que les entiers, les nombres rationnels, les suites, les ensembles, les graphes, les fonctions définies dans des ensembles finis ainsi que les éléments non structurés.

- Les règles (1) et (3) délivrent la représentation des entiers et des suites
- Pour représenter les éléments non structurés d'un problème, il suffit d'assigner à chacun un nom construit par la règle 2 de telle sorte que si l'on en dénombre N, aucun nom de taille strictement supérieur à N ne soit utilisé
- Un ensemble est représenté en ordonnant ses éléments et en considérant la chaîne structurée correspondant à la suite ainsi créée
- Le graphe (X,U) est représenté par la chaîne structurelle (α,β) où α et β sont respectivement les chaînes structurées représentant l'ensemble des sommets X et l'ensemble des arcs U.

- La fonction finie $f : \{U_1, \ldots, U_m\} \to \mathbb{W}$ est représenté par la chaîne $((x_1,y_1), \ldots, (x_m,y_m))$ où x_i est la chaîne structurée représentant l'objet $f(U_i) \in \mathbb{W}$ $(1 \leqslant i \leqslant m)$
- Le nombre rationnel q est représenté par la chaîne structurée (x,y) où x et y représentent respectivement les entiers a et b premiers entre eux et tels que q = a/b.

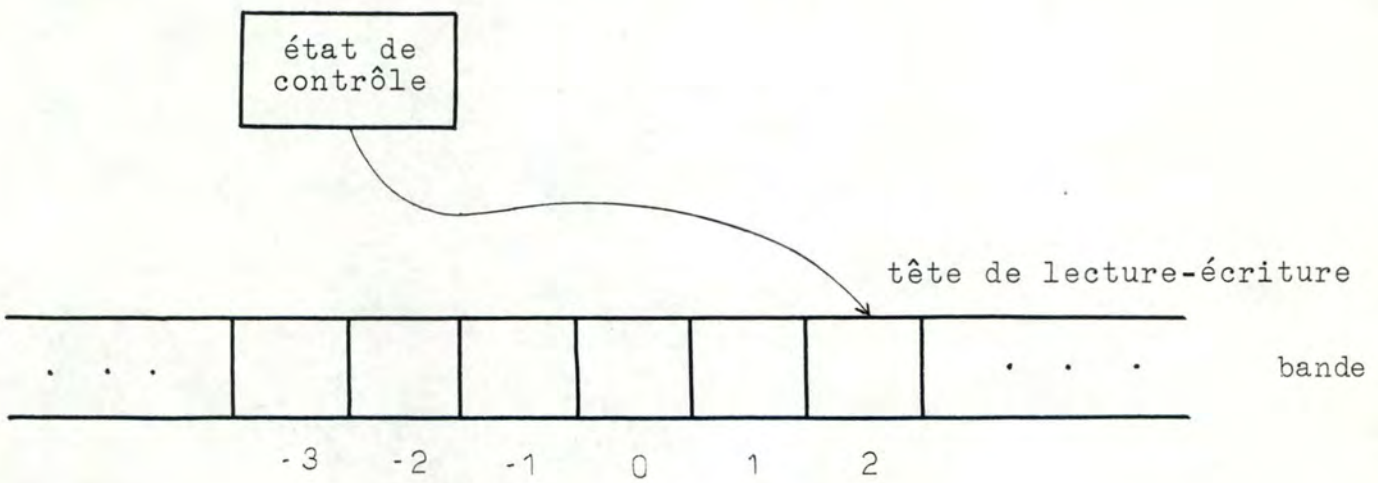
Décrivons à présent la machine déterministe de Turing.

I.9.1. La machine déterministe de Turing

Elle consiste en

- une bande formée de deux suites infinies de carrés, indicées par $\mathbb{Z}$
- une tête de lecture-écriture
- un état de contrôle
- un programme, appelé D-programme et défini par
  * un ensemble fini Γ de symboles incluant un sous-ensemble $\Sigma \subset \Gamma$ de symboles d'entrée et le symbole blanc $b \in \Gamma - \Sigma$
  * un ensemble fini Q d'états contenant un état de départ q_0 et deux états d'arrêts q_Y et q_N
  * une fonction de transition

    $\delta : (Q \setminus \{q_Y, q_N\}) \times \Gamma \to Q \times \Gamma \times \{-1,1\}$



L'exécution du D-programme se déroule comme suit. Initialement, une chaîne de longueur ℓ composée d'éléments de Σ remplit les cases d'indices 1 à ℓ, l'état de contrôle est q_0 et la tête de lecture-écriture explore le carré d'indice 1. Le processus suivant s'itère. Si l'état de contrôle q est q_Y ou q_N, le calcul se termine et fournit la réponse oui si on a $q = q_Y$, non sinon. Dans le cas contraire, en notant s le symbole sous la tête de lecture-écriture, la fonction δ appliquée à q et s délivre le triplet $\delta(q,s) = (q',s',\Delta)$. La tête de lecture-écriture efface alors s, imprime s' à sa place et se déplace d'un carré vers la droite si $\Delta = 1$, d'un carré vers la gauche si $\Delta = -1$. En outre, l'état de contrôle devient q'. Une nouvelle itération peut alors commencer

Le nombre d'itérations ainsi créées détermine la complexité temporelle de ce programme. De fait, pour tout D-programme dont l'exécution à partir de toute chaîne de Σ^* se termine, nous définissons sa complexité temporelle, comme la fonction de n

$$T_M(n) = \max \{m : \exists \text{ une chaîne } x \in \Sigma^* \text{ de longueur n telle que l'exécution de M à x se termine après m pas}\}$$

Par suite, nous dirons d'un tel D-programme qu'il est de complexité temporelle polynomiale s'il existe un polynôme p tel que pour tout $x \in \mathbb{N}$, $T_M(n) \leqslant p(n)$.

Convenons encore des définitions suivantes :

1) une chaîne $x \in \Sigma^*$ est acceptée par le D-programme M si l'exécution de M avec x pour état initial du ruban se termine sur l'état q_Y ;
2) le langage reconnu par le D-programme M est l'ensemble L_M de chaînes de Σ^* acceptées par x ;
3) un D-programme M résout le problème de décision π sous le schéma d'encodage e si M s'arrête pour toutes les chaînes de son alphabet d'entrée et si l'on a $L_M = L(\pi,e)$.

La classe suivante joue un rôle fondamental dans la théorie de la NP-complétion. Mathématiquement, elle s'écrit

$$P = \{L : \exists \text{un D-programme M de complexité temporelle polynomiale tel que } L = L_M\}$$

Moins formellement, on dit qu'une classe de problèmes de décision π appartient à P sous le schéma d'encodage e si le langage $L(\pi, e)$ appartient à P ou, ce qui est équivalent, s'il existe un D-programme de complexité polynomiale qui résout π sous le schéma d'encodage e. La discussion précédente sur les schémas d'encodage raisonnables nous permet de dire plus simplement dans ces conditions que π appartient à P.

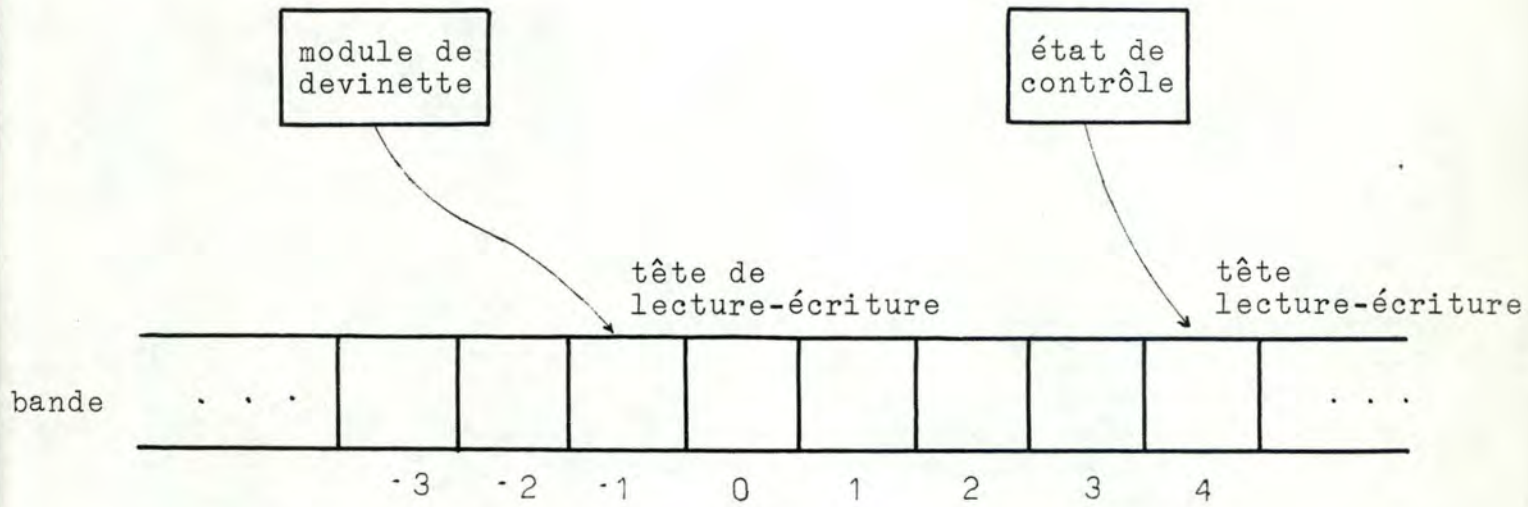
Il est une autre classe tout aussi intéressante. Elle appréhende non pas, comme la précédente, la possibilité de résoudre un problème par un algorithme de complexité temporelle polynomiale mais la faculté de vérifier par un tel algorithme qu'une solution candidate est bien solution du problème. Afin de la définir; on pourrait, en raisonnant intuitivement, tenter de remplacer dans la définition de P, la notion de programme par celle de programme non déterministe, en ce sens que son exécution se déroulerait en une phase de devinette suivie d'une phase de choix. La première consisterait à deviner une solution d'un problème de la classe à résoudre qui conjointement à la description de ce problème conduirait la deuxième, déterministe cette fois, à la réponse oui, à la réponse non, ou encore à un bouclage infini. Ceci n'est évidemment raisonnable que si tout programme non déterministe vérifie l'équivalence : pour tout problème I de la classe π analysée, la réponse à I est oui si et seulement s'il existe une solution de I telle que l'exécution de la phase de choix avec pour donnée cette solution et la description de I délivre la réponse oui.

Remarquons qu'elle introduit une dissymétrie entre les réponses oui et non qui n'existe pas dans la définition de la classe P. De fait, dans ce cas, la résolution par un programme déterministe du problème "Etant donné I, est-ce que X est vrai pour I" implique que son complémentaire "Etant donné I, X est-il faux pour I" peut être résolu par un programme déterministe. Comme tous les programmes déterministes, considérés pour définir la classe P, convergent pour toute entrée, il suffit, en effet, d'interchanger, dans le programme déterministe, les états d'arrêts q_Y et p_N. Un tel raisonnement ne s'adopte bien sûr pas aux programmes non déterministes.

La machine non déterministe de Turing permet de définir rigoureusement les programmes non déterministes.

I.9.2. La machine non déterministe de Turing

La machine non déterministe de Turing ressemble en certains points à la machine déterministe de Turing. Elle dispose aussi d'une bande formée de deux suites infinies de carrés indicée par $\mathbb{Z}$ et possède de même un état fini de contrôle ainsi qu'une tête de lecture-écriture. Toutefois, un module de devinette muni de sa propre tête de lecture-écriture s'y ajoute. Enfin, le D-programme est remplacé par un NP-programme défini de la même façon mais d'exécution différente car composée de deux stades distincts.



Dans le premier, une chaîne devinée est écrite dans les cases d'indices négatifs. Pour ce faire, le procédé suivant est utilisé. Initialement, une chaîne x de longueur $|x|$ remplit les carrés d'indices 1 à $|x|$; les autres carrés de la bande contiennent le symbole blanc. La tête de lecture-écriture du module de devinette scanne le carré d'indice -1 et celle de l'état de contrôle le carré d'indice 1. Le processus suivant s'itère alors. Le module de devinette écrit un symbole deviné arbitrairement sur la case qu'explore sa tête de lecture-écriture. Puis une des deux actions suivantes s'exécute. La première déplace cette tête d'un carré vers la gauche et une nouvelle itération peut commencer. La seconde arrête le module de devinette ; l'état de contrôle est alors activé

à l'état q_o. Le choix entre ces deux situations est décidé aléatoirement par le module de devinette.

Le second, le stade de choix, s'exécute une fois l'état de contrôle activé à l'état q_o. A partir de ce point, tout se déroule conformément à l'exécution d'un D-programme. Bien entendu, cette fois, la chaîne devinée peut et est habituellement examinée. L'exécution s'arrête à nouveau lorsque l'état de contrôle est un des deux états d'arrêt. S'il s'agit de q_y, l'exécution est dite acceptable.

Notons ici qu'à tout ND-programme M est associé une infinité d'exécutions possibles correspondant à chaque chaîne devinée. Nous dirons qu'un ND-programme accepte la chaîne x si au moins une de ces exécutions à partir de x comme contenu initial de la bande est acceptable. Le langage reconnu par le ND-programme M, noté L_M, est à nouveau l'ensemble $\{x \in \Sigma^* : M \text{ accepte } x\}$

Du fait de l'existence d'un module et d'une phase de devinette, la complexité d'un ND-programme diffère de celle d'un D-programme . Nous définirons le temps requis par un ND-programme M pour accepter la chaîne $x \in L_M$, comme le minimum sur tous les calculs acceptables de M pour x, du nombre de pas apparaissant dans les étapes de devinette et de choix jusqu'à l'obtention de l'état q_y. Par suite, la complexité temporelle théorique de M vaut

$$T_M(n) = \max \{1, \max \{ m : \exists \text{ une chaîne } x \in L_M \text{ de longueur } n \text{ telle que le temps mis par M pour accepter x est } m\}\}$$

Un ND-programme M est dit "de temps polynomial" s'il existe un polynôme p tel que $T_M(n) \leqslant p(n)$, pour tout n.

Nous sommes à présent à même de définir la classe NP. Formellement, celle-ci s'écrit

$$NP = \{L : \exists \text{ un ND-programme M de temps polynomial tel que } L_M = L\}$$

Un raisonnement analogue à celui effectué pour la classe P permet d'affirmer qu'une classe de problèmes π appartient à NP sous le schéma d'encodage e si le langage $L(\pi,e)$ appartient à NP et, par suite, d'identifier la classe NP à l'ensemble des classes de problèmes de décision résolubles par des algorithmes non déterministes de temps polynomial.

I.10. A propos de l'algorithme le plus performant

Les paragraphes précédents décrivent plusieurs outils de performance d'algorithmes. Ils soulèvent inévitablement les questions :

Quand peut-on considérer qu'un algorithme est bon ?

Quand peut-on considérer qu'un algorithme est efficace ?

Quand peut-on considérer qu'un algorithme est performant ?

Quels outils convient-il d'appliquer pour évaluer judicieusement les performances d'un algorithme ?

Ces questions sont tout particulièrement importantes car, pour un utilisateur, le critère d'efficacité ou de performance est le seul, en dernier recours, qui lui permette de choisir entre deux ou plusieurs algorithmes ayant, par ailleurs, les mêmes prétentions.

Selon Donovan - et d'une façon générale tous ceux qui choisissent un langage de base pour représenter les algorithmes - pour apprécier correctement les performances d'un algorithme, il faut calculer son temps réel moyen d'exécution. Aucune autre mesure ne peut en donner une bonne approximation.

Cet avis peut être critiqué. D'une part, toute moyenne n'a de sens que si elle est rapportée à une distribution probabiliste, ce qui est rarement le cas pour les données. Pour nous en convaincre, considérons l'exemple suivant de A. Cournot ([11]). Prenons, dit-il, plusieurs triangles rectangles et faisons la moyenne des longueurs des côtés. Les trois moyennes obtenues peuvent bien sûr servir à bâtir un nouveau triangle. On pourrait peut-être l'appeler triangle moyen mais il n'est pas rectangle. Pour preuve, les quatre triangles rectangles suivants

	1 er côté	2 eme côté	hypothénuse
	5	12	13
	15	8	17
	3	4	5
	7	24	25
moyenne :	7,5	12	15

pour lesquels le résultat n'est pas un triangle rectangle puisque la somme des carrés des deux premiers côtés est $(7,5)^2 + (12)^2 = 200,25$ et que le carré de l'hypothénuse vaut $(15)^2 = 225$. Dès lors, en combinant plusieurs objets possédant tous une même régularité, on obtient un objet qui n'a plus la même propriété. Par conséquent, l'interprétation des données statistiques ne peut reposer sur la la seule moyenne · il convient de tenir compte d'autres paramètres importants des distributions manipulées.

D'autre part, le temps moyen n'apporte aucune information sur le comportement d'un algorithme dans un cas particulier. Or, ce qui intéresse fondamentalement l'utilisateur, c'est de pouvoir répondre à la question : étant donné tel cas de figure, quel algorithme se montrera le plus performant ?

Enfin, le temps moyen se rapporte à un modèle particulier d'ordinateur. Par suite, les conclusions peuvent ne pas se transposer exactement à un autre modèle.

Il n'empêche, il convient de l'avouer, que cette mesure constitue, en pratique, un excellent indicateur de performance.

Notre propos n'est pas de polémiquer sur ce sujet ; nous voulons simplement mettre en évidence les quelques remarques suivantes qui nous semblent fondamentales :

1) Pour évaluer un algorithme capable de résoudre un problème, il n'existe pas un seul outil à appliquer, mais plusieurs, certains étant mieux appropriés à certaines situations qu'à d'autres. Ainsi, R. Lessuisse compare-t-il en [17] des algorithmes de recherche dichotomiques sur base du nombre d'essais. Ainsi comparons-nous en annexe deux algorithmes de programmation linéaire sur base de multiplications et de divisions qu'ils contiennent.

2) Si on n'y prend garde, les mesures prises peuvent très bien sous-estimer ou surestimer les capacités d'un algorithme. Par exemple, la complexité tient compte de tous les cas de figure possibles mais il est fort possible que, pour les applications traitées, l'algorithme soit appliqué à des données favorables et, par suite, ait un comportement meilleur que celui annoncé. De même, en n'y faisant pas attention, on peut très bien calculer un temps moyen d'exécution en générant aléatoirement des données qui n'ont que peu de rapports avec les données manipulées en réalité.

3) Il est agréable lorsqu'on développe un produit logiciel de pouvoir disposer d'algorithmes capables de résoudre le problème qu'on leur a assigné mais, en outre, en les modifiant légèrement, de résoudre certains problèmes voisins. En ce qui concerne ce dernier point, certains algorithmes s'adaptent mieux que d'autres.

Nous conseillons donc pour évaluer un algorithme ou encore pour comparer plusieurs, de prendre autant de mesures que possible de façon la plus réaliste et la plus adéquate qui soit, puis de les agréger. La procédure d'agrégation sera bien sûr propre à chaque utilisateur, de même que la réponse aux questions précédentes ainsi qu'à la question de savoir l'algorithme le plus performant d'un ensemble d'algorithmes.

CHAPITRE II

VERS UNE METHODOLOGIE DE RECHERCHE D'ALGORITHMES EFFICACES

Sous ce titre bien ambitieux, nous décrivons certaines approches permettant d'obtenir, étant donné un problème ou plus généralement une classe de problèmes, un algorithme de complexité spatiale et temporelle faible qui le résout. Il s'agit, sans nul doute, d'une des branches les plus difficiles de la programmation et, par voie de conséquence, l'obtention d'un algorithme efficace restera certainement un art à part entière. Bien qu'il n'existe pas de règles miracles, on peut, cependant, guider la recherche d'un algorithme performant. En écrivant ce chapitre, notre but est moins d'énoncer une liste fastidieuse de méthodes utilisées que de présenter quelques points saillants qui ont retenu notre attention. De ce fait, ce chapitre peut paraître incomplet mais, répétons-le, nous n'avons nullement la prétention d'être exhaustif.

Enonçons, tout d'abord, quelques principes généraux.

II.1. Quelques principes généraux

II.1.1. Diviser pour régner

L'une des idées les plus décisives consiste à décomposer un problème de taille n en une opération d'une certaine complexité, $\mathcal{O}(n)$ ou $\mathcal{O}(1)$ par exemple, et en un problème semblable de taille m, inférieure à n, ou plus généralement k problèmes de tailles $m_1, m_2, \ldots, m_k$ telles que $m_1 + m_2 + \ldots + m_k \leqslant n$.

Ceci n'est rien d'autre que l'idée d'une résolution récursive d'un problème. Pour rappel, cette méthode implique les trois étapes

- paramétrage du problème, faisant apparaître les différents éléments dont dépend la solution et, en particulier, la taille du problème à résoudre.
- recherche d'un cas trivial et de sa solution. C'est souvent l'étape-clé de l'algorithme. (Un tel cas apparaît généralement quand la taille du problème est nulle ou égale à 1)
- décomposition du cas général visant à le ramener à un ou plusieurs problèmes, en principe plus simple(s), de taille plus petite.

Bon nombre de méthodes de tri en découlent.

II.1.2. Equilibrer

Toutes les méthodes de division, cependant, ne donnent pas le même résultat du point de vue de la complexité théorique. Pour nous en convaincre, considérons les deux méthodes de division suivantes.

La première ramène le problème de taille n à un problème de taille n-1, en exécutant des opérations de complexité temporelle $\mathcal{O}(n)$. Il vient, dès lors, si T(n) désigne la complexité temporelle de l'algorithme qui en découle et c une constante,

pour n assez grand,

$$T(n) \leqslant c\,n + T(n-1)$$

et encore, en développant,

$$T(n) \leqslant c\,n + c(n-1) + \ldots + c + T(0)$$

c'est-à-dire

$$T(n) \leqslant c\,\frac{n(n+1)}{2} + T(0)$$

Par conséquent, l'algorithme sous-jacent à cette méthode est de complexité temporelle théorique $\mathcal{O}(n^2)$.

La deuxième ramène un problème de taille n en deux sous-problèmes de taille $\lceil \frac{n}{2} \rceil$ (où $\lceil x \rceil$ désigne le plus petit entier plus grand ou égal à x) moyennant aussi des opérations de complexité $\mathcal{O}(n)$. Il vient alors, en utilisant des notations semblables,

$$T'(n) \leqslant c\,n + 2\,T'(\lceil \tfrac{n}{2} \rceil)$$

et, donc,

$$T'(n) \leqslant c\,n + 2c\,\tfrac{n}{2} + 4c\,\tfrac{n}{4} + \ldots + 2^{\log_2 n}\,T'(1)$$

c'est-à-dire

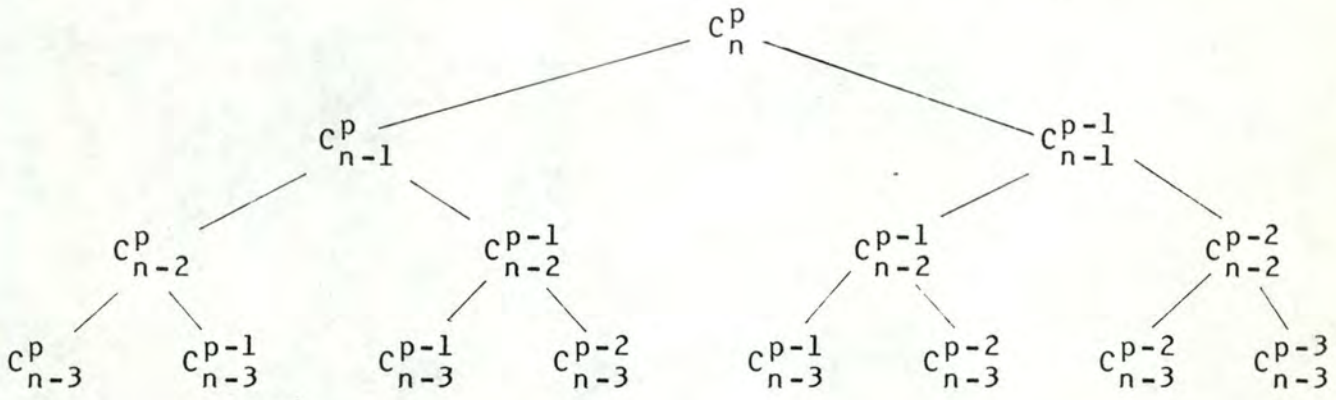
$$T'(n) \leqslant \log_2 n\ c\,n + n\,T'(1).$$

Dans ces conditions, l'algorithme obtenu est, cette fois, de complexité temporelle théorique $\mathcal{O}(n \log n)$, ce qui est incomparablement meilleur que $\mathcal{O}(n^2)$.

Il ne suffit donc pas de diviser pour obtenir des algorithmes efficaces, encore faut-il équilibrer les morceaux résultant des divisions. A ce propos, l'expérience enseigne que, en général, parmi toutes les partitions possibles, c'est la division en parties égales qui donne les meilleurs résultats.

II.1.3. Eviter les répétitions

L'étude approfondie d'un algorithme met parfois en évidence la répétition de l'évaluation d'une même quantité. Ainsi en est-il de l'algorithme récursif de calcul du nombre de combinaisons C_n^p basé sur les formules $C_n^p = C_{n-1}^p + C_{n-1}^{p-1}$, $C_o^n = 1$, $C_n^o = 1$ pour tout $n,p > 0$. Comme le montre le graphe ci-dessous, il entraîne, en effet (notamment), le calcul de C_{n-2}^{p-1} deux fois ainsi que l'évaluation de C_{n-3}^{p-1} et C_{n-3}^{p-2} trois fois.



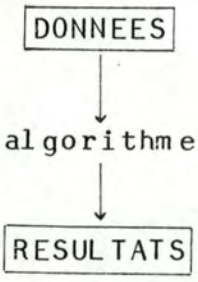
De telles répétitions ne peuvent évidemment être tolérées si elles peuvent être éliminées, ce qui est généralement le cas. Ainsi dans l'exemple précédent, on procédera avantageusement de façon ascendante en remontant des feuilles de l'arbre à sa racine, plutôt qu'en parcourant l'arbre de haut en bas.

II.1.4. Eviter un encombrement mémoire et l'exécution d'instructions inutiles

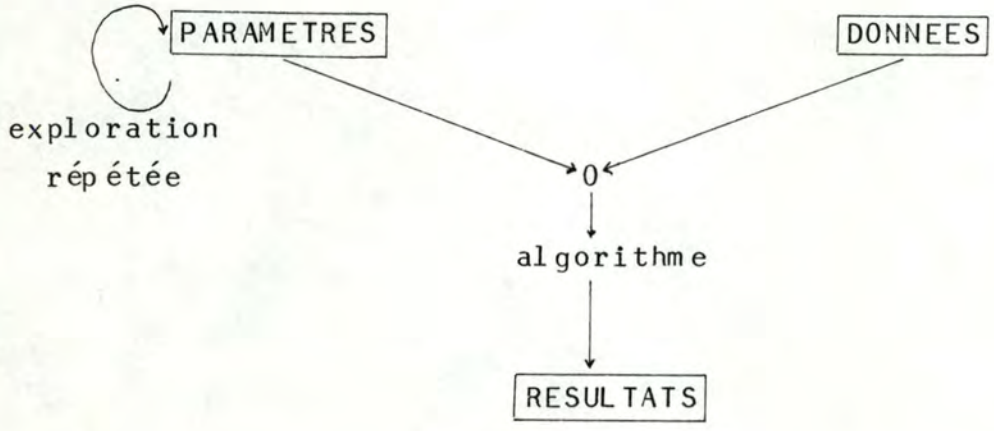
Ce principe paraît évident. Il est, toutefois, bon de le rappeler comme le suggère l'exemple classique de l'évaluation récursive de la fonction factorielle.

II.1.5. Compiler les paramètres

La première ébauche d'un algorithme conduit souvent à traiter toutes les données telles quelles:

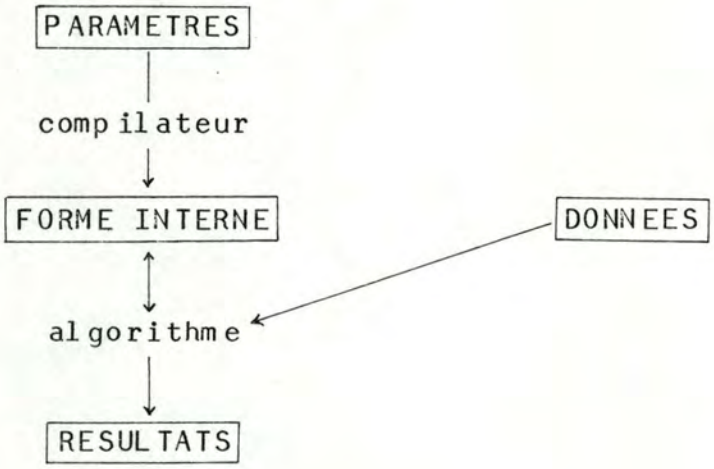


Souvent, cependant, les données sont de deux types: les unes servent à paramétrer l'algorithme et sont réutilisées plusieurs fois; les autres servent à décrire les caractéristiques précises de chaque problème que l'on cherche à résoudre à un moment donné. Le schéma ci-dessus est donc, en fait, mieux décrit par:



Par analogie avec la structure des systèmes permettant d'exécuter des programmes écrits dans des langages de programmation et en considérant les "paramètres" comme un programme écrit dans un certain langage, on peut donc considérer un tel algorithme comme un interprète. Or, il est bien souvent décisif, dans l'amélioration de la complexité de l'algorithme, de mettre au préalable les paramètres sous une forme interne, permettant une utilisation beaucoup plus efficace. On est donc conduit, dans un tel cas,

à compiler les paramètres



II.1.6. Le compromis espace-temps

Bien des programmes existants peuvent être améliorés d'un ordre de grandeur à la fois en temps et en espace par la seule application de techniques élémentaires, par exemple, en améliorant les structures de données employées. Toutefois, une fois un niveau d'efficacité convenable atteint, on est souvent obligé de choisir entre une amélioration de la complexité spatiale et une amélioration de la complexité temporelle, l'une se faisant au détriment de l'autre.

A titre d'exemple, supposons trois grandeurs X, Y, Z reliées par une équation du type

$$X = f(Y,Z) \qquad (*) .$$

On peut imaginer, pour fixer les idées, qu'ils représentent les crédits, les débits et les soldes. Si l'on traite un grand nombre de triplets de la forme (X,Y,Z), disons 10 000 ou 1 000 000, on peut se poser la question de savoir s'il faut conserver tous les triplets, ou seulement les paires (Y,Z), les X correspondants étant recalculés par la relation (*) lorsqu'on en a besoin. Il n'y a évidemment pas

de réponse universelle à cette question : le choix dépend du mode et de la fréquence d'utilisation des diverses données et de l'importance respective des limitations en temps et en espace. Deux règles générales se dégagent cependant :

- garder présente à l'esprit l'importance du compromis espace-temps : bien souvent, par exemple, on arrive à traiter des problèmes qui paraissaient demander une capacité de mémoire énorme, en recalculant systématiquement certaines quantités, recalcul qui paraît souvent absurde en première approche;
- considérer toujours un nom de donnée comme représentant moins la donnée elle-même qu'un mécanisme d'accès à cette donnée. De cette façon, les programmes utilisant X, dans l'exemple ci-dessus, n'ont pas besoin de "savoir" si X est obtenu directement ou recalculé : il ne sera pas nécessaire de le modifier si l'on change la représentation interne des triplets, par exemple, si, à la suite de tests statistiques, on décide qu'il est plus économique de calculer les X_i par un appel de sous-programme que de les stocker.

II.2. La théorie de la NP-complétion

La théorie de la NP-complétion offre aussi des directions de recherche d'algorithme performant.

Afin d'établir sa pertinence, rappelons qu'au vu des tableaux I.4.1., I.4.2. et I.4.3., les algorithmes de complexité polynomiale semblent supérieurs aux algorithmes de complexité exponentielle. J. Edmonds écrit d'ailleurs à ce sujet en [6] : "Les algorithmes de complexité (temporelle) exponentielle ne devraient pas être considérés comme de bons algorithmes, alors qu'il en est ainsi d'habitude. La plupart d'entre eux ne sont que des variantes de recherches exhaustives tandis que les algorithmes de complexité (temporelle) polynomiale reposent généralement sur la perception de propriétés

intrinsèques du problème qu'ils résolvent. ... Il est généralement admis qu'un problème n'est convenablement résolu que lorsqu'un algorithme de complexité (temporelle) polynomiale qui le solutionne a été découvert.".

A cela, on pourrait objecter que l'exécution d'algorithmes de complexité temporelle exponentielle peut très bien se montrer plus rapide que celle d'algorithmes de complexité temporelle polynomiale. En outre, certains algorithmes de complexité temporelle exponentielle sont très utilisés et apparaissent très efficaces en pratique. Ainsi en est-il de l'algorithme primal du simplexe en programmation linéaire et des algorithmes "branch and bound" généralement considérés comme des succès en dépit de leur complexité exponentielle.

Malheureusement, de tels cas sont rares. Bien que beaucoup d'algorithmes, de temps d'exécution exponentiel soient connus, peu d'entre eux sont réellement employés en pratique. Et même la popularité des algorithmes précédents n'a pas découragé les recherches d'algorithmes de complexité polynomiale qui les substitueraient. En fait, dans ces derniers cas, leur grand succès conduit à penser qu'ils captent d'une façon ou d'une autre une propriété cruciale dont le raffinement pourrait fournir des méthodes de résolution meilleures encore.

Pour les chercheurs de la théorie de la NP-complétion, tous les problèmes insolubles par des algorithmes de complexité temporelle polynomiale en la taille du problème sont intraitables. Cette affirmation fait apparaître trois types d'intraitabilité. Le premier concerne les problèmes insolubles algorithmiquement. Pour de tels problèmes, toute discussion sur les algorithmes qui les résolvent est évidemment vaine. Le second rassemble les problèmes de nature si complexe que seuls des algorithmes de complexité temporelle exponentielle peuvent les résoudre. Enfin, le troisième traite des problèmes dont les solutions sont si longues à exprimer qu'il est impossible de les décrire par un algorithme de complexité temporelle polynomiale. La variante du problème du voyageur de commerce dans laquelle on demande de déterminer tous les tours du voyageur de longueur inférieure ou égale à une constante B arbitrairement grande en est un exemple. Ce dernier type d'intraitabilité est particulièrement significatif car il met en évidence le caractère peu réaliste

de la définition du problème traité : sa résolution demande, en effet, beaucoup plus d'information qu'on ne peut en utiliser.

Les classes P et NP appréhendent ce qui précède. Pour le voir, rappelons tout d'abord quelques résultats théoriques très utiles de la théorie de la NP - complétion.

II.2.1. Quelques résultats de la théorie de la NP - complétion

Les classes P et NP introduites au chapitre I vérifient l'inclusion $P \subset NP$. De fait, chaque problème résoluble par un algorithme déterministe de complexité temporelle polynomiale l'est aussi par un algorithme non déterministe de même complexité : il suffit, en effet, d'ignorer le résultat produit par la phase de devinette et de prendre pour algorithme de la phase de choix l'algorithme déterministe qui le résout.

La question de savoir si cette inclusion est stricte reste ouverte. Toutefois, l'expérience conduit à penser que P diffère de NP. L'argument le plus probant en faveur de cette thèse est, sans doute, qu'à ce jour, aucune méthode générale convertissant un algorithme non déterministe en un algorithme déterministe n'est connue. En fait, le meilleur résultat que l'on puisse établir est, pour toute classe de problème de taille n de NP, l'existence d'un polynôme p et d'un algorithme déterministe de complexité temporelle $\mathcal{O}(2^{p(n)})$ qui le résout. Dès lors, la capacité d'un algorithme non déterministe de choisir un nombre exponentiel de possibilités en un temps borné polynomialement tend à penser que les algorithmes non déterministes de complexité temporelle polynomiale sont strictement plus puissants que les algorithmes déterministes de même complexité. Pratiquement, pour beaucoup de problèmes de la classe NP tel que celui du voyageur de commerce ou encore le problème d'isomorphisme de graphe, aucun algorithme les solutionnant en un temps borné polynomialement n'ont pu être trouvés et ce, en dépit d'efforts de beaucoup de chercheurs. Pour ces raisons, l'hypothèse selon laquelle les classes P et NP diffèrent est généralement admise. Cependant, l'absence de preuves indéniables oblige à formuler tous les résultats

en l'incluant dans les hypothèses. Ainsi, on ne démontrera pas qu'une classe de problèmes appartient à NP - P mais que si P diffère de NP, cette classe appartient à NP - P.

L'outil de base le plus utilisé dans cette approche conditionnelle est sans aucun doute la transformation polynomiale. Formellement, une transformation polynomiale d'un langage $L_1 \subset \Sigma_1^*$ dans un langage $L_2 \subset \Sigma_2^*$ est une fonction $f : \Sigma_1^* \to \Sigma_2^*$ qui satisfait aux deux conditions

1) il existe un D - programme de complexité temporelle polynomiale qui la calcule ;
2) pour toute chaîne $x \in \Sigma_1^*$, on a $x \in L_1 \Longleftrightarrow f(x) \in L_2$

Par convention, nous écrirons

$$L_1 \propto L_2$$

s'il existe une transformation polynomiale de L_1 dans L_2 et, pour toute classe de problèmes π_1 et π_2

$$\pi_1 \propto \pi_2$$

s'il existe des schémas d'encodage raisonnables e_1 et e_2 tels que $L(\pi_1, e_1) \propto L(\pi_2, e_2)$. Comme d'habitude, nous oublierons toute référence aux schémas d'encodage et affirmerons qu'une transformation polynomiale entre la classe de problèmes π_1 et la classe de problèmes π_2 est une fonction $f: \pi_1 \to \pi_2$ soumise aux deux propriétés

1) il existe un D - programme de complexité temporelle qui la calcule
2) pour tout problème I, on a $I \in \pi_1 \Leftrightarrow f(I) \in \pi_2$

Bien entendu, l'existence d'une telle transformation entre les classes π_1 et π_2 et la connaissance d'un algorithme résolvant les problèmes de la classe π_1 assurent l'existence d'un algorithme résolvant ceux de la classe π_2. Réciproquement, l'inexistence d' algorithme résolvant les problèmes de la classe π_2 implique, sous l'existence d'une transformation polynomiale entre π_1 et π_2, l'inexistence d'algorithme résolvant les problèmes de la classe π_1. Par suite, on peut interpréter la relation $\pi_1 \propto \pi_2$ en disant que les problèmes de la classe π_2 sont aussi difficiles à résoudre que ceux de la classe π_1.

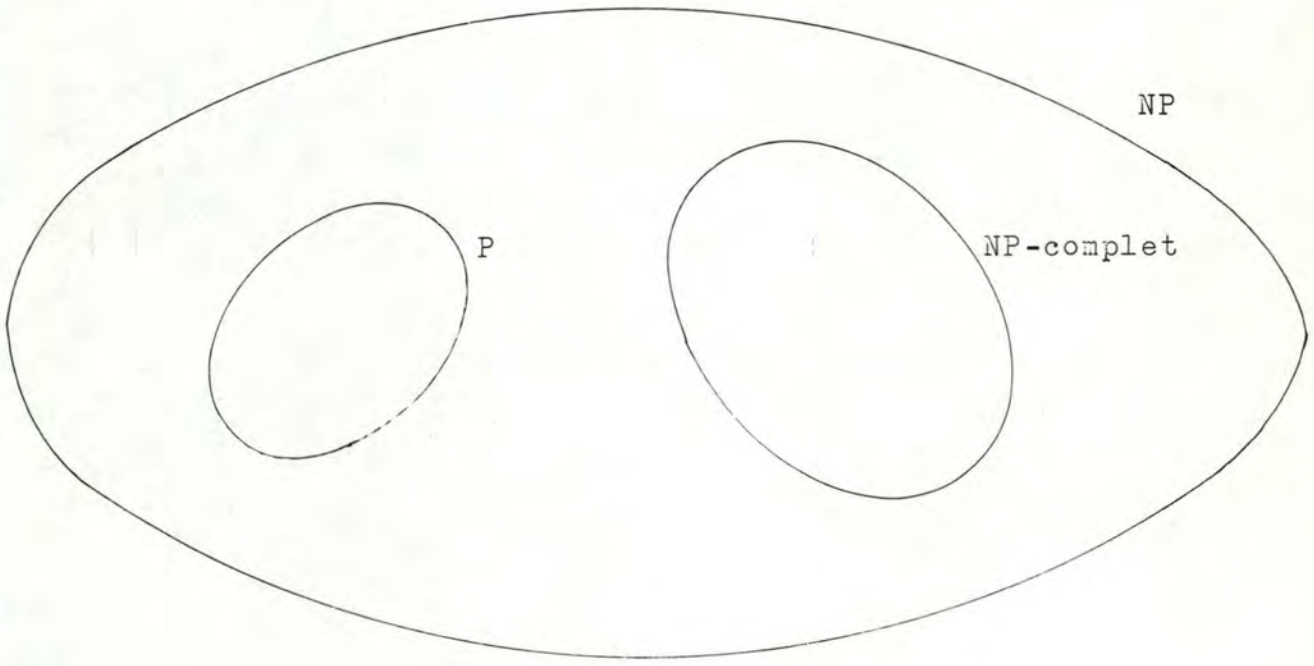
Remarquons qu'un langage L_1 appartient à la classe P s'il existe un langage L_2 qui y appartient et tel que $L_1 \propto L_2$.

Cette notion de transformation polynomiale permet de définir les problèmes NP-complets. Nous dirons, tout d'abord, par définition, d'un langage qu'il est NP - complet s'il appartient à la classe NP et si, pour tout langage L' de NP distinct de L, on a $L' \propto L$. Par analogie, une classe π de problème est NP - complète si elle appartient à la classe NP et si toute classe de problème π' de NP distincte de π vérifie la relation $\pi' \propto \pi$.

La remarque précédente permet d'identifier les problèmes NP - complets comme les problèmes les plus difficiles à résoudre. De fait, si un problème NP - complet peut être résolu par un algorithme de complexité temporelle polynomiale alors il en est ainsi de tous les problèmes de NP. De plus, si tous les problèmes de NP sont intraitables, il en est de même des problèmes NP - complets. Plus précisément, tout problème NP - complet π vérifie l'équivalence

$$\pi \in P \Longleftrightarrow P = NP \ .$$

Supposant que la classe P diffère de la classe NP, on peut donc représenter les trois classes P, NP et des problèmes NP - complets comme suit.



L'intérêt principal de la théorie de la NP - complétion réside, comme le suggère son nom, dans l'étude des problèmes NP - complets. Historiquement, S.A. Cook a tout d'abord établit la NP - complétion du problème de satisfaisabilité. Ensuite, six autres problèmes de base ont été démontrés NP - complets. A présent, des centaines de problèmes sont reconnus NP - complets. Afin d'être complet, nous énonçons les sept problèmes mentionnés ci-avant.

PROBLEME DE SATISFAISABILITE

Soit $U = \{u_1, \ldots, u_m\}$ un ensemble de variables booléennes. Définissons tout d'abord une assignation de vérité sur U comme une fonction $t : U \to \{vrai, faux\}$ et disons qu'une variable booléenne u est vraie (respectivement fausse) sous t si et seulement s'il en est ainsi de t(u). De plus, à toute variable u de U et à toute assignation de vérité t sur U, associons les deux littéraux u_t et $\overline{u}_t$ sur U dont nous dirons par définiton qu'ils sont vrais sous t si et seulement si la variable u est, respectivement, vraie et fausse sous t. Appelons enfin clause sur U la disjonction de littéraux. Nous affirmerons qu'une telle clause est satisfaite par une assignation de vérité si et seulement si au moins un de ses membres est vrai sous cette assignation.

Cela étant, le problème de la satisfaisabilité s'énonce comme suit : soient U un ensemble de varialbles et C une collection de clauses sur U. Existe-t-il une assignation de vérité qui satisfait toutes les clauses de C ?

PROBLEME DE 3-SATISFAISABILITE

Soit C une collection de clauses de cardinal égal à 3 d'un ensemble fini U de variables booléennes. Existe-t-il une assignation de vérité qui satisfait toutes les clauses de C ?

PROBLEME D'ASSORTIMENT TRI-DIMENSIONNEL

Soient X, Y, Z trois ensembles deux à deux disjoints ayant le même nombre q d'éléments. Soit en outre M un ensemble de XxYxZ. Existe-t-il un sous-ensemble M' de M de cardinal égal à q et tel que deux quelconques de ses éléments n'aient aucune coordonnée en commun ?

PROBLEME DE COUVERTURE DE SOMMETS D'UN GRAPHE

Soit un graphe (X,U) dont l'ensemble des sommets X possède au moins k éléments. Existe-t-il un sous-ensemble X' de X de cardinal inférieur ou égal à k tel que tout arc de U ait au moins une extrémité dans X'?

PROBLEME DE CLIQUE

Soit un graphe (X,U) dont l'ensemble des sommets X possède au moins k éléments. Existe-t-il un sous-ensemble X' de X cardinal k au moins tel que deux quelconques de ses sommets soient joints par un arc de U.

PROBLEME DU CIRCUIT HAMILTONIEN

Un graphe (X,U) donné contient-il un circuit hamiltonien ?

PROBLEME DE LA PARTITION

Soient un ensemble fini A et une fonction $S : A \longrightarrow \mathbb{N}$. Existe-t-il un sous-ensemble A' de A tel que $\Sigma_{a \in A'} S(a) = \Sigma_{a \in A - A'} S(a)$?

Cependant, démontrer directement qu'un problème est NP - complet est une tâche ardue. Aussi préfère-t-on appliquer la procédure suivante. Soit π la classe de problèmes dont on veut établir la NP - complétion.

(1) Etablir l'appartenance de la classe π à la classe NP
(2) Sélectionner une classe de problème π' déjà prouvée NP - complète
(3) Construire une transformation f de la classe π' dans π
(4) Démontrer que f établit une transformation polynomiale.

Le choix de la transformation f varie considérablement d'un problème à l'autre. Toutefois, on peut dégager les trois méthodes suivantes.

1. méthode de restriction

Elle consiste à établir une bijection entre une sous-classe π " de la classe π et la classe π' connue comme étant NP-complète qui préserve les réponses oui et non. Toute la difficulté de cette méthode réside bien sûr dans les restrictions à apporter aux problèmes de π pour obtenir π".

2. Méthode de remplacement local

Elle consiste à obtenir les problèmes de la classe π en remplaçant dans l'énoncé générique de la classe π', certaines structures par d'autres, les remplacements des diverses structures étant indépendants les uns des autres.

3. Méthode de construction par composante

Elle consiste à utiliser les constituants de la classe π pour construire certaines composantes qui, combinées, réalisent les problèmes de la classe π'.

Montrons à présent en quoi la théorie de la NP - complétion peut être utile.

II.2.2. L'utilité de la théorie de la NP - complétion.

Ce qui précède suggère que la première question à se poser, lorsqu'un problème à résoudre, algorithmiquement s'entend, apparaît, est

Le problème est-il résoluble par un algorithme de complexité temporelle polynomiale ?

Si la réponse est de toute évidence positive, rien d'autre ne peut être déduit de la théorie de la NP-complétion. Il reste, dans ce cas, à rechercher un tel algorithme, aussi efficace que possible. Par contre, si aucun algorithme de complexité temporelle polynomiale n'est apparent - ce qui est souvent le cas - une deuxième question appropriée est

Le problème est-il NP - complet ?

Dans l'affirmative, une preuve indéniable qu'il ne peut être résolu par un algorithme de complexité temporelle polynomiale est établie. Toutefois, alors même que cela réponde aux deux questions précédentes, beaucoup d'autres peuvent encore être soulevées. Détaillons les principales.

D'une part, le problème analysé peut être mal posé et exiger beaucoup plus que ce que l'on pourrait vouloir a priori, par exemple parce qu'il a été obtenu par raffinement d'un problème durant lequel quelques détails furent écartés. Dans ce cas, une modification de l'énoncé peut altérer suffisamment le problème posé pour le rendre résoluble par un algorithme de complexité temporelle tout en garantissant sa compatibilité avec le problème réel.

D'autre part, il apparaît généralement des sous-problèmes très intéressants évitant le traitement des cas qui entraîne la NP-complétion du problème. C'est en particulier le cas des problèmes de graphe que l'on restreint aux cas des graphes planaires, bipartis, acycliques, ...

Enfin, même si l'on sait le problème résoluble uniquement au moyen d'algorithme de complexité temporelle exponentielle, certains sont préférables à d'autres. L'analyse des résultats de la théorie de la NP-complétion et tout particulièrement de la complexité des sous-problèmes éclaire généralement sur ce point. Ainsi, si un sous-problème obtenu à partir du problème analysé en fixant la valeur d'un paramètre m à 2 s'avère NP-complet, il est exclu de trouver un algorithme de complexité $\mathcal{O}(n^m)$ (qui résout le problème analysé) puisqu'un tel algorithme particularisé au sous-problème où m = 2 est de complexité polynomiale.

Analysons à présent les méthodes de résolution algorithmique des problèmes NP-complet. Il existe essentiellement deux alternatives.

La première consiste à reconnaître le caractère inévitablement exponentiel de la complexité des algorithmes qui résolvent les problèmes NP-complets et tente d'améliorer autant que faire se peut la recherche exhaustive de solution. Les méthodes "Branch and Bound" constituent l'exemple le plus utilisé d'une telle approche. Pour rappel, elles génèrent des solutions partielles ainsi qu'une structure

d'arbre dans lequel elles progressent. Dans ce but, certaines utilisent des méthodes de bornation pour reconnaître les solutions qui ne peuvent probablement pas être étendues, éliminant de la sorte en un pas des branches entières de l'arbre. D'autres incluent, en outre, de la programmation dynamique. Nous donnons en annexe une application de ces méthodes qui visent à résoudre les problèmes de la programmation linéaire en nombres entiers.

Notons encore qu'il est parfois possible de réduire substantiellement le temps de recherche exhaustive pour les cas pathologiques simplement en faisant un choix plus intelligent des objets sur lesquels elle travaille. A titre d'exemple, citons la résolution du problème de la partition en [12].

La deuxième s'applique uniquement aux problèmes d'optimisation. Elle implique une diminution de nos ambitions et se restreint à trouver, à la place d'une solution optimale, une "bonne solution" endéans une quantité acceptable de temps. Les algorithmes qui en découlent sont appelés heuristiques parce qu'ils résultent souvent de procédés empiriques. Bien que les méthodes utilisées pour les construire soient généralement spécifiques du problème traité, on peut toutefois mettre en évidence un principe généralement employé. Il consiste en une recherche de voisinages dans laquelle un ensemble présélectionné d'opérations, spécifiques au problème étudié, est utilisé pour améliorer de façon répétitive une solution initiale. Ce processus se poursuit jusqu'à ce qu'aucune amélioration de la solution courante ne puisse être effectuée.

Les algorithmes construits de cette sorte ont un succès complet en pratique bien qu'une quantité considérable de temps de calcul soit habituellement requise pour obtenir un résultat satisfaisant. En conséquence, il est rarement possible de prédire comment de tels algorithmes s'exécuteront en les analysant auparavant. Aussi préfère-t-on habituellement les évaluer ou les comparer au travers d'une combinaison d'études empiriques et d'arguments intuitifs. Toutefois, il est parfois possible de prouver, par application de la théorie de la NP-complétion, que les solutions trouvées par application de certains algorithmes heuristiques ne différeront jamais de plus d'un certain pourcentage d'une solution optimum. Dans les autres cas, si elles s'en écartent trop, il reste

à améliorer les algorithmes heuristiques par adjonction d'heuristiques sophistiquées et de techniques d'optimisation spécifiques.

Il convient ici d'observer les faits suivants. D'une part, quand elles existent, les bornes déterminant le pourcentage d'erreur de la solution trouvée sont établies de façon à tenir compte de tous les cas de figure possibles et en particulier des cas pathologiques. Par suite, il est fort probable qu'en pratique, l'heuristique fournisse des solutions les plus proches d'une solution optimum. D'autre part, on pourrait être tenté de prendre comme indicateur d'erreur le pourcentage moyen d'erreurs des solutions proposées par l'heuristique. Nous l'avons déjà dit, une telle moyenne n'a de sens que lorsqu'une distribution probabiliste du pourcentage d'erreur a pu être établie, ce qui est rarement le cas. De plus, elle ne renseigne en rien sur le comportement de l'heuristique dans les cas particuliers, alors que les bornes, bien qu'elles ne garantissent pas exactement ce comportement, donnent au moins une idée pessimiste de sa performance.

En conclusion, à notre sens, si l'on désire obtenir autant d'informations que possible sur le comportement d'un heuristique en pratique, il est meilleur de l'analyser d'autant de façons que possible à la fois du point de vue des cas les plus défavorables et du point de vue des cas moyens.

II.3. Conclusion

Les résultats des paragraphes précédents suggèrent le méta-algorithme suivant de construction d'un algorithme de résolution d'une classe de problèmes donnée.

Soit π la classe de problèmes à résoudre.

(1). Si la réponse à l'une des questions

π peut-il être résolu par un algorithme de complexité polynomiale ?

π est-il NP - complet ?

π est-il insoluble algorithmiquement ?

est immédiate alors passer à la phase suivante. Sinon, aller en (3).

(2). Si π peut être résolu par un algorithme de complexité temporelle polynomiale
alors rechercher un tel algorithme aussi efficace que possible, par exemple, en s'appuyant sur les principes du paragraphe II.1.

Si π est NP - complet
alors construire un algorithme de complexité temporelle exponentielle ou, le cas échéant, une heuristique aussi performante que possible, en se basant sur les résultats des deux paragraphes précédents.

Si π ne peut être résolu algorithmiquement
alors abandonner ou modifier, si c'est possible, le problème de façon à obtenir un problème suffisamment proche résoluble algorithmiquement.

(3). Arrivé à cette phase, aucune des trois questions précédentes n'a reçu de réponse. Il serait tentant de concentrer tous les efforts dans une même direction de recherche. Cependant, l'expérience apprend que si l'intuition reste un très bon guide, il lui arrive néanmoins de fausser complètement la recherche d'une solution. Le fait que de nombreux problèmes résolubles par des algorithmes de complexité temporelle polynomiale diffèrent seulement de peu d'autres NP-complets est particulièrement éloquent à ce sujet.

Dès lors, il est préférable de procéder à une analyse tri-directionnelle. Pendant que nous essayons d'une part, de construire un algorithme de complexité temporelle polynomiale, nous tentons, d'autre part, de découvrir

une preuve de NP-complétion ou d'insolubilité algorithmique. Celle des directions que l'on décide d'accentuer à un moment donné dépend, sans doute, de l'état courant des recherches mais à chaque instant, il doit être possible d'inverser la direction de recherche et de prospecter dans une autre.

En fait, cette alternance permet généralement de suggérer des résultats de recherche dans une direction d'autres résultats dans une autre direction. Ainsi la recherche d'un algorithme de complexité temporelle polynomiale peut-elle être à la base d'une preuve de NP-complétion. De même, la recherche d'une telle preuve peut-elle conduire à la découverte d'un algorithme de complexité temporelle polynomiale.

Il est clair que l'application d'une telle approche tri-directionnelle demande un certain esprit imaginatif aussi bien pour construire de telles preuves que pour élaborer de tels algorithmes.

On remarquera le caractère récursif de l'algorithme précédent. La recherche d'un algorithme de complexité temporelle exponentielle ou d'une heuristique dans le cas où π est NP-complet et la résolution d'un nouveau problème "voisin" de π quand il est insoluble algorithmiquement impliquent, en effet, l'analyse de nouveaux problèmes.

CHAPITRE III

ETUDE D'UN CAS : L'ANALYSE DE 7 ALGORITHMES SOLUTIONNANT LE PROBLEME DU FLOT MAXIMUM

A titre illustratif, ce chapitre présente l'étude et la comparaison de 7 algorithmes solutionnant le problème du flot maximum. Il résume les recherches que nous exposons par ailleurs en [13]. Par suite, nous nous bornerons ici à rappeler les quelques concepts fondamentaux utilisés et admettrons les résultats cités sans démonstration. Nous renvoyons le lecteur désireux d'en connaître davantage à l'annexe [13].

III.1. Le problème

Le problème du flot maximum est sans nul doute l'un des problèmes de flot le plus important. Il repose sur les notions de flot et de réseau de transport. Afin de le définir, précisons tout d'abord le sens que nous leur donnons.

1) Un réseau de transport est un triplet $R = (X,U,c)$ soumis aux deux conditions suivantes :

(i) $(X,U) = (\{x_1, \ldots, x_n\}, \{u_1, \ldots, u_m\})$ est un 1-graphe sans boucle dans lequel le sommet x_1, appelé l'entrée du réseau est tel que $d^-(x_1) = 0$ et pour lequel le sommet x_n, appelé la sortie du réseau vérifie l'égalité $d^+(x_n) = 0$;

(ii) $c : U \to \mathbb{R}^+$ est une fonction qui, à tout arc $u \in U$, associe un nombre réel positif $c(u)$ appelé la capacité de l'arc u

2) Un flot d'un réseau de transport $R = (X,U,c)$ est une fonction $f : U \to \mathbb{R}^+$ telle que

$$\text{(i)} \quad \forall u \in U : 0 \leqslant f(u) \leqslant c(u) \qquad (1)$$

$$\text{(ii)} \quad \forall x \in X \setminus \{x_1, x_n\} : \sum_{u \in \omega^-(x)} f(u) = \sum_{u \in \omega^+(x)} f(u) \qquad (2)$$

Le nombre $f(u)$ ainsi associé à l'arc u est appelé le flux dans l'arc u. Par abus de langage nous l'appelons de même si la fonction f viole la propriété (2) tout en vérifiant les inégalités (1).

Dans la suite de ce travail, les notations

R ou (X,U,c)

désignent un réseau de transport dont nous notons systématiquement

X ou $\{x_1, \ldots, x_n\}$; U ou $\{u_1, \ldots, u_m\}$; x_1 ou e; x_n ou t

respectivement l'ensemble des sommets, l'ensemble des arcs, l'entrée et la sortie.

La proposition I.1.4. de l'annexe [13] affirme que l'on a, pour tout flot f du réseau de transport R,

$$\sum_{u \in \omega^+(e)} f(u) = \sum_{u \in \omega^-(t)} f(u) .$$

Cette valeur est encore appelée la valeur du flot f. Elle est notée $|f|$.

Les résultats précédents s'interprêtent aisément. Si l'on assimile les arcs du réseau R à des voies de communication et les flux f(u) à des quantités d'un bien quelconque circulant sur ces voies, la relation (1) affirme que la quantité nette f(u), néces-

sairement positive, du bien considéré qui circule sur u n'excède pas la capacité de l'arc u et la relation (2) qu'il n'y a ni création ni destruction de bien en un sommet quelconque distinct de l'entrée et de la sortie de R. Par ailleurs, $|f|$ représente la quantité de bien qui sort de x_1 vers le réseau et arrive du réseau en x_n.

Cela étant, le problème du flot maximum consiste à rechercher dans l'ensemble des flots que l'on peut définir sur un réseau R un flot dont la valeur est maximum. De nombreux problèmes s'y ramènent. En guise d'exemples, citons le problème de distribution, le problème du trafic maritime, le problème de la bataille de la Marne ainsi que le problème des représentants, tous quatre développés en [13].

Tous les algorithmes présentés dans ce chapitre visent à le résoudre. Néanmoins, certains d'entre eux peuvent, en outre, servir de base à la résolution d'autres problèmes : le problème du flot b-canalisé maximum, le problème du flot maximum de coût minimum, le problème de transfert de Hitchcock.

. Le premier consiste à rechercher, étant donné une fonction b définie sur R telle que $0 \leqslant f(u) \leqslant c(u)$, pour tout $u \in U$, parmi les flots vérifiant les inégalités $b(u) \leqslant f(u) \leqslant c(u)$, pour tout $u \in U$, un flot de valeur maximum.

Le second permet d'appréhender le coût qu'occasionne un flot. Dans ce but, on définit, si d représente une fonction de coût de U dans $\mathbb{R}^+$, le d-coût du flot f comme le nombre

$$\sum_{u \in U} d(u)f(u) .$$

Dans ces conditions, le problème du flot maximum de coût minimum revient, étant donné une telle fonction d, à rechercher dans l'ensemble des flots de valeur maximum que l'on peut définir sur R un flot de d-coût minimum.

Enfin, le problème de transfert de Hitchcock particularise le problème précédent aux réseaux de Hitchcock. Pour rappel, un tel réseau est un réseau de transport $R = (X,U,c)$ vérifiant les propriétés suivantes :

(i) les ensembles X et U peuvent s'écrire sous la forme

$$X = \{e, e_1, \ldots, e_m, t_1, \ldots, t_n, t\} \quad \text{avec } m,n \geqslant 1$$

$$U = \{(e,e_i) : i = 1, \ldots ,m\} \cup \{(t_j,t) : j = 1, \ldots ,n\}$$

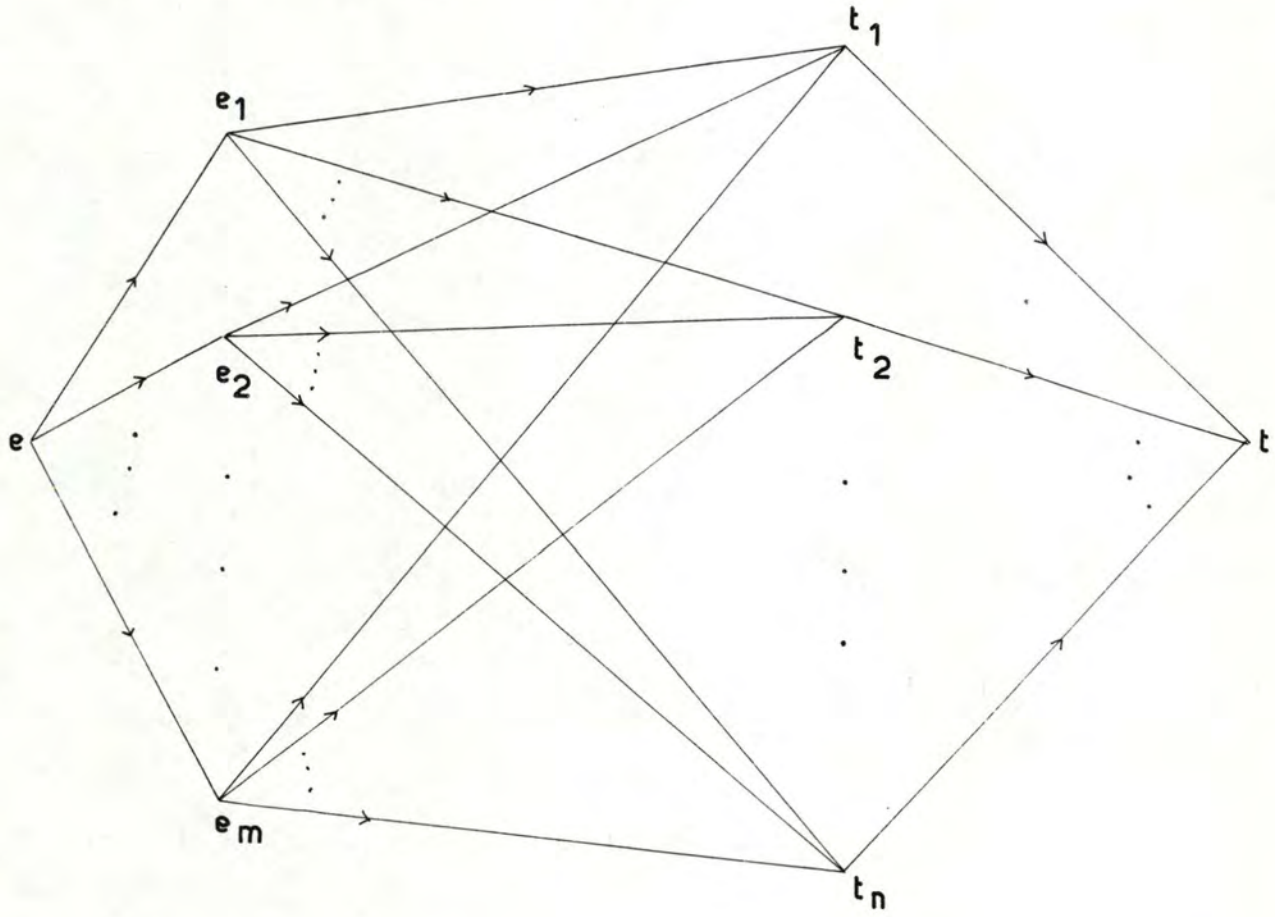
$$\cup \{(e_i,t_j) : i = 1, \ldots ,m;\ j = 1, \ldots ,n\};$$

(ii) pour tout $i \in \{1, \ldots ,m\}$ et tout $j \in \{1, \ldots ,n\}$ on a

$$c(e_i,t_j) = +\infty;$$

(iii) on a $\Sigma_{i=1}^{m} c(e,e_i) = \Sigma_{j=1}^{n} c(t_j,t)$.

Il peut se schématiser comme suit



Le problème de transfert de Hitchcock consiste à résoudre le problème du flot maximum de coût minimum dans un réseau de Hitchcock $(\{e,e_1, \ldots ,e_m,t_1, \ldots ,t_n,t\},U,c)$, la fonction de coût d vérifiant les égalités

$$d(e,e_i) = 0,\ i = 1, \ldots ,m$$

$$d(t_j,t) = 0,\ j = 1, \ldots ,n.$$

Son interprétation est bien conue. Chaque sommet e_i ($i = 1, \ldots, m$) correspond à une source à laquelle $c(e,e_i)$ unités de marchandises sont disponibles et chaque sommet t_j ($j = 1, \ldots, n$) à une destination demandant $c(t_j,t)$ unités. Le coût unitaire du transport de e_i à t_j est $d(e_i,t_j)$. Une solution est recherchée qui minimise le coût de transport.

III.2. Quelques résultats fondamentaux

Les sept algorithmes ci-après reposent sur les résultats suivants, parmi lesquels le théorème du flot maximum et de la coupe mnimum joue un rôle prépondérant. Afin de les exposer, introduisons, tout d'abord, les notions de coupe de réseau et de capacité de coupe de réseau.

1) La coupe du réseau de transport R associée à un sous-ensemble $A \subset X$ tel que $e \notin A$ et $t \in A$ est l'ensemble $\omega^-(A)$ des arcs incidents à A vers l'intérieur.

2) La capacité de la coupe $\omega^-(A)$ est la quantité

$$c(A) = c(\omega^-(A)) = \sum_{u \in \omega^-(A)} c(u)$$

En outre, posons, pour tous sous-ensembles A,B de X, tout sous-ensemble C de U et toute fonction $\alpha : U \to \mathbb{R}$,

$$\alpha(A,B) = \sum_{\substack{a \in A \\ b \in B \\ (a,b) \in U}} \alpha(a,b) ,$$

$$\alpha(C) = \sum_{u \in C} \alpha(u)$$

et désignons par $\bar{A}$ le complémentaire de A dans X.

Les résultats annoncés s'écrivent

LEMME III.2.1. Pour tout flot f et tout ensemble S tel que $e \in S$ et $t \notin S$, on a

$$|f| = f(S,\bar{S}) - f(\bar{S},S).$$

En particulier, pour tout flot f et toute coupe $\omega^-(A)$, on a

$$|f| \leqslant c(\omega^-(A)).$$

THEOREME DU FLOT MAXIMUM ET DE LA COUPE MINIMUM III.2.2. Si un flot f et une coupe $\omega^-(A)$ sont tels que

$$|f| = c(\omega^-(A))$$

alors f est de valeur maximum et $\omega^-(A)$ de capacité minimum.

Nous les admettons sans démonstration.

Venons en à présent aux méthodes de résolution du problème du flot maximum. Plusieurs méthodes ont été proposées parmi lesquelles nous ne retenons que les sept suivantes dont nous pensons qu'elles représentent au mieux les différentes approches connues à ce jour. Les quatre paragraphes suivants les décrivent. Le premier explique les méthodes d'étiquetage , variantes de l'algorithme bien connu de Ford et Fulkerson. Le second analyse l'approche de redistribution de flux proposée par B. Kinariwala et A.G. Rao. Le troisième expose la méthode des préflots développée par A.V. Karzanov. Enfin, le dernier résout ce problème grâce à la programmation linéaire.

III.3. Les méthodes d'étiquetage

Les méthodes d'étiquetage constituent des méthodes itératives qui, à chaque itération, tentent d'améliorer un flot donné (par l'itération précédente, quand elle existe, par l'initialisation, sinon) en un flot de valeur supérieure. A cet effet, elles utilisent les graphes d'écart définis ci-après.

III.3.1. Le graphe d'écart et l'algorithme AUGMENTATION

Soient $R = (X,U,c)$ un réseau de transport et f un flot dans R. Le graphe d'écart $R(f)$ associé à f est le graphe $(X,U(f))$ dont l'ensemble des arcs $U(f)$ est obtenu en associant à chaque arc $u = (x,y) \in U$, l'arc $u = (x,y)$ si $f(u) < c(u)$ et l'arc $\overleftarrow{u} = (y,x)$ si $f(u) > 0$.

L'intérêt du graphe d'écart apparaît dans le théorème suivant.

THEOREME III.3.1.1. Un flot f d'un réseau de transport R est de valeur maximale si et seulement si le graphe d'écart R(f) associé à f ne contient pas de chemin d'extrémités initiale e et terminale t.

Ainsi, pour déterminer si un flot est maximum, il suffit de rechercher dans le graphe d'écart qui lui est associé un chemin d'origine e et d'extrémité terminale t. S'il n'en existe pas, le flot traité est de valeur maximum. Sinon, il est possible de modifier les flux des arcs d'un tel chemin de façon à obtenir un flot de valeur supérieure. L'algorithme AUGMENTATION suivant répond, comme l'établit la proposition II.1.4. de l'annexe [13], à cette spécification.

ALGORITHME AUGMENTATION$(f,(y_1, \ldots ,y_p))$ III.3.1.2. Soient f un flot dans R et $(y_1, \ldots ,y_p)$ un chemin d'extrémités initiale e et terminale t dans R(f).

```
m := +∞;
Pour i variant de 1 à p-1 effectuer
    si (y_i, y_{i+1}) ∈ U et (y_{i+1}, y_i) ∉ U alors
        min := c(y_i, y_{i+1}) - f(y_i, y_{i+1});
    si (y_i, y_{i+1}) ∉ U et (y_{i+1}, y_i) ∈ U alors
        min := f(y_{i+1}, y_i);
    si (y_i, y_{i+1}) ∈ U et (y_{i+1}, y_i) ∈ U alors
        min := c(y_i, y_{i+1}) - f(y_i, y_{i+1}) + f(y_{i+1}, y_i);
    si m > min alors
        m := min;
```

```
Pour tout u ∈ U effectuer ℓ'(u) := ℓ(u);
Pour i variant de 1 à p-1 effectuer
  si (y_i, y_{i+1}) ∈ U et (y_{i+1}, y_i) ∉ U alors
    ℓ'(y_i, y_{i+1}) := ℓ(y_i, y_{i+1}) + m;
  si (y_i, y_{i+1}) ∉ U et (y_{i+1}, y_i) ∈ U alors
    ℓ'(y_i, y_{i+1}) := ℓ(y_i, y_{i+1}) - m;
  si (y_i, y_{i+1}) ∈ U et (y_{i+1}, y_i) ∈ U alors
    ℓ'(y_i, y_{i+1}) := ℓ(y_i, y_{i+1}) + min{m, c(y_i, y_{i+1}) - ℓ(y_i, y_{i+1})};
    ℓ'(y_{i+1}, y_i) := ℓ(y_{i+1}, y_i) - max{0, m - c(y_i, y_{i+1}) + ℓ(y_i, y_{i+1})}
```

Dans la suite, un tel chemin $(y_1, \ldots, y_p)$ sera encore appelé chaîne f-augmentante ou, plus simplement, quand aucun doute sur le flot f n'est permis, chaîne augmentante. De plus, nous appelons augmentation de flot le long de la chaîne f-augmentante α ou, plus simplement, augmentation de flot, l'opération qui consiste à remplacer le flot f par le flot f' résultant de l'exécution de l'algorithme AUGMENTATION appliqué à f et α.

III.3.2. L'algorithme de Ford et Fulkerson

Le paragraphe précédent suggère tout naturellement un algorithme de construction d'un flot de valeur maximum. Partant d'un flot initial f_o, il suffit, en effet, de déterminer de proche en proche grâce à l'algorithme AUGMENTATION III.3.1.2. les éléments $f_o, \ldots, f_N$ d'une suite de flots telle que $|f_i| < |f_{i+1}|$, pour tout $i \in \{1, \ldots, N-1\}$; la suite se terminant lorsqu'un flot maximum a été obtenu.

Cet algorithme a été découvert en premier lieu par L.R. Ford et D.R. Fulkerson en 1962. Il est connu sous le nom d'algorithme de Ford et Fulkerson.

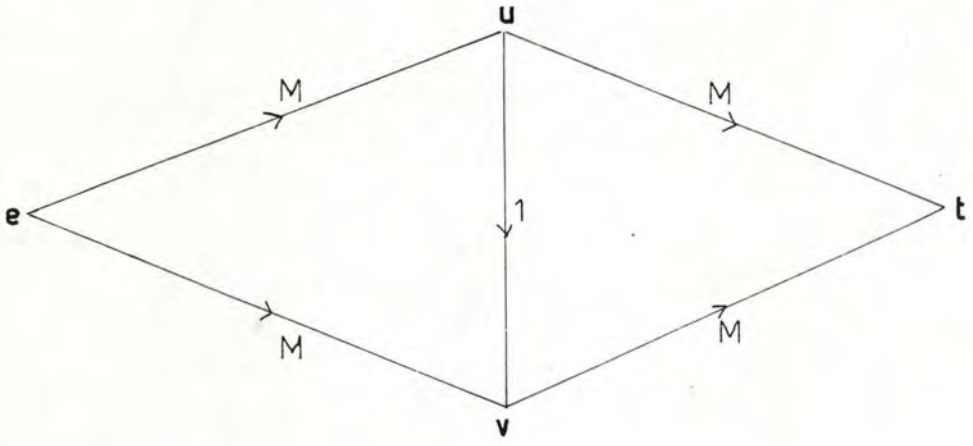
ALGORITHME DE FORD ET FULKERSON III.3.2.1.

Choisir un flot initial f_0 (par exemple, en posant $f_0(u) = 0$, pour tout $u \in U$);
$k := 0$;
Tant qu'il existe une chaîne f_k - augmentante dans $R(f_k)$ répéter
  choisir une telle chaîne, soit $(y_1, \ldots, y_p)$;
  $f_{k+1} := AUGMENTATION(f_k, (y_1, \ldots, y_p))$;
  $k := k + 1$.

Il détermine bien sûr, après exécution, un flot de valeur maximum. Toutefois, sa convergence requiert l'intégrité des capacités du réseau ainsi que des flux constituant le flot initial f_0.

Les pages II.6. à II.13. de l'annexe 13 montrent comment l'adapter de façon à ce qu'il puisse résoudre le problème du flot b-canalisé maximum. De plus, les paragraphes II.6., II.7. et II.8. du même ouvrage le transforment pour qu'il résolve les problèmes du flot maximum de coût minimum et de transfert de Hitchcock.

Il n'indique cependant d'aucune manière la façon de déterminer les chemins $(y_1, \ldots, y_p)$ joignant e à t dans le réseau d'écart. Plusieurs méthodes ont été proposées : nous en avons retenu quatre donnant chacune naissance à une des méthodes d'étiquetage exposées ci-après. Il convient ici de remarquer que le choix d'une telle méthode est particulièrement important; un choix peu heureux peut, en effet, entraîner l'exécution d'un nombre considérable d'itérations. Dans l'exemple suivant



où M désigne un entier strictement positif et où les valuations des arcs définissent la capacité de ceux-ci, l'exécution de l'algorithme III.3.2.1. peut entraîner 2M (soit la valeur maximum d'un flot) itérations de la boucle *Tant que* ... si le procédé de recherche de chemin dans R(f) conduit à sélectionner alternativement les chemins (e,u,v,t) et (e,v,u,t).

III.3.3. La méthode de recherche en profondeur d'abord

La première méthode est basée sur une exploration en profondeur d'abord des sommets d'un graphe. De ce fait, les sommets sont visités selon le principe

- *Ayant déjà visité le sommet* $x_j \in X$, *se déplacer sur un arc* $(x_j, x_k) \in U$ *incident vers l'extérieur à* x_j.
- *Si l'on a déjà visité le sommet* x_k, *revenir en* x_j *et choisir un autre arc incident vers l'extérieur à* x_j;
  sinon, visiter le sommet x_k *et appliquer le processus ainsi défini de manière récursive à* x_k.
- *Si tous les arcs incidents vers l'extérieur à* x_j *ont été explorés, revenir au sommet* $x_i \in X$ *qui a permis d'atteindre* x_j, *par l'arc* (x_i, x_j), *et continuer à explorer les arcs incidents vers l'extérieur à* x_i.

L'algorithme ci-dessous en découle. Soit f le flot courant à augmenter. On y tente, tout d'abord, de déterminer une chaîne f-augmentante. Dans ce but, une procédure récursive RECHERCHE_CHAINE(x) de spécification

> Si le sommet x est visité, alors RECHERCHE_CHAINE(x) recherche dans le sous-graphe de R(f) engendré par l'ensemble des sommets non encore visités uni à {x}, un chemin joignant x à t

a été développée. A l'issue de l'appel RECHERCHE_CHAINE(e), deux cas peuvent se présenter.

Dans le premier, le sommet t a pu être marqué et, par suite, une chaîne f-augmentante α a été découverte. On en déduit alors de même qu'en qu'en III.3.1.2. un flot de valeur strictement supérieure. Toutefois, afin d'améliorer l'efficacité de l'algorithme développé, le nombre m intervenant dans l'algorithme AUGMENTATION III.3.1.2. est calculé lors de la recherche de la chaîne α. Il reste enfin à essayer d'améliorer le nouveau flot obtenu grâce à une itération supplémentaire.

Dans le deuxième, le sommet t n'a pu être marqué. Il s'ensuit qu'aucune chaîne f-augmentante n'existe. Le flot courant est dès lors maximum.

Les structures de données suivantes sont utilisées. A chaque sommet x, on associe

- une valeur booléenne, *marqué(x)*, indiquant si le sommet x a été visité,
- un entier, *min(x)*, permettant de calculer m,
- un sommet, *pred(x)*, déterminant le sommet à partir duquel x a été visité.

L'application *pred* : $X \to X$ détermine, bien entendu, pour tout sommet x marqué un chemin dans R(f) joignant e à x. Le nombre *min(x)* vaut alors le minimum des valuations des arcs de ce chemin, la valuation de l'arc $u = (x,y)$ étant

$c(u) - f(u)$ si $u \in U$ et $\overleftarrow{u} \notin U$,

$f(\overleftarrow{u})$ si $u \notin U$ et $\overleftarrow{u} \in U$,

$c(u) - f(u) + f(\overleftarrow{u})$ si $u \in U$ et $\overleftarrow{u} \in U$,

où $\overleftarrow{u} = (y,x)$. En particulier, *min(t)* vaut m si t est marqué.

ALGORITHME RECHERCHE_CHAINE(s) III.3.3.1. Soit s un sommet de X.

```
Pour tout x∈X effectuer
  si marqué(t) = non et si marqué(x) = non et si
  { ( (s,x)∈U et f(s,x) < c(s,x) ) ou ( (x,s)∈U et f(x,s) > 0 ) }
    alors
      marqué(x) := oui;
      pred(x) := s;
      si (s,x)∈U et (x,s)∉U alors
        min(x) := min{min(s), c(x,s) - f(s,x)};
      si (s,x)∉U et (x,s)∈U alors
        min(x) := min{min(s), f(x,s)};
      si (s,x)∈U et (x,s)∈U alors
        min(x) := min{min(s), c(s,x) - f(s,x) + f(x,s)};
      RECHERCHE_CHAINE(x);
Si marqué(t) = non alors
  Pour tout x∈X effectuer
    si pred(x) = s alors
      marqué(x) := non.
```

ALGORITHME DE FORD ET FULKERSON III.3.3.2.

```
Choisir un flot initial f0;
f := f0;
Pour tout x∈X \ {e} effectuer marqué(x) := non;
marqué(e) := oui;
min(e) := +∞;
RECHERCHE_CHAINE(e);
Tant que marqué(t) = oui répéter
  (* amélioration du flot actuel *)
  s := t;
  ps := pred(s);
  répéter
    si (ps,s)∈U et (s,ps)∉U alors
      f(ps,s) := f(ps,s) + min(t);
    si (ps,s)∉U et (s,ps)∈U alors
      f(s,ps) := f(s,ps) - min(t);
```

```
    si (ps,s)∈U et (s,ps)∈U alors
      f(ps,s) := f(ps,s) + min{min(t), c(ps,s) - f(ps,s)};
      f(s,ps) := f(s,ps) - max{0,min(t) - c(ps,s) + f(ps,s)};
    s := ps;
  tant que s ≠ e;
(* recherche d'une chaîne f-augmentante *)
Pour tout x∈X\{e} effectuer marqué(x) := non;
marqué(e) := oui;
min(e) := +∞;
RECHERCHE_CHAINE(e).
```

Le théorème II.3.3. de l'annexe [13] détermine le nombre d'itérations dans la boucle *Tant que marqué(t) = oui ...* : si les capacités des arcs du réseau de transport R sont entières et si M désigne la valeur d'un flot maximum dans R, l'algorithme III.3.3.2. exécuté à partir d'un flot initial à valeurs entières converge après au plus M augmentations de flot.

III.3.4. La méthode de recherche en largeur d'abord

Dans une deuxième approche, la recherche d'une chaîne f-augmentante s'effectue selon une exploration en profondeur d'abord des sommets du réseau d'écart.

Rappelons ici que, suivant ce mode d'exploration, les sommets d'un graphe sont visités conformément à l'algorithme

```
Choisir un sommet x;
num(x) := 1;
i := 1;
Tant qu'il existe un sommet numéroté i répéter
  numéroter i + 1 tout sommet non encore visité adjacent à
    un sommet déjà marqué i;
  i := i + 1.
```

où *num(x)* est un numéro associé à chaque sommet x.

Ainsi, si l'on appelle "scanner un sommet x" l'opération qui consiste à visiter les sommets non encore visités adjacents au sommet x, l'exploration des sommets en largeur d'abord s'effectue selon le principe "premier visité, premier scanné" par opposition au processus d'exploration en profondeur d'abord s'effectuant quant à lui selon la règle "dernier visité,premier scanné".

En particularisant, cette exploration des sommets en largeur d'abord à la recherche d'un chemin d'extrémités initiale e et terminale t, l'algorithme de Ford et Fulkerson III.3.2.1. s'écrit

ALGORITHME DE FORD ET FULKERSON III.3.4.1.

```
Choisir un flot initial f0;
f := f0;
s := e;
Répéter
  (* amélioration du flot courant *)
  Tant que s ≠ e répéter
    ps := pred(s);
    si (ps,s) ∈ U et (s,ps) ∉ U alors
      f(ps,s) := f(ps,s) + min(t);
    si (ps,s) ∉ U et (s,ps) ∈ U alors
      f(s,ps) := f(s,ps) - min(t);
    si (ps,s) ∈ U et (s,ps) ∈ U alors
      f(ps,s) := f(ps,s) + min{min(t), c(ps,s) - f(ps,s)};
      f(s,ps) := f(s,ps) - max{0, min(t) - c(ps,s) + f(ps,s)};
    s := ps;
  (* recherche d'une chaîne f-augmentante *)
  Pour tout x ∈ X\{e} effectuer num(x) := 0;
  num(e) := 1;
  i := 1;
  min(e) := +∞;
  Répéter
    fin_itérations := vrai;
    Pour tout x ∈ X effectuer
      si num(x) = i alors
```

```
Pour tout y ∈ X effectuer
  si (num(y) = 0)  et  ( { (x,y) ∈ U et f(x,y) < c(x,y) }
  ou  { (y,x) ∈ U et f(y,x) > 0 } )  alors
    num(y) := i+1;
    pred(y) := x;
    fin_itérations := faux;
    si (x,y) ∈ U et (y,x) ∉ U alors
      min(y) := min{min(x), c(x,y) - f(x,y)};
    si (x,y) ∉ U et (y,x) ∈ U alors
      min(y) := min{min(x), f(y,x)};
    si (x,y) ∈ U et (y,x) ∈ U alors
      min(y) := min{min(x), c(x,y) - f(x,y) + f(y,x)};
  i := i + 1;
tant que (fin_itérations = faux) et (num(t) = 0);
s := t;
tant que num(t) ≠ 0.
```

Les structures de données suivantes y sont manipulées : à chaque sommet x, on associe

- un numéro, *num(x)*,
- un sommet, *pred(x)*, déterminant le sommet à partir duquel le sommet x a été visité.

Bien sûr, comme dans l'algorithme III.3.3.2. et pour les mêmes raisons d'efficacité, le nombre m intervenant dans l'algorithme AUGMENTATION III.3.1.2. est calculé lors de la construction de la chaîne augmentante.

Le théorème II.4.10. de l'annexe [13] établit sa complexité temporelle théorique. Il affirme que le nombre d'augmentations de flot effectuées lors d'une exécution est

$$\frac{n\,(n-1)\,(n+2)}{4}\,.$$

En particulier, la complexité temporelle théorique de l'algorithme III.3.4.1. est $\mathcal{O}(n^6)$ au plus. On notera que la démonstration du théorème II.4.10. établit la convergence de cet algorithme même s'il est appliqué à un réseau de transport à capacités non entières.

III.3.5. La méthode d'Edmonds et Karp

Dans leur méthode, J. Edmonds et R.M. Karp suggèrent de rechercher parmi l'ensemble des chaînes augmentantes, la chaîne fournissant l'augmentation la plus grande possible de flot. Dans ce but, ils proposent l'algorithme III.3.5.1.. Les structures de données suivantes y sont utilisées. A chaque sommet $x \in X$, on associe un sommet *pred(x)* déterminant le sommet à partir duquel x a été visité. De plus, deux sous-ensembles *ens_sommets_marqués* et *ens_sommets_non_marqués* constituent après chaque itération une partition de X. Ils mémorisent respectivement l'ensemble des sommets déjà visités et l'ensemble des sommets non encore visités.

ALGORITHME III.3.5.1. Soit f un flot d'un réseau de transport.

```
ens_sommets_marqués := {e};
ens_sommets_non_marqués := X\{e};
Répéter
  eval := +∞;
  fin_itérations := oui;
  Pour tout x ∈ ens_sommets_marqués effectuer
    Pour tout y ∈ ens_sommets_non_marqués effectuer
      si { (x,y) ∈ U et f(x,y) < c(x,y) } ou
         { (y,x) ∈ U et f(y,x) > 0 } alors
        si (x,y) ∈ U et (y,x) ∉ U alors
          eval' := c(x,y) - f(x,y);
        si (x,y) ∉ U et (y,x) ∈ U alors
          eval' := f(y,x);
        si (x,y) ∈ U et (y,x) ∈ U alors
          eval' := c(x,y) - f(x,y) + f(y,x);
        si eval' < eval alors
          x_retenu := x;
          y_retenu := y;
          fin_itérations := non;
          eval := eval';
  Si fin_itérations = non alors
    pred(y_retenu) := x_retenu;
    ens_sommets_marqués := ens_sommets_marqués ∪ {y_retenu};
    ens_sommets_non_marqués := ens_sommets_non_marqués \
                                              {y_retenu};
tant que (fin_itérations = non) et (t ∈ ens_sommets_non_marqués).
```

Cet algorithme, appliqué à un flot f, converge et vérifie, après exécution, l'assertion : le sommet t appartient à l'ensemble *ens_sommets_marqués* si et seulement s'il existe une chaîne f-augmentante. De plus, s'il en est ainsi, l'application $pred : X \to X$ détermine une telle chaîne fournissant, en outre, une augmentation maximum de flot.

Conceptuellement, il repose sur la notion de goulot d'une chaîne augmentante. Précisons tout d'abord cette notion. Soit f un flot d'un réseau de transport R et ξ une chaîne f-augmentante. Un arc $v \in \xi$ est un goulot de ξ s'il vérifie l'égalité

$$ev(v) = \min_{u \in \xi} ev(u)$$

où, pour tout arc $u = (x,y) \in U(F)$, ev(u) est défini par

$$ev(u) = \begin{cases} c(x,y) - f(x,y) & \text{si } (x,y) \in U \text{ et } (y,x) \notin U \\ f(y,x) & \text{si } (x,y) \notin U \text{ et } (y,x) \in U \\ c(x,y) - f(x,y) + f(y,x) & \text{si } (x,y) \in U \text{ et } (y,x) \in U \end{cases}$$

L'idée maîtresse reste, tout comme dans la méthode de recherche en largeur d'abord, d'atteindre de proche en proche, partant de l'entrée e, la sortie t. Toutefois, à chaque itération, l'algorithme ne retient plus tous les sommets non visités adjacents aux sommets visités à l'itération précédente, mais un arc qui, parmi les arcs d'origine visitée et d'extrémité terminale non encore retenue, fournirait, s'il était goulot d'une chaîne augmentante, une augmentation de flot maximum.

L'algorithme d'Edmonds et Karp résulte de l'adaptation de l'algorithme général de Ford et Fulkerson III.3.2.1. à la recherche d'une telle chaîne augmentante.

ALGORITHME D'EDMONDS ET KARP III.3.5.1.

Choisir un flot initial f_0 ;
$f := f_0$;
$s := e$;
Répéter

```
(* amélioration du flot courant f *)
Tant que s ≠ e effectuer
  ps := pred(s);
  si (ps,s)∈ U et (s,ps)∉ U alors
    f(ps,s) := f(ps,s) + min(t);
  si (ps,s)∉ U et (s,ps)∈ U alors
    f(s,ps) := f(s,ps) - min(t);
  si (ps,s)∈ U et (s,ps)∈ U alors
    f(ps,s) := f(ps,s) + min{min(t), c(ps,s) - f(ps,s)};
    f(s,ps) := f(s,ps) - max{0, min(t) - c(ps,s) + f(ps,s)};
  s := ps;
(* recherche d'une chaîne f-augmentante *)
ens_sommets_marqués := {e};
ens_sommets_non_marqués := X\{e};
Répéter
  eval := +∞;
  fin_itérations := oui;
  Pour tout x ∈ ens_sommets_marqués effectuer
    Pour tout y ∈ ens_sommets_non_marqués effectuer
      si { (x,y)∈ U et c(x,y) < f(x,y) } ou
         { (y,x)∈ U et f(y,x) > 0 }   alors
        si (x,y)∈ U et (y,x)∉ U alors
          eval' := c(x,y) - f(x,y);
        si (x,y)∉ U et (y,x)∈ U alors
          eval' := f(y,x);
        si (x,y)∈ U et (y,x)∈ U alors
          eval' := c(x,y) - f(x,y) + f(y,x);
        si eval' < eval alors
          x_retenu := x;
          y_retenu := y;
          fin_itération := non;
          eval := eval';
  Si fin_itérations = non alors
    pred(y_retenu) := x_retenu;
```

```
si (x_retenu, y_retenu) ∈ U et (y_retenu, x_retenu) ∉ U alors
  min(y_retenu) := min{min(x_retenu), c(x_retenu, y_retenu)
                                  - f(x_retenu, y_retenu)} ;
si (x_retenu, y_retenu) ∉ U et (y_retenu, x_retenu) ∈ U alors
  min(y_retenu) := min{min(x_retenu),
                                  f(y_retenu, x_retenu)} ;
si (x_retenu, y_retenu) ∈ U et (y_retenu, x_retenu) ∈ U alors
  min(y_retenu) := min{min(x_retenu), c(x_retenu, y_retenu)
                                  - f(x_retenu, y_retenu)
                                  + f(y_retenu, x_retenu)} ;
ens_sommets_marqués := ens_sommets_marqués ∪ {y_retenu};
ens_sommets_non_marqués := ens_sommets_non_marqués \
                                  {y_retenu} ;
tant que (fin_itérations = non) et (t ∉ ens_sommets_marqués);
s := t;
tant que (t ∈ ens_sommets_marqués).
```

Le théorème II.5.4. de l'annexe [13] précise le nombre maximum d'augmentations de flots qui y sont effectuées. Si M désigne un entier positif tel que, pour toute partition des sommets de X en deux sous-ensembles S et $\bar{S}$, le nombre d'arcs ayant une extrémité dans S et une autre dans $\bar{S}$ est plus petit ou égal à M et si, en outre, f* est un flot de valeur maximum, l'algorithme précédent, appliqué à un réseau de transport à capacités entières converge, après au plus

$$1 + \log_{M/M-1} |f^*|$$

augmentations de flots.

III.3.6. La méthode de Dinic

La méthode de Dinic repose sur la notion de réseau de référence. Définissons la tout d'abord. Le réseau de référence associé au réseau de transport R = (X,U,c) et au flot f de ce réseau est le réseau $(X, U_{r,f}, c)$ dont l'ensemble des arcs $U_{r,f}$ est constitué de l'ensemble des arcs composant les chemins de R(f) d'extrémité initiale e et de longueur inférieure ou égale à la distance de e à t dans R(f). Bien

sûr, l'existence d'un tel réseau suppose l'existence d'une chaîne augmentante.

Plutôt que de construire, comme dans les méthodes précédentes, à chaque itération le réseau d'écart du flot courant, E.A. Dinic propose d'effectuer certaines augmentations successives de flot sur un même réseau, le réseau de référence associé au flot courant. Sa méthode repose donc sur une suite de phases.

Chacune commence par construire le réseau de référence associé au flot courant. Ceci peut, comme le montre l'algorithme III.3.6.1., s'effectuer grâce à une exploration en largeur d'abord des sommets du réseau d'écart que l'on arrête lorsque la sortie a pu être marquée. Bien sûr, si la construction de ce réseau échoue c'est-à-dire si l'algorithme précédent se termine en assignant à la variable *echec* la valeur faux, le flot courant est de valeur maximum. Dans le cas contraire, grâce à une exploration en profondeur des sommets du réseau de référence, une chaîne augmentante est déterminée et le flot courant augmenté. Les arcs du réseau de référence n'appartenant pas au réseau d'écart du nouveau flot sont éliminés et une nouvelle recherche en profondeur d'abord d'une chaîne augmentante commence ... . Et ainsi de suite jusqu'à ce que la sortie t soit devenue inaccessible. A ce moment, une nouvelle phase commence.

L'algorithme de Dinic qui en découle repose encore sur les deux algorithmes auxiliaires suivants. Le premier, RECHERCHE_CHEMIN(s), vise à déterminer à partir d'un sommet s, un chemin entre s et t dans le réseau de référence décrit par les ensembles $succ(x)$ ($x \in X$) des successeurs de x dans ce réseau. Le second, AUGMENTATION_FLOT(f), a pour but d'augmenter le flot f le long de la chaîne augmentante définie par l'application $suiv$ et d'éliminer les arcs du réseau de référence n'appartenant pas au graphe d'écart du nouveau flot.

ALGORITHME CONSTRUCTION_RESEAU_REFERENCE(f,echec) III.3.6.1.
Soit f un flot du réseau de transport R = (X,U,c).

```
Pour tout x ∈ X\{e} effectuer
  num(x) := 0;
  succ(x) := ϕ;
num(e) := 1;
i := 1;
```

```
Répéter
  fin_itérations := vrai;
  Pour tout x∈X effectuer
    si num(x) = i alors
      Pour tout y∈X effectuer
        si { ( (x,y)∈U et f(x,y) < c(x,y) )   ou
             ( (y,x)∈U et f(y,x) > 0 ) }   et  num(y)∈{0,i+1}
        alors
          num(y) := i + 1;
          succ(x) := succ(x) ∪ {y};
          fin_itérations := faux;
  i := i + 1;
tant que (fin_itérations = faux) et (num(t) = 0);
Si num(t) = 0
  alors echec := vrai
  sinon echec := faux.
```

ALGORITHME RECHERCHE_CHEMIN(s) III.3.6.2. Soit s un sommet.

```
Pour tout x∈succ(s) effectuer
  si t_marqué = non alors
    si x = t
    alors
      t_marqué := oui;
      suiv(s) := t;
    sinon
      suiv(s) := x;
      RECHERCHE_CHEMIN(x).
```

ALGORITHME AUGMENTATION_FLOT(f) III.3.6.3. Soit f un flot.

```
x := e;
min := +∞;
Répéter
  y := suiv(x);
  si (x,y)∈U et (y,x)∉U alors
    m := c(x,y) - f(x,y);
  si (x,y)∉U et (y,x)∈U alors
    m := f(y,x);
  si (x,y)∈U et (y,x)∈U alors
    m := c(x,y) - f(x,y) + f(y,x);
```

```
    si min > m alors
      min := m;
    x := y;
  tant que x ≠ t;
  x := e;
  Répéter
    y := suiv(x);
    si (x,y) ∈ U et (y,x) ∉ U alors
      f(x,y) := f(x,y) + min;
      si f(x,y) = c(x,y) alors succ(x) := succ(x) \ {y};
    si (x,y) ∉ U et (y,x) ∈ U alors
      f(y,x) := f(y,x) - min;
      si f(y,x) = 0 alors succ(x) := succ(x) \ {y};
    si (x,y) ∈ U et (y,x) ∈ U alors
      f(x,y) := f(x,y) + min{min, c(x,y) - f(x,y)};
      f(y,x) := f(y,x) - max{0, min - c(x,y) + f(x,y)};
      si f(x,y) = c(x,y) et f(y,x) = 0 alors
        succ(x) := succ(x) \ {y};
    x := y;
  tant que x ≠ t.
```

ALGORITHME DE DINIC III.3.6.4.

```
  Choisir un flot initial f0;
  f := f0;
  Répéter
    CONSTRUCTION_RESEAU_REFERENCE(f, echec);
    si echec = faux alors
      Répéter
        t_marqué := non;
        RECHERCHE_CHEMIN(e);
        si t_marqué = oui alors
          AUGMENTATION_FLOT(f);
      tant que t_marqué = oui;
  tant que echec = faux.
```

Appliqué aux réseaux de transport à capacités entières et à partir d'un flot initial à valeurs entières, l'algorithme de Dinic III.3.6.4. possède une complexité temporelle théorique de $\mathcal{O}(n^4)$ au plus. Toutefois, les réseaux à capacités entières offrent une complexité meilleure encore : le théorème II.9.20. établit, en effet, dans ce cas, qu'il possède une complexité temporelle théorique de $\mathcal{O}(n^{8/3})$ au plus.

III.4. La méthode de redistribution de flux

La méthode de redistribution de flux repose essentiellement sur l'augmentation ou la diminution locale de flux, les équations de conservation de la somme des flux aux sommets étant temporairement délaissées. Toutefois, à chaque fonction $f : U \to \mathbb{R}^+$ elle associe une fonction de poids w_f dont le but est de mémoriser en tout sommet la différence entre la somme des valeurs f(u) des arcs qui lui sont incidents vers l'extérieur et la somme des mêmes valeurs correspondant aux arcs qui lui sont incidents vers l'intérieur. Cette dernière est encore définie par les relations

$$w_f(x) = f(x,X) - f(X,x), \quad \forall x \in X.$$

Tout chemin du réseau R composé d'arcs donnant à f une valeur strictement positive dont l'origine x possède un poids $w_f(x)$ strictement positif et l'extrémité terminale y un poids strictement négatif est appelé f-chemin d'élimination ou, plus simplement, chemin d'élimination. Un tel couple de sommets (x,y) est encore appelé f-paire d'élimination ou, plus simplement, paire d'élimination.

L'algorithme ELIMINATION ci-dessous, appliqué à un sous-ensemble Y de X détermine puis supprime les paires d'élimination de Y. On procède comme suit. On tente tout d'abord, partant d'un sommet de poids strictement positif, de tracer un chemin d'élimination ayant ce sommet pour origine, puis on réduit les flux des arcs qui le composent. Ce processus se répète jusqu'à ce qu'il n'y ait plus de sommet dans Y de poids strictement positif ou qu'il n'existe plus de chemin d'élimination. Dans ce dernier cas, s'il subsiste des

sommets de poids strictement positif, l'algorithme ELIMINATION identifie en plus l'ensemble Z des sommets de Y extrémités terminales d'un chemin composé d'arcs donnant à la fonction courante f une valeur strictement positive et dont l'origine possède un poids strictement positif.

ALGORITHME ELIMINATION(Y,Z) III.4.1. Soient f une fonction du réseau de transport (X,U,c) telle que $0 \leqslant f(u) \leqslant c(u)$, pour tout $u \in U$ et Y un sous-ensemble de X.

```
Z := φ;
lg_chemin := 0;
y_aux := y;
Tant que {y ∈ y_aux : w_f(y) > 0} ≠ φ répéter
  si lg_chemin = 0 alors
    choisir x ∈ X tel que w_f(x) > 0;
    chemin(1) := x;
    lg_chemin := 1;
  RECHERCHE_CHEMIN(chemin, lg_chemin, état_chemin);
  si état_chemin = bloqué alors
    Z := Z ∪ {chemin(lg_chemin)};
    y_aux := y_aux \ {chemin(lg_chemin)};
    lg_chemin := lg_chemin - 1;
  si état_chemin = terminé alors
    θ := min{w_f(chemin(1)), w_f(chemin(lg_chemin)),
                  min        f(chemin(i), chemin(i+1))};
           1 ⩽ i < lg_chemin
    w_f(chemin(1)) := w_f(chemin(1)) - θ;
    w_f(chemin(lg_chemin)) := w_f(chemin(lg_chemin)) + θ;
    Pour i variant de 1 à (lg_chemin - 1) effectuer
      f(chemin(i), chemin(i+1)) := f(chemin(i), chemin(i+1)) - θ;
    lg_chemin := 0.
```

où la procédure RECHERCHE_CHEMIN(chemin,lg_chemin,état_chemin) est

ALGORITHME RECHERCHE_CHEMIN(chemin,lg_chemin,état_chemin)

III.4.2. Soient une fonction f du réseau de transport R = (X,U,c) telle que $0 \leqslant f(u) \leqslant c(u)$, pour tout $u \in U$ et un chemin de ce réseau de longueur *lg_chemin* mémorisé dans le tableau *chemin*.

```
état_chemin := normal;
Tant que ( ∃ y ∈ Y_aux : ℓ(chemin(lg_chemin),y) > 0) et
          (état_chemin ≠ terminé) répéter
 | choisir y ∈ Y_aux tel que ℓ(chemin(lg_chemin),y) > 0;
 | si w_ℓ(y) < 0
 |   alors
 |   | état_chemin := terminé
 |   sinon
 |   |  si y ∈ chemin
 |   |  |  alors
 |   |  |  | indice := min{i : 1 ⩽ i < lg_chemin et y = chemin(i)};
 |   |  |  | δℓ := min{ℓ(chemin(i),chemin(i+1) :
 |   |  |  |                    indice ⩽ i < lg_chemin};
 |   |  |  | Pour i variant de indice à lg_chemin - 1 effectuer
 |   |  |  |   | ℓ(chemin(i),chemin(i+1)) :=
 |   |  |  |   |      ℓ(chemin(i),chemin(i+1)) - δℓ;
 |   |  |  | lg_chemin := indice;
 |   |  |  sinon
 |   |  |  | lg_chemin := lg_chemin + 1;
 |   |  |  | chemin(lg_chemin) := y;
Si état_chemin ≠ terminé alors
 | état_chemin := bloqué.
```

L'utilité de ces algorithmes et des notions précédentes provient de la remarque suivante. Supposons avoir résolu le problème : obtenir un sous-ensemble X' du réseau (X,U,c) et une fonction f tels que

$0 \leqslant f(u) \leqslant c(u), \quad \forall u \in U,$

$e \in X', \quad t \notin X',$

$f(X',\bar{X}') = c(X',\bar{X}'), \qquad (1)$

$f(\bar{X}',X') = 0, \qquad (2)$

$$w_f(x) \leqslant 0, \quad \forall x \in X' \setminus \{e\},$$

et

$$w_f(y) \geqslant 0, \quad \forall y \in \bar{X}' \setminus \{t\}.$$

Comme $f(\bar{X}', X')$ est nul, les poids négatifs dans X' peuvent être associés aux chemins d'élimination d'origine e et d'extrémité terminale distincte de t et les poids positifs aux chemins d'élimination d'origine distincte de e et d'extrémité terminale t. Eliminons grâce à l'algorithme III.4.1. ces classes de chemins. Comme ceci ne modifie en rien les flux des arcs de $X' \times \bar{X}'$ et de $\bar{X}' \times X'$, les équations (1) et (2) restent valables. Par suite, le théorème du flot maximum et de la coupe minimum affirme que la coupe associée à $\bar{X}'$ est minimum et que le flot obtenu est de valeur maximum.

Reste donc, pour résoudre le problème du flot maximum, à déterminer un tel ensemble X'. Or c'est justement le but des itérations de la méthode de redistribution des flux que de construire un tel ensemble X' vérifiant en outre les relations

$$w_f(x) = 0, \quad \forall x \in \bar{X}' \setminus \{t\}.$$

Toutefois, avant de les amorcer une phase d'initialisation est requise.

la phase d'initialisation

Soit f une fonction de U dans $\mathbb{R}^+$ telle que $0 \leqslant f(u) \leqslant c(u)$. Lors de celle-ci, les paires d'élimination sont tout d'abord supprimées de telle sorte qu'on ait, pour tout $x \in X \setminus \{e,t\}$, $w_f(x) \leqslant 0$. (L'égalité $f(t,X) = 0$ permet alors d'interpréter la quantité $w_f(x)$ comme l'excès des flux d'entrée en x en provenance de e) Ensuite, pour faciliter les calculs, les poids fictifs -M et M, où M désigne une constante arbitrairement grande, sont respectivement affectés aux sommets e et t. Ils assurent les inégalités $w_f(e) < 0$ et $w_f(t) > 0$ et, par suite, aucune exécution de la procédure ELIMINATION ne peut réduire les flux des arcs de $\omega^+(e)$ et $\omega^-(t)$. Enfin, un ensemble auxiliaire X' est créé et initialisé au vide.

Le but des itérations est donc de faire évoluer la fonction f ainsi modifiée et l'ensemble X' de telle sorte qu'ils vérifient, après exécution, les relations précédentes. Pour ce faire, chacune

se compose d'une suite de phases de déplacement de flux suivie d'une phase d'élimination. Analysons tout d'abord les premières.

les phases de déplacement de flux

Avant toute exécution de cette suite de phases, l'ensemble $\bar{X}'^{-}$ des sommets de $\bar{X}'$ de poids strictement négatif est non vide et l'on a, pour tout $x \in \bar{X}' \setminus \{t\}$, $w_f(x) \leqslant 0$. Ceci est assuré à la première itération par l'exécution de la phase d'initialisation et, pour toute itération ultérieure, par l'itération précédente.

Cela étant, lors de toute phase r, le complémentaire de l'ensemble X' courant est partitionné en deux sous-ensembles X_r et Y_r dont le dernier cité contient les sommets contre lesquels l'excès de flux d'entrée disponible dans $X_r'^{-}$ (défini de même que $\bar{X}'^{-}$) est à déplacer. De plus, cette partition vérifie les relations

$$X_r^+ = \{x \in X_r : w_f(x) > 0\} = \phi,$$

$$f(X', Y_r) = c(X', Y_r)$$

$$f(X_r, Y_r) \leqslant c(X_r, Y_r)$$

et

$$f(Y_r, X' \quad X_r) = 0.$$

Elle donne, en outre, lieu à une des deux situations suivantes. Dans la première, on a $X_r^- = \phi$ ou $f(X_r, Y_r) = c(X_r, Y_r)$. Dans ces conditions, aucun déplacement de flux n'est possible, l'ensemble X_r est ajouté à X' et la phase d'élimination, décrite ci-dessous, est exécutée. Dans la seconde, aucune des deux égalités précédentes n'est vérifiée. Les arcs de $X_r \times Y_r$ sont alors saturés augmentant et diminuant respectivement les poids de X_r et Y_r. Des paires d'élimination peuvent maintenant exister entre X_r^+ et X_r^-. Leur suppression réduit les poids des sommets de X_r^+ et l'excès de flux d'entrée disponible dans X_r^-. La quantité correspondante de flux est ensuite déplacée contre Y_r et l'ensemble L_r des sommets de X_r extrémités terminales de chemins composés d'arcs donnant à f une valeur strictement positive et d'origine appartenant à X_r^+ est retiré de X_r pour être ajouté à Y_r. Une nouvelle phase, numérotée r+1, commence.

Il convient ici de clarifier le concept de déplacement de flux.

le concept de déplacement de flux

La suppression des paires d'élimination représente une réduction d'une certaine quantité de flux dans $X_r^+ \times X_r^-$. Ceci n'est possible que parce qu'un poids positif supérieur ou égal fut créé pendant le processus de saturation (représentant une augmentation de flux dans $X_r \times Y_r$). Dès lors, une certaine quantité de flux contre X_r^- a été réduite et, au moins, une quantité correspondante de flux contre Y_r a été accrue. Nous interprétons ce changement comme un déplacement de flux puisque le flux contre X_r^- a été déplacé contre Y_r. Dans la suite, nous utiliserons ce terme "déplacement" pour désigner un moyen de réduction de flux entre les paires d'élimination, celle-ci étant compensée par une augmentation correspondante de flux dans une autre direction.

Décrivons à présent la phase d'élimination.

la phase d'élimination

Arrivé en ce point, un nombre fini h de phases de déplacement de flux ont été effectuées et on a

$$Y_{h+1} = \{t\} \cup L_1 \cup \ldots \cup L_h$$

chaque ensemble L_i $(1 \leqslant i < h)$ pouvant contenir des sommets de poids positif ou négatif créé durant l'exécution de celles-ci.

Dans la phase d'élimination, les arcs saturés durant les phases de déplacement de flux sont sélectionnés dans l'ordre inverse de l'ordre de saturation et leur flux est réduit. Ce processus se répète jusqu'à ce que tous les poids positifs soient éliminés.

On peut prouver les inclusions

$$L_i \times L_{i-1} \cap U \subset X_i \times Y_i \cap U, \quad \forall\, i \in \{1, \ldots, h\}$$

et démontrer que l'élimination des poids positifs dans L_i^+ $(i = 1, \ldots, h)$

nécessite une réduction de flux, initialement augmenté dans $L_i \times L_{i-1} \cap U$. Toutefois, avant que les flux dans $(L_i \times L_{i-1}) \cap U$ $(i = 1, \ldots, h)$ ne soient réduits, il convient de supprimer toutes les paires d'élimination existant dans L_i. Notons qu'une telle suppression entraîne des déplacements successifs de flux à travers les ensembles L_i, L_{i-1}, ..., $\{t\}$.

Enfin, la phase d'élimination se termine par la réinitialisation des flux dans $\bar{X}'$ de telle sorte à avoir $w_f(x) \leqslant 0$, pour tout $x \in \bar{X}' \setminus \{t\}$.

L'algorithme suivant formalise ce qui précède.

ALGORITHME DE REDISTRIBUTION DE FLUX III.4.3. Soit f une fonction d'un réseau de transport telle que $0 \leqslant f(u) \leqslant c(u)$, pour tout $u \in U$.

```
Pour tout x ∈ X\{e} effectuer
  w(x) := f(x,X) - f(X,x);
w(e) := -M;
ELIMINATION(X,Z);
w(t) := M;
X' := φ;
Tant que X̄' ≠ φ effectuer
  X_0 := X̄' ;
  Y_0 := φ ;
  L_0 := {t};
  Pour tout u ∈ U effectuer δf(u) := 0;
  X_1 := X̄'\{t};
  Y_1 := {t};
  r := 1;
  Tant que X_r⁻ ≠ φ et f(X_r,Y_r) < c(X_r,Y_r) répéter
    Pour tout (x,y) ∈ (X_r x Y_r) ∩ U effectuer
      δf(x,y) := c(x,y) - f(x,y);
      si δf(x,y) > 0 alors
        f(x,y) := c(x,y);
        w(x) := w(x) + δf(x,y);
        w(y) := w(y) - δf(x,y);
    ELIMINATION(X_r,L_r);
    r := r+1;
    X_r := X_{r-1} \ L_{r-1};
    Y_r := Y_{r-1} ∪ L_{r-1};
```

```
Si X_r^- ≠ φ alors X' := X' ∪ X_r ;
r := r - 1;
Tant que r ≠ 0 effectuer
  ELIMINATION(L_r, Z);
  Tant que L_r^+ ≠ φ effectuer
    choisir un arc (x,y) ∈ (L_r^+ x L_{r-1}) ∩ U tel que
      θ = min{δℓ(x,y), w(x)} > 0;
    ℓ(x,y) := ℓ(x,y) - θ;
    w(x) := w(x) - θ;
    w(y) := w(y) + θ;
    δℓ(x,y) := δℓ(x,y) - θ;
  r := r - 1;
w(e) := c(X',X̄') + |w(X'^-)|;
w(t) := - c(X',X̄');
ELIMINATION(X',Z).
```

Sa complexité temporelle théorique est $\mathcal{O}(n^5)$ au plus.

On notera que cette méthode ne requiert pas que la fonction f que l'on tente de transformer en un flot de valeur maximum soit un flot. Ceci constitue un avantage par rapport aux méthodes précédentes, par exemple, lorsque, disposant d'un flot de valeur maximum d'un réseau, on recherche un même flot d'un réseau "voisin" obtenu par une légère modification du premier.

III.5. La méthode des préflots

La méthode des préflots repose sur les notions de préflots et de réseau de couches. Définissons les tout d'abord.

1) Une fonction $f : U \to \mathbb{R}^+$ est un <u>préflot</u> si elle vérifie les trois propriétés suivantes.

(i) Pour tout arc $u \in U$, on a $0 \leqslant f(u) \leqslant c(u)$.

(ii) Pour tout $x \in X \setminus \{e,t\}$, on a $f(\omega^-(x)) \geqslant f(\omega^+(x))$.

(iii) Tout chemin d'extrémités initiale e et terminale t contient au moins un arc saturé c'est-à-dire donnant aux fonctions f et c les mêmes valeurs.

2) Le <u>réseau de couches</u> associé au réseau de transport $R = (X,U,c)$ et au flot f de ce réseau est le graphe dont l'ensemble des sommets est composé des couches L_i définies par les relations

$$L_0 = \{e\},$$

$$L_i = \{x \in X : x \notin U_{j=1}^{i-1} L_j \text{ et } \exists\, y \in U_{j=1}^{i-1} L_j :$$
$$((y,x) \in U \text{ et } f(y,x) < c(y,x)) \text{ ou } ((x,y) \in U \text{ et } f(x,y) > 0)\}$$

$$L_k = \{t\},$$

où k désigne la longeur minimum d'une chaîne f-augmentante, et dont l'ensemble des arcs est constitué des arcs $(x,y) \in U$ tels que $f(x,y) < c(x,y)$ pour lesquels il existe $i \in \{0, \ldots, k-1\}$ tel que $x \in L_i$ et $y \in L_{i+1}$ et des arcs $(y,x) \in U$ tels que $f(y,x) > 0$ et pour lesquels il existe $i \in \{0, \ldots, k-1\}$ tel que $x \in L_i$ et $y \in L_{i+1}$. Bien sûr, quand il existe, le graphe $(X,U_{R,f})$ du réseau de référence $(X,U_{R,f},c)$ s'identifie au réseau de couches associé au flot f et au réseau de transport $R = (X,U,c)$.

Décrivons à présent brièvement la méthode des préflots. Elle consiste en une suite d'itérations dont chacune a pour objectif de transformer un flot donné en un flot de valeur supérieure. Dans ce but, chacune d'entre elles tente tout d'abord de construire le réseau de couches associé au flot courant. En cas d'échec, ce flot est de valeur maximum. Sinon, un flot de valeur supérieure peut être défini. Pour ce faire, les arcs incidents vers l'extérieur à l'entrée e sont saturés puis une succession de phases d'augmentation et d'équilibrage modifie progressivement le flot courant, établissant ainsi une suite de préflots dont le dernier élément est le flot cherché.

L'algorithme suivant en découle

ALGORITHME DES PREFLOTS III.5.1.

```
| f := flot nul;
| Pour tout x ∈ X effectuer
|   |(in - out)(x) := 0;
| Répéter
|   |CONSTRUCTION_RESEAU_COUCHES(f, echec);
|   |Si echec = faux alors
|   |  |Pour tout x ∈ X effectuer
|   |  |  |α_i(x) := r_i(x);
|   |  |  |
|   |  |  |α_e(x) := ω_e(x);
|   |  |  |
|   |  |  |β(x) := ϕ;
|   |  |Pour tout x ∈ L_1 effectuer
|   |  |  |(in - out)(x) := (in - out)(x) - f(e,x) + c(e,x);
|   |  |  |ajout(β(x),(i,e,c(e,x) - f(e,x));
|   |  |  |(in - out)(e) := (in - out)(e) - c(e,x) + f(e,x);
|   |  |  |f(e,x) := c(e,x);
|   |  |s := 1
|   |  |Répéter
|   |  |  |f := PHASE_AUGMENTATION(f, s);
|   |  |  |f := PHASE_EQUILIBRAGE(f, s);
|   |  |tant que s ≠ 0;
| tant que echec = faux.
```

Les procédures CONSTRUCTION_RESEAU_COUCHES, PHASE_AUGMENTATION, PHASE_EQUILIBRAGE ainsi que les structures de données qui apparaissent sont trop compliquées pour être expliquées ici. Nous renvoyons le lecteur désireux d'en connaître davantage à l'annexe [13].

Cet algorithme converge et détermine, après exécution, un flot de valeur maximum. Sa complexité temporelle théorique est de $\mathcal{O}(n^3)$ au plus.

III.6. La programmation linéaire

La théorie de la programmation linéaire résout les problèmes du flot maximum, du flot b-canalisé maximum, du flot maximum de coût minimum et de transfert de Hitchcock.

De fait, le premier (resp. le second) consiste en la recherche de m nombres (réels ou entiers) $f(u_1), \ldots, f(u_m)$ maximisant la fonction

$$w = \sum_{u \in \omega^-(t)} f(u)$$

et soumis aux contraintes

$$\sum_{u \in \omega^-(x)} f(u) = \sum_{u \in \omega^+(x)} f(u), \quad \forall x \in X \setminus \{e,t\},$$

$$0 \leqslant f(u) \leqslant c(u) \text{ (resp. } b(u) \leqslant f(u) \leqslant c(u)), \quad \forall u \in U$$

c'est-à-dire à la résolution du problème de programmation linéaire

$$\min \; - \sum_{u \in \omega^-(t)} f(u)$$

sous les contraintes

$$\sum_{u \in \omega^-(x)} f(u) = \sum_{u \in \omega^+(x)} f(u), \quad \forall x \in X \setminus \{e,t\},$$

$$0 \leqslant f(u) \leqslant c(u) \text{ (resp. } b(u) \leqslant f(u) \leqslant c(u)), \quad \forall u \in U$$

en les inconnues (réelles ou entières) $f(u_1), \ldots, f(u_m)$.

Par ailleurs, le problème du flot maximum de coût minimum revient à la recherche parmi les flots maximum d'un flot minimisant la fonction

$$\sum_{u \in U} d(u) f(u) .$$

Un tel flot peut être obtenu par résolution du problème de programmation linéaire en les inconnues $f(u_1), \ldots, f(u_m)$

$$\min \; \{-M \sum_{u \in \omega^-(t)} f(u) + \sum_{u \in U} d(u) f(u)\}$$

sous les contraintes

$$\sum_{u \in \omega^-(x)} f(u) = \sum_{u \in \omega^+(x)} f(u), \quad \forall x \in X \setminus \{e,t\},$$

$$0 \leqslant f(u) \leqslant c(u), \quad \forall u \in U$$

où M désigne une constante positive arbitrairement grande.

Enfin, le problème de transfert d'Hitchcock, cas particulier du problème précédent peut bien sûr être ramené à un problème de programmation linéaire.

La résolution du problème du flot maximum en variables réelles semble exiger l'application de la méthode des deux phases ou de la méthode des pénalités. Toutefois, un examen plus attentif montre qu'on peut déduire des contraintes du PLS associé une base réalisable et, par suite, appliquer directement l'algorithme primal du simplexe. En effet, en introduisant dans les m équations $f(u_i) \leqslant c(u_i)$, $(i = 1, \ldots, m)$ des variables d'écarts $e_1, \ldots, e_m$ et en associant aux n-2 équations de conservation de la somme des flux aux sommets distincts de e et t les variables artificielles $a_2, \ldots, a_{n-1}$ de coût arbitrairement grand, le nouveau PLS ainsi obtenu admet la base réalisable $(e_1, \ldots, e_n, a_2, \ldots, a_{n-1})$. Sa solution fournit, bien sûr, la solution du problème du flot maximum. On réalise de la sorte un gain appréciable de temps ainsi qu'une économie non négligeable de m colonnes d'un tableau du simplexe.

La complexité temporelle théorique de l'algorithme obtenu reste néanmoins de $\mathcal{O}(A_{2n+m}^{n+m}\ (n+m)^2)$ au plus.

Par ailleurs, la résolution de ce même problème mais en variables entières relève de techniques plus sophistiquées que nous exposons en [13].

III.7. Tableau récapitulatif

Le tableau suivant résume les résultats rappelés dans ce chapitre. Dans ses cinq premières colonnes, il énonce les contraintes imposées par les sept algorithmes et indique, pour chacun, la possibilité qu'il offre de s'adapter à la résolution des trois problèmes annexes : problème du flot b-canalisé maximum, du flot maximum de coût minimum et de transfert de Hitchcock. En outre, ses trois

dernières colonnes déterminent, pour chacun, la place mémoire qu'il requiert ainsi que, quand elle a été démontrée, sa complexité temporelle théorique ou, sinon, une borne sur le temps d'exécution.

La signification des symboles utilisés est la suivante :

+ renseigne une adaptation aisée,

- renseigne une inadaptation,

± renseigne une adaptation possible mais non immédiate,

$|f^*|$ représente la valeur maximum d'un flot,

N désigne un entier positif tel que, pour toute partition des sommets de X en deux ensembles S et $\bar{S}$, le nombre d'arcs ayant une extrémité dans S et l'autre dans $\bar{S}$ est plus petit ou égal à N.

De plus, les abréviations bM, Mcm, Hi, alg, ens, vect, mat, dim remplacent respectivement les termes problème du flot b-canalisé maximum, problème du flot maximum de coût minimum, problème de transfert de Hitchcock, algorithme, ensemble, vecteur, matrice et dimension.

algorithme ou méthode	contraintes sur le réseau de transport	résolution en nombre	bM	Mcm	Hi	complexité théorique temporelle	borne temporelle	place mémoire
alg. de Ford et Fulkerson recherche en profondeur d'abord	capacités entières	entier	+	±	±	/	au plus $\|f^*\|$ augmentations de flot	3 vect. de dim. n 2 mat. de dim. n x n
alg. de Ford et Fulkerson recherche en largeur d'abord	/	entier ou réel	+	±	±	$\mathcal{O}(n^6)$ au plus	au plus $n(n-1)(n+2)/4$ augmentations de flot	3 vect. de dim. n 2 mat. de dim n x n
alg. d'Edmonds et Karp	capacités entières	entier	+	±	±	/	au plus $1+\log_{N/N-1}\|f^*\|$ augmentations de flot	3 vect. de dim. n 2 mat. de dim n x n
alg. de Dinic	capacités entières	entier	+	±	±	$\mathcal{O}(n^4)$ au plus	/	2 vect. de dim. n 2 mat. de dim. n x n n ens. = 1 vect. de dim. 2m
méthode de redistribution de flux	capacités entières	entier	-	-	-	$\mathcal{O}(n^5)$ au plus	/	3 mat. de dim. n x n 2 vect. de dim. m 2 ens. = 1 vect. de dim. n n ens. = 1 vect. de dim. n
méthode des préflots	/	entier ou réel	-	-	-	$\mathcal{O}(n^3)$ au plus	/	2 vect. de dim. n 2 mat. de dim. n x n 6n ens. = 2 vect. de dim. 2m n ens. = 1 vect. de dim. n n ens. de dim. non bornées
programmation linéaire	/	entier ou réel	+	+	+	$(A^m_{2n}\ (n+m)^2)$ au plus pour résolution réelle	/	1 mat. de dim. $(m+n+2) \times (2m+n)$ pour résolution réelle

remarque : La taille de l'espace mémoire nécessaire à l'exécution des différents algorithmes a été déterminée comme suit :

1) la méthode de recherche en profondeur d'abord nécessite l'emploi
   - de 3 vecteurs de dimension n pour mémoriser les tableaux *min*, *marqué* et *pred*,
   - de 2 matrices de dimension n x n pour mémoriser le flot et les capacités du réseau de transport;

2) la méthode de recherche en largeur d'abord nécessite l'emploi
   - de 3 vecteurs de dimension n pour mémoriser les tableaux *num*, *min* et *pred*,
   - de 2 matrices de dimension n x n pour mémoriser le flot et les capacités du réseau de transport;

3) l'algorithme d'Edmonds et Karp nécessite l'emploi
   - de 3 vecteurs de dimension n pour mémoriser les tableaux *min*, *num* et les ensembles *ens_sommets_marqués*, *ens_sommets_non_marqués*,
   - de 2 matrices de dimension n x n pour mémoriser le flot et les capacités du réseau de transport;

4) l'algorithme de Dinic nécessite l'emploi
   - de 2 vecteurs de dimension n pour mémoriser les tableaux *num* et *suiv*,
   - de n ensembles *succ* dont la longueur totale représente un vecteur de 2m éléments,
   - de 2 matrices de dimension n x n pour mémoriser le flot et les capacités du réseau de transport;

5) la méthode de redistribution de flux nécessite l'emploi
   - de 3 matrices de dimension n x n pour mémoriser le flot, les capacités du réseau de transport et la matrice δf,
   - de 2 vecteurs de dimension n pour mémoriser les tableaux *w* et *chemin*,
   - de deux ensembles X_r et Y_r dont la longueur totale représente un vecteur à n éléments,
   - d'au plus n ensembles L_o, ... ,L_n dont la longueur totale représente un vecteur à n éléments;

6) la méthode des préflots nécessite l'emploi

- de 2 vecteurs de dimension n pour mémoriser les tableaux *num* et *(in - out)*,
- de 2 matrices de dimension n x n pour mémoriser le flot et les capacités du réseau de transport;
- d'au plus n ensembles L_i dont la longueur totale représente au plus un vecteur de n éléments,
- de n ensembles β de taille non bornée,
- de 4n ensembles ω_e, ω_i, r_i, r_e dont la longueur totale représente un vecteur de 2m éléments,
- de 2n ensembles α_e, α_i dont la longueur totale représente au plus un vecteur de 2m éléments;

7) l'algorithme expliqué au paragraphe III.6. nécessite l'emploi d'une matrice de dimension (m+n+2) x (2m+n) pour mémoriser le tableau du simplexe.

III.8. Mesures pratiques

Le tableau III.8.1. ci-dessous, extrait de [5] , compare les méthodes de recherche en profondeur d'abord (1), de recherche en largeur d'abord (2), d'Edmonds et Karp (3), de Dinic (4), des préflots (5) ainsi que de redistribution de flux (6). Il fournit, en fonction du nombre de sommets et du nombre d'arcs, la moyenne des temps d'exécution (en seconde et sur IBM 360/65) de chacun de ces algorithmes appliqués à dix réseaux de transport générés aléatoirement et dont la fonction de capacité est soumise, par groupe de deux, aux cinq bornes supérieures : 50, 100, 300, 500 et 1000. Les * indiquent que les cas correspondant n'ont pas été traités, les résultats précédents ayant clairement établi l'infériorité de l'algorithme considéré.

num.	|X|	|U|	(1)	(2)	(3)	(4)	(5)	(6)
1	25	60	0,07	0,08	0,14	0,07	0,10	0,13
2	25	120	0,17	0,15	0,43	0,17	0,31	0,34
3	25	180	0,24	0,28	0,90	0,26	0,55	0,48
4	25	240	0,43	0,47	1,64	0,33	0,63	0,66
5	25	300	0,48	0,49	1,84	0,41	0,75	0,78
6	25	360	0,61	0,65	2,76	0,48	0,84	0,83
7	25	420	0,85	0,85	3,82	0,60	0,96	1,17
8	25	480	0,94	0,99	4,73	0,66	1,06	1,32
9	25	540	1,04	1,08	5,13	0,70	1,06	1,43
10	50	246	0,46	0,47	1,51	0,45	0,88	0,83
11	50	490	1,03	1,07	5,48	0,98	1,48	1,38
12	50	736	1,86	1,53	8,91	1,39	1,79	1,78
13	50	980	2,66	2,40	15,66	1,87	2,72	2,53
14	50	1226	4,19	3,52	26,60	2,38	3,60	3,66
15	50	1470	4,19	3,87	25,39	2,63	4,13	4,74
16	50	1714	5,38	4,64	75,71	3,11	4,54	5,56
17	50	1960	5,35	4,64	*	3,34	4,38	6,26
18	50	2204	6,52	6,20	*	3,63	5,11	7,99
19	75	556	1,59	1,41	*	1,43	2,35	2,08
20	75	1110	3,24	2,68	*	2,69	3,77	3,58
21	75	1666	5,50	4,52	*	4,09	7,20	6,08
22	75	2220	7,83	6,64	*	5,49	7,62	7,91
23	75	2776	11,06	8,52	*	6,54	9,05	10,31
24	75	3330	12,12	11,52	*	8,42	10,70	12,76
25	75	3884	15,19	13,87	*	10,45	12,57	16,30
26	75	4440	18,20	17,78	*	11,66	14,45	17,74
27	75	4994	20,59	18,82	*	12,93	14,66	22,09
28	100	594	1,98	1,91	*	1,82	3,00	*
29	100	692	2,43	2,20	*	2,13	3,29	*
30	100	792	2,90	2,71	*	2,44	3,66	*
31	100	890	3,55	3,14	*	2,97	4,72	*
32	100	990	3,63	3,29	*	3,04	5,35	*
33	100	1980	8,29	7,04	*	6,35	9,23	*
34	100	2970	12,15	11,51	*	9,78	12,68	*
35	100	3960	17,02	16,21	*	12,57	14,65	*
36	100	4950	21,78	20,93	*	15,42	18,13	*
37	200	1194	7,21	6,82	*	6,32	10,93	*
38	200	1592	10,35	8,94	*	8,54	14,93	*
39	200	1900	12,76	11,21	*	10,38	15,78	*
40	200	2388	15,33	13,47	*	12,49	17,45	*
41	200	2786	17,04	15,82	*	14,47	20,07	*
42	200	3184	20,45	19,23	*	17,17	22,78	*
43	200	3582	24,89	21,86	*	18,96	27,23	*
44	200	3980	26,76	24,15	*	20,84	28,93	*
45	200	7960	56,12	51,62	*	43,35	53,68	*
46	300	1794	16,46	15,93	*	14,82	*	*
47	300	2692	25,53	24,49	*	21,48	*	*
48	300	3588	32,13	28,19	*	29,84	*	*
49	300	4486	35,65	35,72	*	37,84	*	*
50	300	6278	56,96	52,60	*	52,92	*	*
51	500	2496	34,29	32,51	*	31,37	*	*
52	500	3742	49,22	49,37	*	46,45	*	*
53	500	4990	68,18	67,47	*	62,91	*	*
54	1000	2998	76,90	77,17	*	74,12	*	*
55	1000	3996	103,79	99,58	*	93,94	*	*
56	1000	4996	135,71	132,08	*	124,60	*	*
57	1500	4998	175,37	184,06	*	173,94	*	*
58	1500	5622	218,04	214,95	*	203,63	*	*

TABLEAU III.8.1.

Temps moyen d'exécution

On peut tirer les conclusions suivantes :

1) L'algorithme de Dinic s'avère le plus efficace. De fait, seules la méthode de recherche en profondeur d'abord dans les cas 3 et 49 et la méthode de recherche en largeur d'abord dans les cas 2, 19, 20, 48, 49 et 50 se montrent plus rapides. Toutefois, elles ne s'y révèlent supérieures que d'une manière insignifiante.

Il semble donc judicieux d'éviter certains étiquetages en recherchant plusieurs chaînes augmentantes dans le même réseau.

2) Pour les problèmes de taille faible, les performances des méthodes de recherche en profondeur d'abord et en largeur d'abord sont comparables à celles de l'algorithme de Dinic. Ce fait, couplé à leur facilité de programmation mais aussi au faible espace mémoire qu'elles requièrent en fait de "bons" algorithmes.

3) L'algorithme d'Edmonds et Karp est tout à fait inefficace : les 16 cas traités montrent clairement la croissance rapide de son temps d'exécution. Cette inefficacité est certainement due au processus d'étiquetage : le marquage d'un nouveau sommet exige, en effet, l'examen de tous les arcs joignant un sommet marqué à un sommet non marqué.

Cependant, comme le montre la table III.8.2. ci-après, il requiert, parmi les quatre algorithmes dérivés de l'algorithme de Ford et Fulkerson, le moins d'augmentations de flot.

4) La méthode de redistribution de flux est tout aussi médiocre. Ce fait et sa complexité de conception et de codage nous conduisent à la rejeter.

5) Bien que de complexité temporelle théorique inférieure, la méthode des préflots impose des temps d'exécution supérieurs aux temps d'exécution de la méthode de Dinic.

num.	|X|	|U|	(1)	(2)	(3)	(4)
1	25	60	1.07	1.00	1.00	1.00
2	25	120	1.57	1.39	1.00	1.39
3	25	180	1.74	1.54	1.00	1.54
4	25	240	2.12	1.62	1.00	1.62
5	25	300	2.06	1.60	1.00	1.60
6	25	360	3.24	1.78	1.00	1.78
7	25	420	2.47	1.83	1.00	1.83
8	25	480	2.27	1.77	1.00	1.77
9	25	540	2.07	1.67	1.00	1.67
10	50	246	1.50	1.67	1.00	1.67
11	50	490	1.75	1.88	1.00	1.88
12	50	736	2.40	1.65	1.00	1.65
13	50	980	3.64	2.06	1.00	2.06
14	50	1226	4.04	2.04	1.00	2.04
15	50	1470	3.07	1.98	1.00	1.98
16	50	1714	2.60	1.62	1.00	1.62

TABLEAU III.8.2.

Proportion du nombre moyen d'augmentations de flot relativement à l'algorithme (3)

III.9. Le meilleur algorithme

Au vu des résultats précédents, l'algorithme de Dinic est, à nos yeux, le meilleur algorithme et ce, pour les raisons suivantes :

1) il possède le meilleur temps moyen d'exécution,

2) il requiert un espace mémoire des plus faibles,

3) il possède une complexité temporelle théorique des meilleures,

4) il s'adapte aisément à la résolution d'autres problèmes tel que le problème du flot b-canalisé maximum,

5) il est de conception et de codage relativement simples.

BIBLIOGRAPHIE

[1] A.V. AHO, J.E. HOPCROFT, D. ULLMAN, The design and analysis of computer algorithms, Addison-Wesley, Reading, Massachussets (1974).

[2] C. BAUDOIN, B. MEYER, Méthodes de programmation, Eyrolles, Paris (1980).

[3] A. BORONDIN, Computational Complexity and the Existence of Complexity Gaps, Journal of the ACM, vol 19, n° 1 (1972), 158-174.

[4] J. CHAITIN, On the length of programs for computing finite binary sequences : statistical considerations, Journal of the ACM, vol. 16, n° 1 (1969), 145-159.

[5] T.Y. CHEUNG, Computational Comparison of Eight Methods for the Maximum Network Flow Problem, ACM Transactions on Mathematical Software, vol. 6, n° 1 (1980), 1-16.

[6] J. EDMONDS, Paths, trees and flowers, Canad. J. Math. 17 (1965), 449-467.

[7] J. FICHEFET, Cours d'aide à la décision multicritère, Cours de 2^{e} licence, Ed. Ronéotypée, Namur (1984).

[8] J. FICHEFET, Cours de recherche opérationnelle, Cours de 1^{ere} licence, Ed. Ronéotypée, Namur (1983).

[9] J. FICHEFET, Structures de données et complexité des algorithmes : Regards sur le problème de l'arbre maximal de poids extrémal, Cahier du C.E.R.O., vol. 24, n° 2-3-4 (1982), 193-220.

[10] M. GAREY, D.S. JOHNSON, Computers and intractability, a guide to the theory of NP-completeness, Freeman, San Francisco (1978).

[11] G.T. GUILBAUD, Les théories de l'intérêt général et le problème logique de l'agrégation, Economie appliquée, tome V, n° 4 (1952), 501-502.

[12] E. HOROWITZ, S. SAHNI, Computing partitions with applications to the knapsack problem, Journal of the ACM 21 (1974), 277-292.

[13] J.M. JACQUET, Conception d'algorithmes de flot dans les réseaux de transport, Ed. Ronéotypée, Namur (1984).

[14] L.I. KRONSJE, Algorithms : their complexity and efficiency, John Wiley & Sons, New-York (1979), 1-9.

[15] H. LEROY, La théorie de la calculabilité, Cours de 2 eme licence, Ed. Ronéotypée, Namur (1984).

[16] H. LEROY, Méthodologie de la programmation, Cours de 1 e licence, Ed. Ronéotypée, Namur (1983).

[17] R. LESSUISSE, Analyse des erreurs contenues dans 19 algorithmes de recherche dichotomique, Thèse de doctorat, Namur (1980).

[18] J.E. SAVAGE, The complexity of Computing, John Wiley & Sons, New-York (1976).

[19] M. THATCHER, Complexity of computer computations, Plenum Breas, New-York (1972).

[20] J.F. TRAUB, Algorithms and Complexity, new directions and recent results, New-York (1976), 281-299.

[21] A.M. TURING, On computable numbers with an application to the Entscheidungsproblem, Proc. Lond. Math. Soc., vol. 42 (1936), 230-265.

TABLE DES MATIERES

ANNEXE : CONCEPTION D'ALGORITHMES DE FLOT DANS LES RESEAUX DE TRANSPORT

Annexe au mémoire

ANALYSE DES PERFORMANCES D'ALGORITHMES ET APPLICATION A LA COMPARAISON D'ALGORITHMES DE FLOT DANS LES RESEAUX DE TRANSPORT

présentée par

J.M. JACQUET

Année académique : 1983-1984

INTRODUCTION

Dans cet ouvrage, nous présentons les différentes recherches effectuées à propos d'algorithmes solutionnant certains problèmes de flot dans les réseaux de transport et tout spécialement le problème du flot maximum. Ce dernier nous a paru particulièrement intéressant en raison de ses nombreuses applications. Citons pour preuve le problème de distribution, le problème du trafic maritime ainsi que le problème des représentants.

Au chapitre I, nous précisons les divers concepts utilisés dans la suite : notions de flot, de réseau de transport, de fonction admissible, de préflot, de pseudo-flot, ... Nous définissons aussi le problème du flot maximum ainsi que trois problèmes annexes qui nous permettent de tester la modifiabilité des algorithmes qui le résolvent : les problèmes du flot b-canalisé maximum, du flot maximum de coût minimum et de transfert de Hitchcock. Nous donnons, en outre, pour chacun d'eux des exemples d'application. Enfin, nous rappelons les notions de complexités d'algorithme et démontrons le théorème fondamental du flot maximum et de la coupe minimum. Ce dernier résultat nous a suggéré le théorème du flot b-canalisé maximum très utile à la résolution du problème du flot b-canalisé maximum.

Le second chapitre a pour objectif l'étude des méthodes d'étiquetage. Nous décrivons tout d'abord le graphe d'écart en montrant son intérêt au travers de l'algorithme AUGMENTATION. Ensuite, nous généralisons l'algorithme de Ford et Fulkerson (généralement exposé dans le cadre de 1-graphes antisymétriques) au cas de 1-graphes. Nous montrons, en outre, en nous appuyant sur le théorème du flot b-canalisé maximum, les modifications à lui apporter pour qu'il résolve le problème du flot b-canalisé maximum. Enfin, nous particularisons cet

algorithme à quatre types de recherche de chaînes augmentantes en établissant, pour chaque méthode qui en découle, sa complexité théorique temporelle ou une borne sur le nombre d'augmentations de flot.

La première résulte d'une exploration en profondeur d'abord des sommets du graphe d'écart. Nous prouvons, qu'appliquée à des réseaux de transport à capacités entières, elle converge après un nombre d'augmentations de flot égal au plus à la valeur d'un flot maximum du réseau traité.

La deuxième se base sur la recherche en largeur d'abord d'une chaîne augmentante. Nous établissons qu'elle converge après au plus

$$\frac{n\,(n-1)\,(n+2)}{4}$$

augmentations de flots, où n désigne le nombre de sommets du réseau considéré, et prouvons que sa complexité temporelle théorique est de $\mathcal{O}(n^6)$ au plus.

Dans une troisième méthode, nous recherchons, à la suite de J. Edmonds et R.M. Karp, une chaîne augmentante fournissant l'augmentation la plus grande possible de flot. Nous démontrons qu'elle converge après au plus

$$1 + \log_{M/M-1} |f^*|$$

augmentations de flot, où M désigne, pour tout sous-ensemble de sommets S, le nombre maximum d'arcs ayant une extrémité dans S et une autre dans son complémentaire $\bar{S}$ et $|f^*|$ la valeur d'un flot maximum, et comparons, en outre, ces deux dernières bornes.

Enfin, la quatrième se propose de rechercher plusieurs chaînes augmentantes dans un même réseau appelé réseau de référence. Nous établissons qu'elle possède une complexité temporelle théorique de $\mathcal{O}(n^4)$ au plus et démontrons que, dans le cas plus favorable de réseaux de transport à capacités unitaires, cette complexité est de $\mathcal{O}(n^{8/3})$ au plus.

Nous modifions aussi la méthode d'Edmonds et Karp de façon à résoudre les problèmes du flot maximum de coût minimum et de transfert de Hitchcock. Une première démarche permet de construire un algorithme les solutionnant dont nous établissons qu'il converge après au plus

$\frac{1}{4}(n-1)^2 n(n+2) D + 1$

augmentations de flot, où D est la valeur maximum des coûts associés aux arcs du réseau considéré. Toutefois, cette borne n'étant guère performante, nous développons une variante, la méthode de graduation, pour laquelle cette borne dépend de façon logarithmique plutôt que linéaire des capacités. Nous l'exposons tout d'abord dans le cadre plus simple de la résolution du problème de transfert de Hitchcock et montrons ensuite comment l'adapter à la résolution du problème du flot maximum de coût minimum.

Au troisième chapitre, nous expliquons la méthode de redistribution de flux de B. Kinariwala et A.G. Rao. Nous définissons, tout d'abord, les notions de fonction de poids et de déplacement de flux sur lesquelles elle repose. Puis, nous construisons l'algorithme qui en découle en donnant une démonstration personnelle plus simple et plus compréhensible. Sa complexité temporelle théorique est de $\mathcal{O}(n^5)$ au plus.

Le chapitre IV décrit la méthode des préflots. Nous introduisons notamment la notion de réseau de couches et expliquons les phases d'augmentation et d'équilibrage qui la caractérisent. Nous prouvons, en outre, que l'algorithme obtenu possède une complexité temporelle théorique de $\mathcal{O}(n^3)$ au plus.

Enfin, le dernier chapitre contient les techniques de la programmation linéaire nécessaires à la résolution des problèmes précédents. Nous rappelons l'algorithme primal du simplexe, le modifions de telle sorte qu'il puisse tenir directement compte de variables bornées supérieurement et donnons encore une forme révisée évitant le calcul inutile de certains éléments du tableau du simplexe. Nous comparons, en outre, cet algorithme révisé avec l'algorithme primal du simplexe classique sur base du nombre de multiplications et de divisions. Nous exposons ensuite les méthodes des deux phases et des pénalités ainsi que l'algorithme dual dont nous fournissons aussi une forme révisée. Après quoi, nous construisons et démontrons un algorithme particulièrement efficace d'inversion de base et traitons quelques problèmes de postoptimisation, d'analyse de sensibilité et de paramétrage. Enfin, nous concluons ce travail en résolvant grâce

à une méthode "branch and bound" le problème de programmation linéaire en nombres entiers.

Les algorithmes sont écrits dans un pseudo-langage proche du Pascal qui nous semble assez suggestif. Néanmoins, le lecteur qui éprouverait des difficultés à le comprendre peut consulter l'excellent ouvrage de C. Baudoin et B. Meyer ([2]). De même, en ce qui concerne les notions élémentaires de la théorie des graphes, nous le renvoyons à [8].

Nous tenons à exprimer notre particulière reconnaissance à Monsieur le Professeur J. Fichefet, qui nous a permis de réaliser ce travail, pour l'aide constante et efficace qu'il nous a apportée. Nos remerciements vont aussi à ceux qui, de près ou de loin, ont collaboré à sa réalisation.

CHAPITRE I

PROBLEMES DE FLOT DANS LES RESEAUX DE TRANSPORT

I.1. Flots et réseaux de transport

Un des meilleurs exemples pour illustrer la notion de flot sur un graphe est certainement celui du courant électrique continu parcourant un réseau de dipôles. C'est d'ailleurs dans ce cadre qu'ont d'abord été étudiés les problèmes de flot. Toutefois, l'intérêt que l'on porte aujourd'hui à cette notion vient de ce qu'elle fournit, couplée à la notion de réseaux de transport, un modèle général applicable à un grand nombre d'autres problèmes concrets: problèmes de transport (routiers, aériens, ferroviaires), structuration et dimensionnement optimaux des réseaux de communication, problèmes de gestion de stocks, d'ordonnancement, d'affectation ainsi que de son intérêt considérable dans les mathématiques combinatoires: théorèmes de Menger, Hall, König-Egervary, Gallai-Milgram, Dilworth, ...

Afin d'être aussi complet que possible, définissons tout d'abord les concepts de flot et de réseau de transport.

DEFINITION I.1.1. Un réseau de transport est un triplet $R = (X,U,c)$ soumis aux deux conditions suivantes:

(i) $(X,U) = (\{x_1, \ldots, x_n\}, \{u_1, \ldots, u_m\})$ est un 1-graphe sans boucle dans lequel le sommet x_1, appelé l'entrée du réseau, est tel que $d^-(x_1) = 0$ et pour lequel le sommet x_n, appelé la sortie du réseau, vérifie l'égalité $d^+(x_n) = 0$.

(ii) $c: U \to \mathbb{R}^+$ est une fonction qui, à tout arc $u \in U$, associe un nombre réel positif $c(u)$, appelé la capacité de l'arc u.

CONVENTION I.1.2. Dans la suite de ce travail,

R ou (X,U,c)

désignent un réseau de transport, dont nous notons systématiquement

X ou $\{x_1, \ldots, x_n\}$; U ou $\{u_1, \ldots, u_m\}$; x_1 ou e; x_n ou t

l'ensemble des sommets, l'ensemble des arcs, l'entrée et la sortie.

DEFINITION I.1.3. Un flot dans R est une fonction $f: U \to \mathbb{R}^+$ telle qu'on ait

$$\forall u \in U : 0 \leqslant f(u) \leqslant c(u) \qquad (1)$$

$$\forall x \in X \setminus \{e,t\} : \sum_{u \in \omega^-(x)} f(u) = \sum_{u \in \omega^+(x)} f(u) \qquad (2)$$

Le nombre f(u) ainsi associé à l'arc u est appelé le flux dans l'arc u. Par abus de langage, nous l'appelons de même, si la fonction $f : U \to \mathbb{R}^+$ viole la proposition (2) tout en vérifiant les inégalités (1).

PROPOSITION I.1.4. Pour tout flot f d'un réseau de transport R, on a

$$\sum_{u \in \omega^+(e)} f(u) = \sum_{u \in \omega^-(t)} f(u)$$

Démonstration. C'est une conséquence immédiate de la condition I.1.3.(2) ///

DEFINITION I.1.5. Le nombre

$$|f| = \sum_{u \in \omega^+(e)} f(u) = \sum_{u \in \omega^-(t)} f(u)$$

est <u>la valeur du flot</u> f

REMARQUE I.1.6. Remarquons dès à présent que, s'agissant de traiter des problèmes de flot, les restrictions imposées au couple G = (X,U) en I.1.1. sont très peu contraignantes. D'une part, on peut toujours supposer qu'on a, pour tout couple de sommets $(x,y) \in X$, $m_G^+(x,y) \leqslant 1$. De fait, si deux sommets $x,y \in X$ sont tels que $m_G^+(x,y) > 1$, il suffit de résoudre le problème de flot proposé sur le graphe obtenu en ajoutant un sommet sur chaque arc d'origine x et d'extrémité terminale y ou encore, si on ne s'intéresse qu'à la valeur des flots, en introduisant un seul arc (x,y) de capacité égale à la somme des capacités des arcs initiaux d'origine x et d'extrémité terminale y. D'autre part, on peut toujours supposer avoir $d^-(x_1) = 0$ et $d^+(x_n) = 0$. Il suffit, en effet, dans le cas contraire, d'introduire, selon les besoins, une nouvelle entrée x_1', une nouvelle sortie x_n' et de nouveaux arcs (x_1', x_1) et (x_n', x_n) de capacité infinie.

En outre, des arcs de capacité négative peuvent toujours être remplacés par des arcs d'orientation et de capacité opposées.

Enfin, le cas d'un graphe simple peut être traité en considérant chaque arête comme deux arcs de sens opposés et de même capacité.

REMARQUE I.1.7. On peut donner une interprétation aux résultats précédents. Si l'on assimile les arcs du réseau R à des voies de communication et les flux f(u) à des quantités d'un bien quelconque circulant sur ces voies, la relation I.1.3.(1) affirme que la quantité nette f(u), nécessairement positive du bien considéré qui circule sur u, n'excède pas la capacité de l'arc u et la relation I.1.3.(2) qu'il n'y a ni création, ni destruction de bien en un sommet quelconque distinct de l'entrée et de la sortie de R. Par ailleurs, |f| représente la quantité de bien qui sort de x_1 vers le réseau et arrive du réseau en x_n.

Remarquons ici que les notions développées ci-dessus permettent de tenir compte d'une perte ou d'un gain de bien en un sommet. Pour ce faire, il suffit d'introduire, pour tout tel sommet x, un sommet auxiliaire x* et, dans le premier cas, deux arcs (x,x^*) et (x^*,x_n) de capacités égales à la perte maximum autorisée, dans le deuxième cas, deux arcs (x_1,x^*), (x^*,x) de capacités égales au gain maximum autorisé puis de considérer les problèmes de flot sur le graphe ainsi obtenu.

De même, on peut prendre en considération une capacité maximum de traitement de bien en un sommet. Il suffit, pour y parvenir, de dédoubler tout tel sommet x en deux sommets x',x", d'introduire un arc (x',x") de capacité égale à la capacité maximum de traitement en x, de remplacer tout arc de la forme (y,x) (resp. (x,y)), $y \in X$, par un arc (y,x') (resp. (x",y)) puis, enfin, de résoudre le problème de flot sur le graphe ainsi construit.

Les notions de préflot et de pseudo-flot seront largement utilisées dans la suite. Afin de les expliciter, adoptons les définitions suivantes.

DEFINITION I.1.8. Une fonction $f: U \to \mathbb{R}^+$ est <u>admissible</u> si elle vérifie les inégalités

$$0 \leqslant f(u) \leqslant c(u) \quad , \quad \forall u \in U$$

DEFINITIONS I.1.9. Soit $f: U \to \mathbb{R}^+$ une fonction admissible.

a) Un arc u est <u>f-saturé</u> s'il vérifie l'égalité $f(u) = c(u)$
b) Un chemin est <u>f-bloqué</u> s'il contient au moins un arc f-saturé
c) Un sommet x est <u>f-bloqué</u> si tout chemin d'extrémité initiale x et d'extrémité terminale t est f-bloqué.

DEFINITION I.1.10. Un flot f est <u>limite</u> si l'entrée e est f-bloquée.

DEFINITION I.1.11. Une fonction $f: U \to \mathbb{R}^+$ est un <u>préflot</u> si elle vérifie les trois propriétés suivantes.

(i) f est admissible

(ii) Pour tout $x \in X \setminus \{e,t\}$, on a

$$\sum_{u \in \omega^-(x)} f(u) \geq \sum_{u \in \omega^+(x)} f(u)$$

(iii) Le sommet e est f-bloqué.

DEFINITION I.1.12. Une fonction $f: U \to \mathbb{R}^+$ est pseudo-flot si elle vérifie les égalités

$$\sum_{u \in \omega^-(x)} f(u) = \sum_{u \in \omega^+(x)} f(u) \ , \ \forall\, x \in X \setminus \{e,t\}$$

La proposition suivante est immédiate.

PROPOSITION I.1.13.

a) Pour tout flot limite f, l'entrée e est f-bloquée
b) Tout pseudo-flot admissible est un flot et réciproquement
c) Tout flot limite f est un préflot et un pseudo-flot pour lequel e est f-bloqué et réciproquement.

Démonstration. C'est immédiat. ///

REMARQUE I.1.14. On peut, bien sûr, donner une interprétation aux notions de pseudo-flot et de préflot. Si l'on assimile à nouveau les arcs du réseau R à des voies de communication et les nombres f(u) à des quantités d'un bien quelconque circulant sur ces voies, la notion de pseudo-flot formalise le fait qu'il n'y a ni création, ni destruction de bien en un sommet quelconque distinct de l'entrée et de la sortie de R. Par contre, la notion de préflot implique que la quantité nette f(u), nécessairement positive, n'excède pas la capacité de l'arc u du bien considéré qui circule sur u et qu'il n'y a création de bien en aucun sommet distinct de l'entrée et de la sortie de R.

I.2. Problèmes du flot maximum, du flot b-canalisé maximum, du flot maximum de coût minimum, de transfert de Hitchcock

Un problème d'importance parmi les problèmes de flot est, sans aucun doute, le problème du flot maximum. Nous le définissons dans ce paragraphe et donnons ensuite quelques exemples d'application.

Tous les algorithmes développés dans les chapitres suivants visent à le résoudre. Certains d'entre-eux peuvent, en outre, servir de base à la résolution d'autres problèmes annexes: le problème du flot b-canalisé maximum, le problème du flot maximum de coût minimum, le problème de transfert de Hitchcock. Dans le but de fixer dès à présent les appellations, nous les définissons aussi, tout en fournissant, pour chacun d'eux quelques exemples d'application.

DEFINITION I.2.1. Le problème du flot maximum consiste à rechercher dans l'ensemble des flots que l'on peut définir sur un réseau R un flot dont la valeur est maximum.

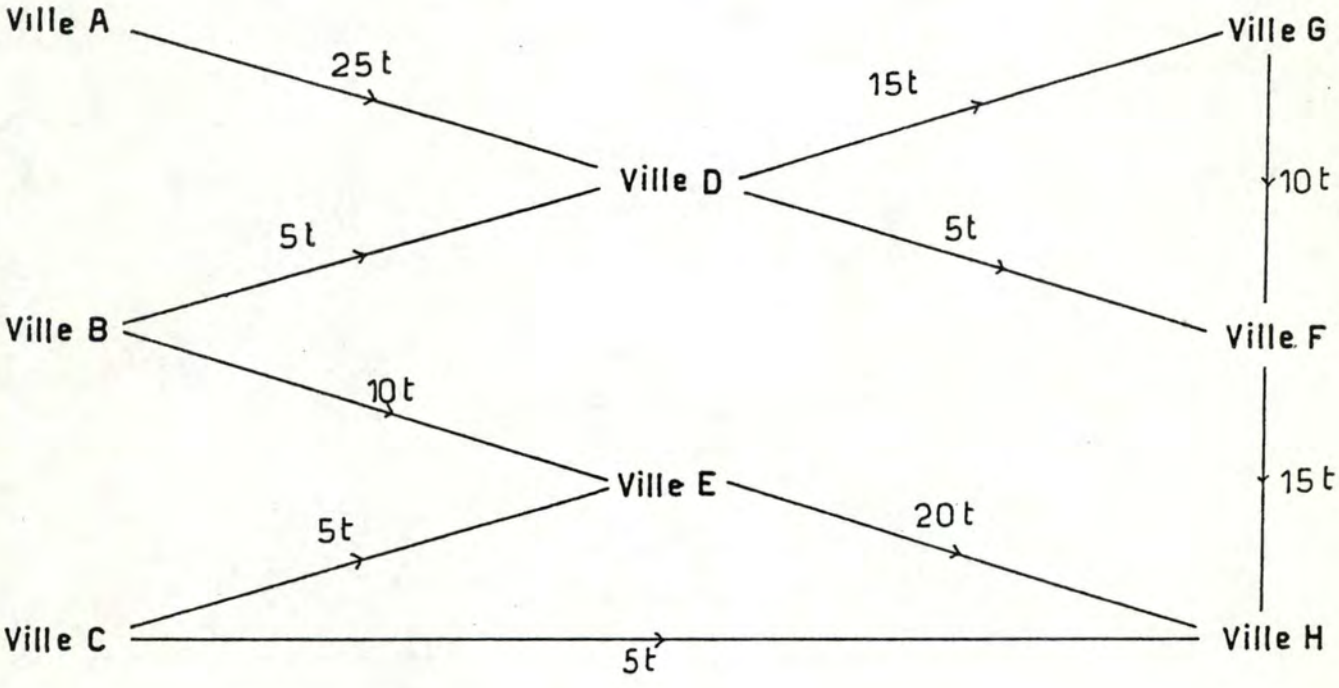
De nombreux problèmes se ramènent à la résolution d'un problème de flot maximum. En guise d'exemples, citons le problème de distribution, le problème du trafic maritime, la bataille de la Marne et le problème des représentants.

EXEMPLE I.2.2. Problème de distribution. Un grand magasin désire approvisionner en marchandises ses succursales à partir de ses entrepôts. Les ressources disponibles dans les entrepôts $e_1, \ldots, e_p$ sont respectivement $r_1, \ldots, r_m$ tonnes; les magasins $m_1, \ldots, m_q$ ne peuvent entreposer respectivement plus de $s_1, \ldots, s_m$ tonnes. Le transport des marchandises est assuré par le grand magasin: ses camions ont un itinéraire fixe et un tonnage limité. Il s'agit d'acheminer une quantité maximale de marchandises des entrepôts aux magasins en utilisant les camions.

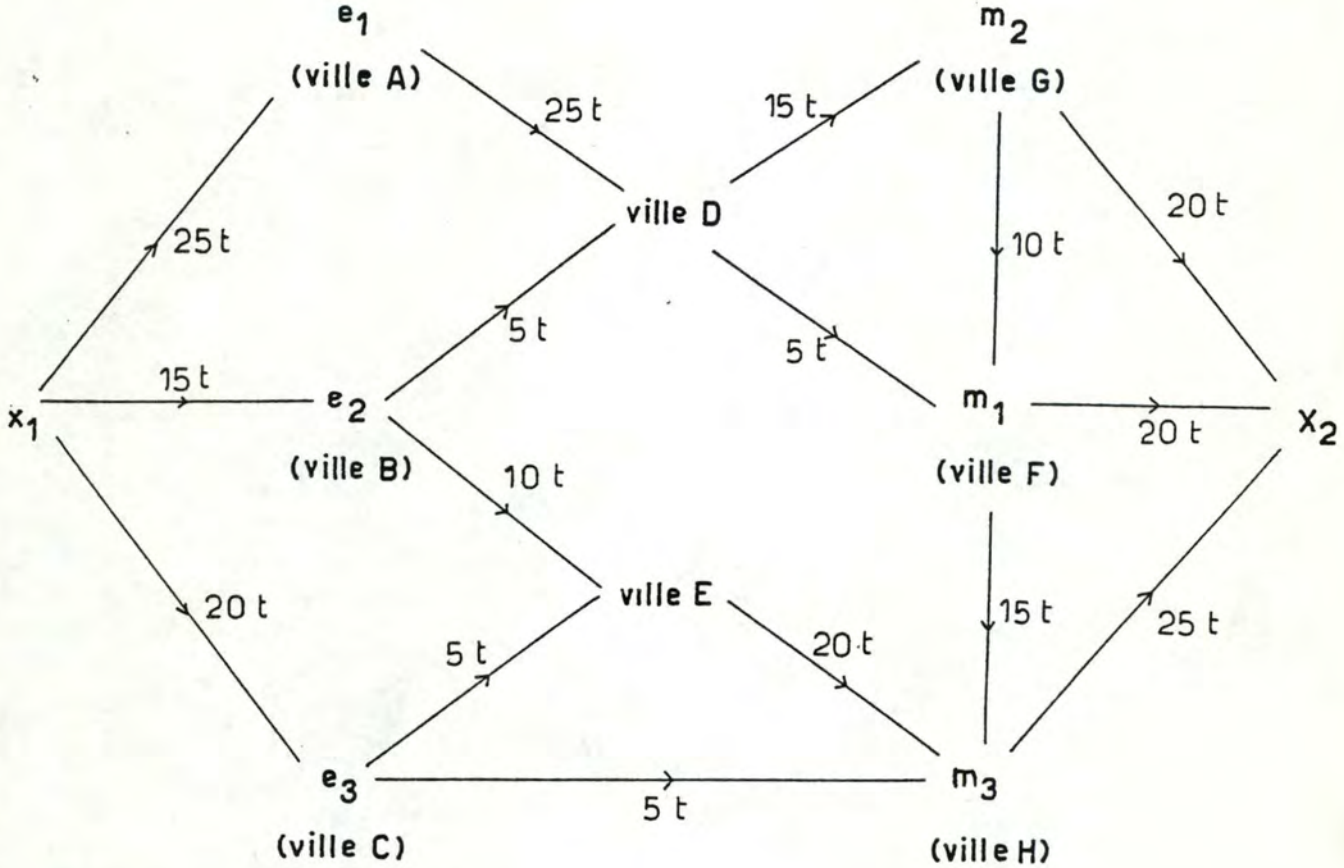
Afin de fixer les idées, particularisons ce problème au cas suivant:

- $p = 3$ et les entrepôts e_1, e_2, e_3 situés respectivement aux villes A, B, C ont respectivement pour ressources 25, 15 et 20 tonnes,

- q = 3 et les magasins respectivement situés aux villes F, G et H ne peuvent respectivement entreposer plus de 20, 20 et 25 tonnes,
- l'itinéraire est fixé par le graphe ci-dessous où les flèches indiquent les relations existantes et les valuations les quantités transportables.



Comme on le vérifie de suite, résoudre ce problème revient à rechercher un flot maximum dans le réseau de transport défini à partir du graphe précédent en introduisant les sommets x_1 (entrée du réseau) et x_2 (sortie du réseau). Les valuations sur les flèches définissent l'application c du réseau.



EXEMPLE I.2.3. Problème du trafic maritime. Des ports maritimes $a_1, \ldots, a_p$, où sont stockésrespectivement $s_1, \ldots, s_p$ litres de pétrole doivent alimenter les ports $b_1, \ldots, b_q$ dont la demande en pétrole est respectivement $d_1, \ldots, d_q$ litres. Le transport du pétrole est effectué par des navires ayant un itinéraire fixe et ne pouvant transporter plus de t_{ij} tonnes entre les ports a_i et b_j. Est-il possible de satisfaire à toutes les demandes?

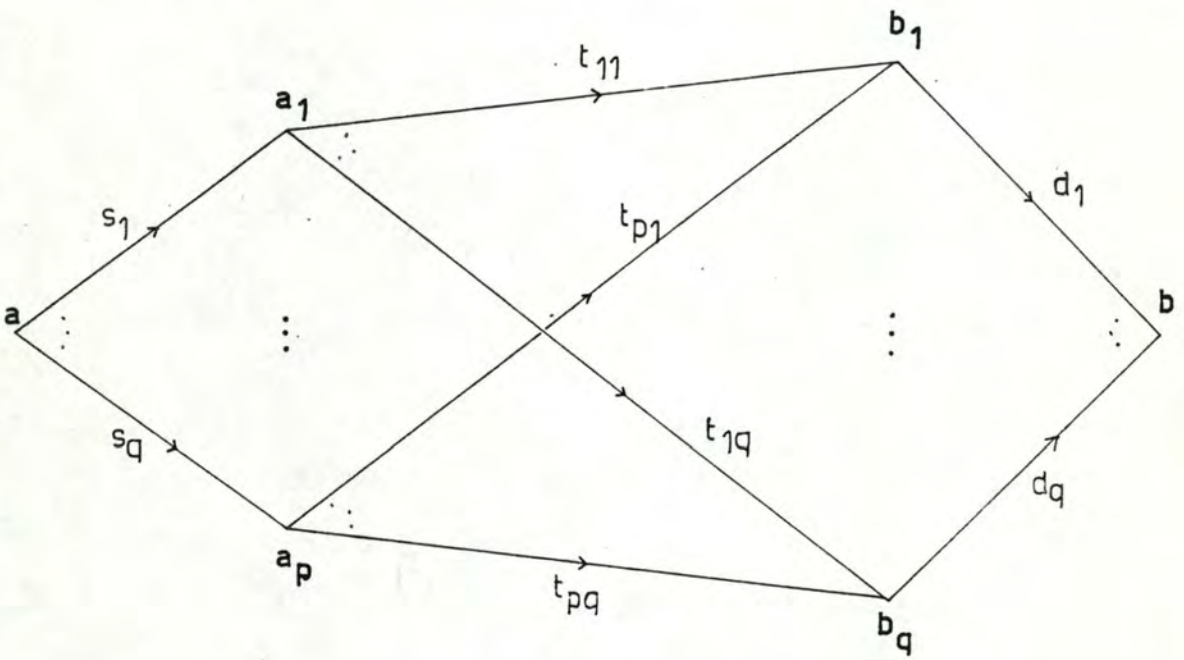
Comment organiser les expéditions?

Répondre à ces questions revient à rechercher un flot maximum dans le réseau de transport $(\{a,a_1, \ldots ,a_p,b_1, \ldots ,b_q,b\}, \{(a,a_i) : i\in\{1, \ldots ,p\}\} \cup \{(a_i,a_j) : i\in\{1, \ldots ,p\}, j\in\{1, \ldots ,q\}\},c)$ où l'application c est définie par les relations

$$c(a,a_i) = s_i \quad \forall i\in\{1, \ldots ,p\}$$

$$c(b_j,b) = d_j \quad \forall j\in\{1, \ldots ,q\}$$

$$c(a_i,b_j) = t_{ij} \quad \forall i\in\{1, \ldots ,p\},\ j\in\{1, \ldots ,q\}$$



EXEMPLE I.2.4. La bataille de la Marne. De différentes villes $a_1, \ldots , a_n$ partent des voitures désirant se rendre en une destination unique b. S'il existe une route reliant directement la ville a_i et la ville a_j (resp. b), c_{ij} (resp. c_{ib}) représente le nombre de véhicules pouvant partir de a_i vers a_j (resp. b) par unité de temps et t_{ij} (resp. t_{ib}) le temps du parcours. Le nombre s_i représente le

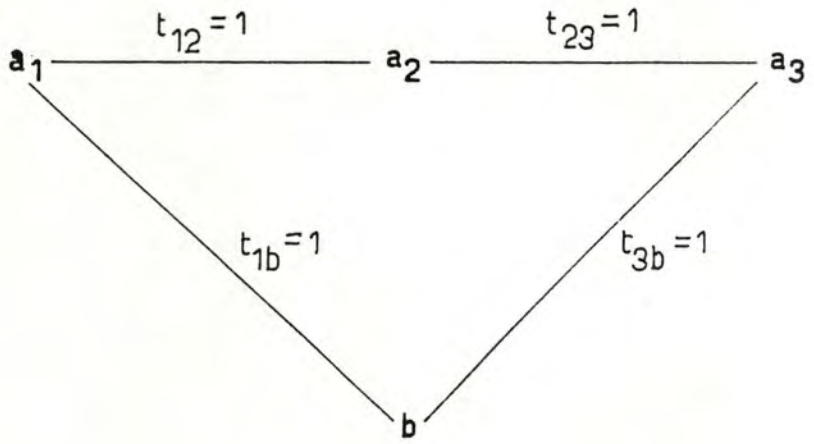
nombre de voitures disponibles en a_i au temps initial et c_i le nombre de voitures pouvant stationner en a_i.

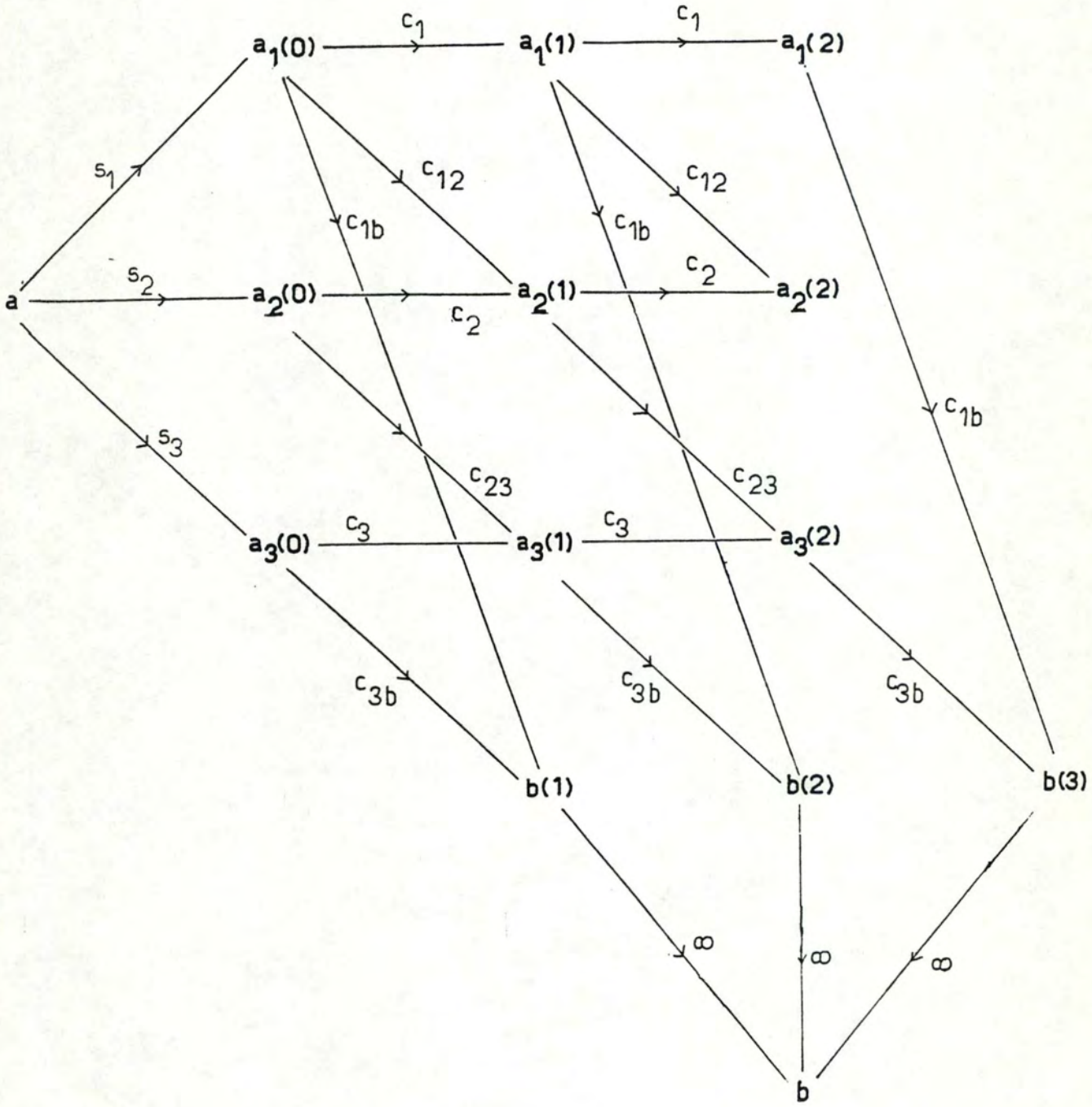
Comment organiser le trafic routier pour que, dans un intervalle de temps donné T le nombre de voitures arrivées en b soit aussi grand que possible?

Ce problème a été étudié par R. Fulkerson sous le nom de réseaux dynamiques. On peut le ramener à un problème de flot maximum en considérant dans l'espace-temps les sommets $a_i(t)$ pour $i \in \{1, \ldots, n\}$ et $t \in \{0,1, \ldots, T\}$. Il suffit alors de joindre, si $t < T$, le sommet $a_i(t)$ au sommet $a_i(t+1)$ par un arc de capacité c_i et de tracer, s'il existe une route reliant a_i à a_j, un arc d'extrémité initiale $a_i(t)$ et d'extrémité terminale $a_j(t+t_{ij})$, de capacité c_{ij}. Pour obtenir un réseau de transport, il reste enfin à ajouter une entrée a, une sortie b et des arcs $(a,a_i(o))$ de capacité s_i et $(b(t),b)$ de capacité infinie. Le flot maximum dans ce réseau de transport déterminera le trafic routier optimum.

Ajoutons que cet exemple est très caractéristique et constitue un bon modèle pour beaucoup de problèmes de transports (routiers, ferroviaires, ...).

Le graphe ci-dessous donne un exemple d'un tel réseau dans le cas suivant où $n = 3$ et $T = 3$.




a
$a_1(0)$
$a_1(1)$
$a_1(2)$
$a_2(0)$
$a_2(1)$
$a_2(2)$
$a_3(0)$
$a_3(1)$
$a_3(2)$
b(1)
b(2)
b(3)
b
s_1
s_2
s_3
c_1
c_2
c_3
c_{12}
c_{23}
c_{1b}
c_{3b}
∞

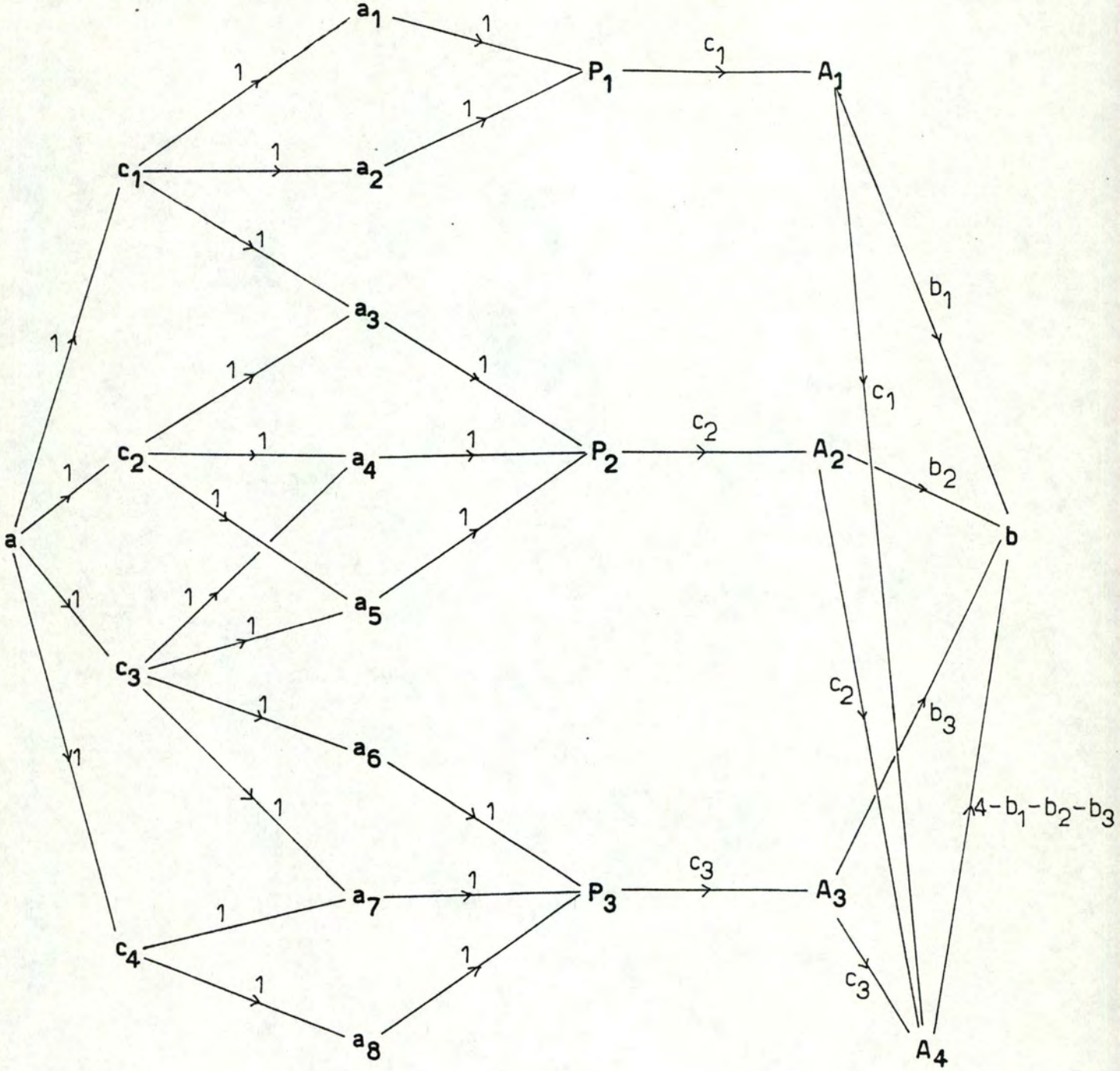
EXEMPLE I.2.5. Le problème des représentants. Les individus d'une société A se répartissent dans un certain nombre de clubs $C_1, \ldots, C_q$, formant ainsi des sous-ensembles de A non nécessairement disjoints, et un certain nombre de partis politiques $P_1, \ldots, P_r$ qui représentent des sous-ensembles disjoints de A. Chaque club choisit parmi ses membres un représentant disposant de pouvoirs spéciaux; un individu ne peut représenter qu'un seul club même s'il appartient à plusieurs.

Comment doit-on choisir ce système de représentants distincts $X = \{x_1, \ldots, x_q\}$ si l'on veut que le nombre de représentants appartenant à chaque parti P_j vérifie les inégalités

$$b_j \leqslant \#(X \cap P_j) \leqslant c_j$$

où b_j et c_j représentent des entiers tels que $\Sigma_{j=1}^{r} b_j \leqslant q$?

Une solution est donnée par la recherche d'un flot maximum, de valeur q, dans un réseau de transport construit comme celui de la figure ci-dessous où $q = 4$, $r = 3$, $A = \{a_1, \ldots, a_8\}$. Les points A_1, A_2, A_3, A_4 y apparaissant sont des points auxilliaires; les capacités des différents arcs sont indiquées sur ceux-ci.


a
a1
a2
a3
a4
a5
a6
a7
a8
c1
c2
c3
c4
P1
P2
P3
A1
A2
A3
A4
b
1
b1
b2
b3
4-b1-b2-b3

La résolution de certains problèmes impose non seulement la recherche d'un flot f dans un réseau de transport mais aussi des contraintes sur les flux f(u), $u \in U$, du type

$$b(u) \leqslant f(u) \quad , \quad \forall u \in U$$

où les réels positifs b(u) satisfont aux relations $b(u) \leqslant c(u)$, pour tout $u \in U$. Ainsi en est-il en pratique pour le problème de distribution I.2.2. et pour le problème de trafic maritime I.2.3. .

Avant de les expliciter, introduisons tout d'abord la définition suivante.

DEFINITION I.2.6. Soient R = (X,U,c) un réseau de transport et $b: U \to \mathbb{R}^+$ une fonction admissible.

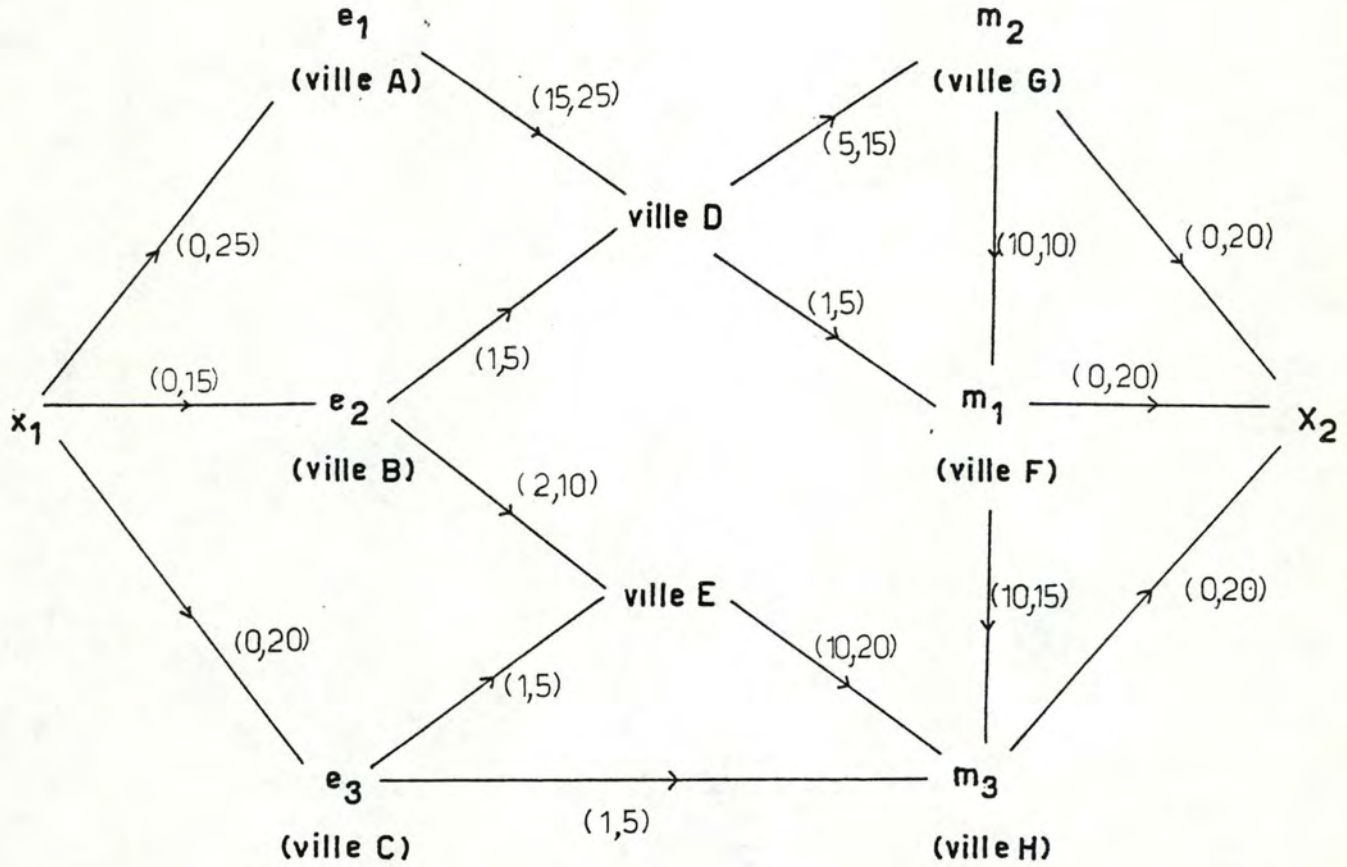
a) Un <u>flot</u> f est <u>b-canalisé dans</u> R s'il vérifie les inégalités

$$\forall u \in U: f(u) \geqslant b(u)$$

b) Le <u>problème du flot b-canalisé maximum</u> dans R consiste à rechercher dans l'ensemble des flots b-canalisés que l'on peut définir dans R, un flot b-canalisé dont la valeur est maximum.

EXEMPLE I.2.7. <u>Problème de distribution</u>. Dans la vie courante, les camions de l'exemple I.2.2. n'assureront le transport de marchandises que si celles-ci atteignent un tonnage minimum.

La résolution de ce problème revient alors à la recherche d'un flot b-canalisé de valeur maximum dans un réseau de transport semblable à celui de la figure ci-dessous.



Les valuations sur les arcs indiquent respectivement les tonnages minimum et maximum que transporteront les différents camions.

EXEMPLE I.2.8. Problème du trafic maritime. De même, en réalité, les ports a_i n'accepteront d'alimenter les ports b_j que si la demande de ceux-ci est au moins de b_{ij} litres de pétrole.

Résoudre ce problème revient alors à la recherche d'un flot b-canalisé de valeur maximum dans le réseau de transport défini en I.2.3., l'application b étant définie par les relations:

$$b(a_i, b_j) = b_{ij} \quad , \quad \forall i \in \{1, \ldots, p\}, \; j \in \{1, \ldots, q\}$$

$$b(a, a_i) = b(b_j, b) = 0 \quad , \quad \forall i \in \{1, \ldots, p\}, \; j \in \{1, \ldots, q\}$$

Il arrive souvent, notamment dans le cas du problème de distribution et du problème du trafic maritime, que l'on veuille déterminer une solution optimale minimisant un certain coût. Le problème du flot maximum de coût minimum permet d'appréhender cette notion. Afin d'en donner une définition formelle, définissons tout d'abord ce que nous entendons par coût d'un flot.

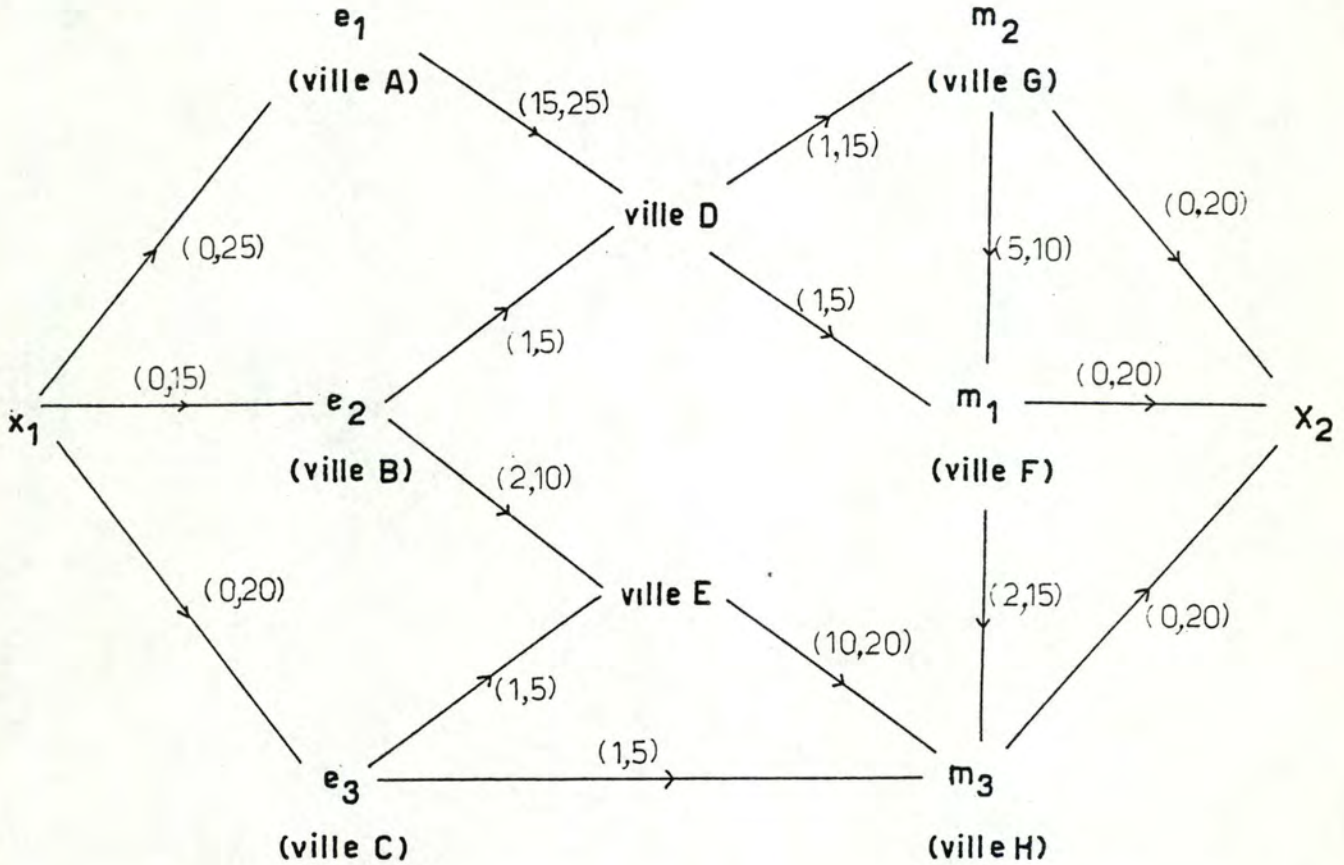
DEFINITION I.2.9. Soient f un flot d'un réseau de transport R et $d: U \to \mathbb{R}^+$ une fonction. Le nombre

$$\sum_{u \in U} d(u) f(u)$$

est le d-coût du flot f.

DEFINITION I.2.10. Soient R un réseau de transport et $d: U \to \mathbb{R}^+$ une fonction de coût. Le problème du flot maximum de coût minimum consiste à rechercher, dans l'ensemble des flots de valeur maximum que l'on peut définir sur R, un flot de d-coût minimum.

EXEMPLE I.2.11. Problème de distribution. En réalité, les déplacements des camions de l'exemple I.2.2. occasionnent certains frais qu'il convient de prendre en considération. La résolution du problème revient alors à la résolution d'un problème de flot maximum de coût minimum dans un réseau de transport semblable à celui de la figure suivante



Les valuations sur les arcs indiquent respectivement les coûts de transport et les tonnages maximums des différents camions.

Remarquons que si on avait voulu considérer des coûts de stockage aux entrepôts e_i ($i = 1, \ldots, 3$) ou aux magasins m_j ($j = 1, \ldots, 3$), il aurait suffit d'introduire des coûts non nuls sur les arcs (x_1, e_i) ou sur les arcs (m_j, x_2). Par ailleurs, il est toujours possible de prendre en compte des coûts de traitement en un sommet. Pour y parvenir, il suffit, tout comme dans la remarque I.1.7., de dédoubler un tel sommet x en deux sommets x',x", d'introduire un arc (x',x") portant un coût unitaire

de traitement et de remplacer tout arc de la forme (y,x) (resp.(x,y)), $y \in X$, par un arc (y,x')(resp.(x'',y)).

EXEMPLE I.2.12. Problème du trafic maritime. De même, dans la vie courante, le transport des marchandises occasionne un certain coût. La résolution du problème revient, si on veut en tenir compte, à la recherche, parmi les flots de valeur maximum, d'un flot de d-coût minimum, où la fonction de coût d est définie par

$$d(a_i,b_j) = d_{ij} \quad , \quad \forall i \in \{1, \ldots ,p\}, j \in \{1, \ldots ,q\}$$

$$d(a,a_i) = b(b_j,b) = 0 \quad , \quad \forall i \in \{1, \ldots ,p\}, j \in \{1, \ldots ,q\}$$

Le problème de transfert de Hitchcock constitue un autre exemple du problème de flot maximum de coût minimum.

DEFINITION I.2.13. Un réseau de Hitchcock est un réseau de transport $R = (X,U,c)$ vérifiant les propriétés suivantes

(i) $X = \{e,e_1, \ldots ,e_m,t_1, \ldots ,t_n,t\}$ avec $m,n \geq 1$

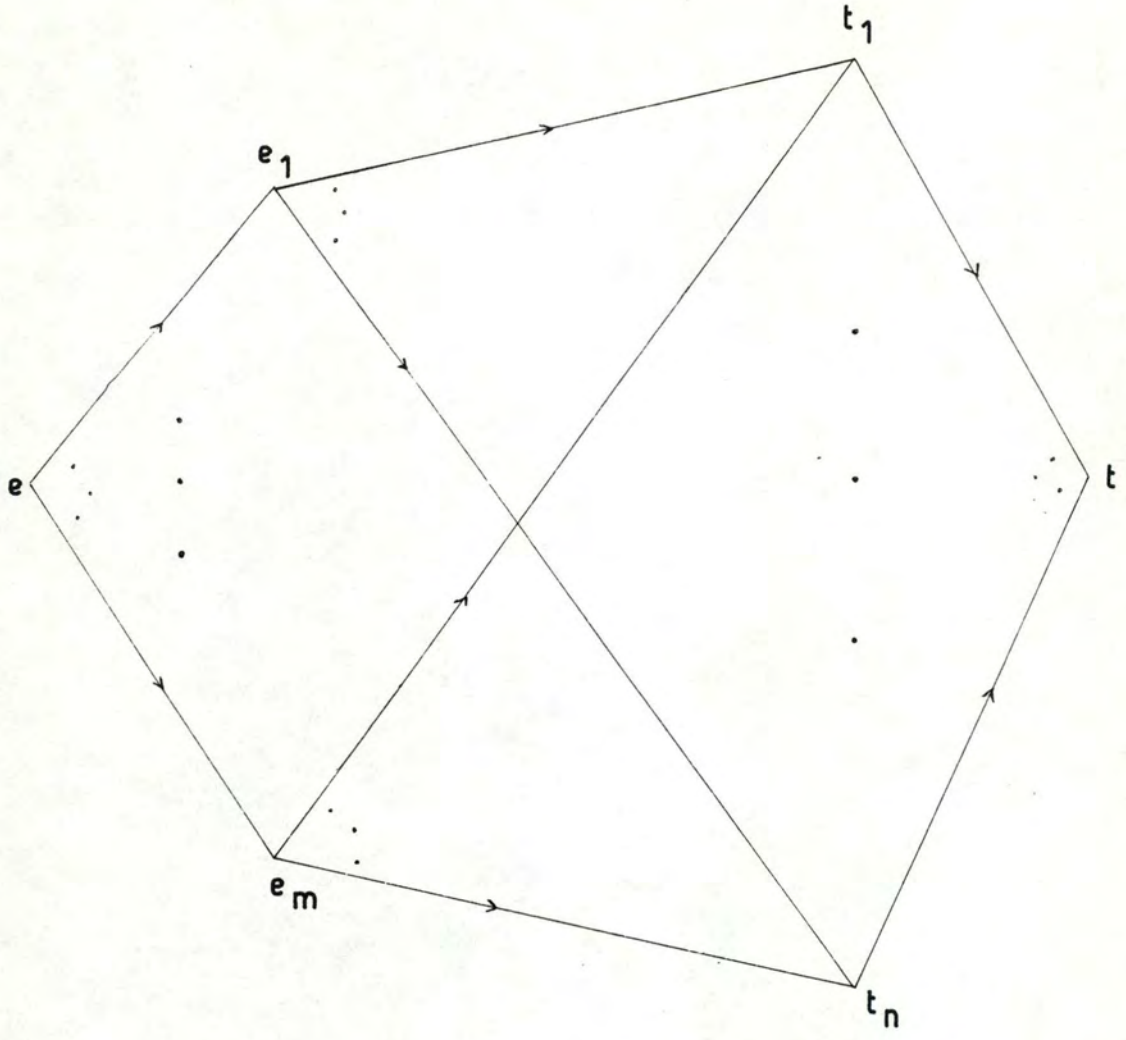
(ii) $U = \{(e,e_i): i = 1, \ldots ,m\} \cup \{(e_i,t_j): i = 1, \ldots ,m;$
$j = 1, \ldots ,n\} \cup$

$\{(t_j,t): j = 1, \ldots ,n\}$

(iii) $\forall i \in \{1, \ldots ,m\}, j \in \{1, \ldots ,n\}: c(e_i,t_j) = +\infty$

(iv) $\sum_{i=1}^{m} c(e,e_i) = \sum_{j=1}^{n} c(t_j,t)$

Un tel réseau peut se schématiser comme suit:



La valeur d'un flot maximum est bien sûr $\sum_{i=1}^{m} c(e,e_i)$.

DEFINITION I.2.14. Le <u>problème de transfert de Hitchcock</u> consiste à résoudre le problème du flot maximum de coût minimum dans un réseau de Hitchcock $(\{e,e_1, \ldots ,e_m,t_1, \ldots ,t_n,t\},U,c)$, la fonction de coût d vérifiant les égalités

$$d(e,e_i) = 0 \quad , i = 1, \ldots ,m$$

$$d(t_j,t) = 0 \quad , j = 1, \ldots ,n$$

REMARQUE I.2.15. L'interprétation du problème de transfert de Hitchcock est bien connue. Chaque sommet e_i ($i = 1, \ldots, m$) correspond à une source à laquelle $c(e,e_i)$ unités de marchandises sont disponibles. Chaque sommet t_j correspond à une destination demandant $c(t_j,t)$ unités de marchandises. Le coût unitaire de transport de e_i à t_j est $d(e_i,t_j)$. Une solution est recherchée qui minimise le coût de transport.

EXEMPLE I.2.16. Comme on le vérifie de suite, le problème du trafic maritime I.2.12. constitue un problème de Hitchcock.

CONVENTION I.2.17. Dans la suite de ce travail, sauf mention explicite du contraire, les appellations

flot b-canalisé, d-coût

sous-entendent respectivement que b est une fonction admissible et que d est une fonction définie de U dans $\mathbb{R}^+$. En outre, pour tout réseau de Hitchcock, nous noterons systématiquement

$$e; e_1, \ldots, e_m ; t_1, \ldots, t_n; t$$

respectivement l'entrée, l'ensemble des sources, l'ensemble des destinations et la sortie; et poserons

$$a_i = c(e,e_i) \quad , \quad \forall i \in \{1, \ldots, m\}$$

$$b_j = c(t_j,t) \quad , \quad \forall j \in \{1, \ldots, n\}$$

I.3. Le théorème du flot maximum et de la coupe minimum

Le théorème du flot maximum et de la coupe minimum joue un rôle fondamental dans la résolution du problème du flot maximum. Afin de l'établir, introduisons tout d'abord les notions de coupe de réseau et de capacité de coupe de réseau.

DEFINITION I.3.1.

a) La coupe du réseau de transport R associé à un sous-ensemble $A \subset X$ tel que $e \notin A$ et $t \in A$ est l'ensemble $\omega^-(A)$ des arcs incidents à A vers l'intérieur.

b) La capacité de la coupe $\omega^-(A)$ est la quantité

$$c(A) = c(\omega^-(A)) = \sum_{u \in \omega^-(A)} c(u)$$

NOTATION I.3.2. Soient A,B deux sous-ensembles de X, C un sous-ensemble de U et $\alpha: U \to \mathbb{R}^+$ une fonction. Dans la suite de ce travail, nous posons

$$\alpha(A,B) = \sum_{\substack{a\ A \\ b\ B \\ (a,b) \in U}} \alpha(a,b)$$

et

$$\alpha(C) = \sum_{u \in C} \alpha(u) .$$

En outre, nous notons,

$$\bar{A}$$

le complémentaire de A dans X.

LEMME I.3.3. Pour tout flot f et tout ensemble S tel que $e \in S$ et $t \notin S$, on a

$$|f| = f(S,\bar{S}) - f(\bar{S},S).$$

En particulier, pour tout flot f, tout flot b-canalisé f_b et toute coupe $\omega^-(A)$, on a

$$|f| \leqslant c(\omega^-(A))$$

et

$$|f_b| \leqslant c(\omega^-(A)) - b(\omega^+(A))$$

Démonstration. De fait, comme on a $\omega^+(\{x_n\}) = \phi$, il vient, pour tout flot g

$$|g| = \sum_{u \in \omega^-(\{x_n\})} g(u)$$

$$= \sum_{u \in \omega^-(\{x_n\})} g(u) + \sum_{x \in \overline{S} \setminus \{x_n\}} \left(\sum_{u \in \omega^-(\{x\})} g(u) - \sum_{u \in \omega^+(\{x\})} g(u) \right)$$

$$= \sum_{x \in \overline{S}} \sum_{u \in \omega^-(\{x\})} g(u) - \sum_{x \in \overline{S}} \sum_{u \in \omega^+(\{x\})} g(u)$$

$$= \sum_{x \in \overline{S}} \sum_{u \in \omega^-(\{x\})} g(u) - \sum_{u \in U \cap \overline{S} \times \overline{S}} g(u)$$

$$- \sum_{x \in \overline{S}} \sum_{u \in \omega^+(\{x\})} g(u) - \sum_{u \in U \cap \overline{S} \times \overline{S}} g(u)$$

$$= \sum_{\substack{s \in S \\ x \in \overline{S} \\ (s,x) \in U}} g(s,x) - \sum_{\substack{s \in S \\ x \in \overline{S} \\ (x,s) \in U}} g(x,s)$$

$$= g(S,\overline{S}) - g(\overline{S},S)$$

La thèse résulte alors du fait qu'on a, pour tout flot f et tout flot b-canalisé f_b,

$$c(S,\overline{S}) \geqslant f(S,\overline{S}) - f(\overline{S},S)$$

$$c(S,\overline{S}) - b(\overline{S},S) \geqslant f_b(S,\overline{S}) - f_b(\overline{S},S)$$

et de la définition de la coupe d'un réseau.

THEOREME DU FLOT MAXIMUM ET DE LA COUPE MINIMUM I.3.4. Si un flot f et une coupe $\omega^-(A)$ sont tels que

$$|f| = c(\omega^-(A))$$

alors, f est de valeur maximum et $\omega^-(A)$ de capacité minimum.
Démonstration. C'est une conséquence immédiate du lemme précédent. ///

THEOREME DU FLOT B-CANALISE MAXIMUM I.3.5. Si un flot b-canalisé f_b et une coupe $\omega^-(A)$ sont tels que

$$|f_b| = c(\omega^-(A)) - b(\omega^+(A))$$

alors f_b est un flot b-canalisé maximum.

Démonstration. Cela résulte aussitôt du lemme précédent. ///

I.4. La complexité d'algorithme

Dans les chapitres suivants, nous présentons différents algorithmes capables de résoudre le problème du flot maximum. Afin de donner un sens précis à l'idée intuitive de "coût" qu'ils occasionnent, nous rappelons succinctement la notion temporelle théorique d'algorithme. Nous renvoyons le lecteur désireux d'en connaître davantage à [26].

DEFINITION I.4.1. Soient un algorithme A et un paramètre p caractérisant la taille des données auxquelles A s'applique dans chaque cas. Soient, en outre, T(p) le temps d'exécution de l'algorithme A exprimé en fonction de p et f(p) une fonction de p.

a) L'algorithme A est de complexité temporelle théorique $\mathcal{O}(f(p))$ si la limite

$$\lim_{p \to +\infty} \frac{T(p)}{f(p)}$$

existe, est finie et diffère de 0.

b) L'algorithme A est de complexité temporelle théorique $\mathcal{O}(f(p))$ au plus s'il existe une constante C et une valeur p_o du paramètre p telles que

$$T(p) \leqslant C f(p), \quad \forall p \geqslant p_o.$$

Ces définitions se généralisent aisément au cas où le temps d'exécution dépend de plusieurs paramètres.

DEFINITION I.4.2. Soient $T(p_1, \ldots, p_n)$ le temps d'exécution de l'algorithme A exprimé en fonction des paramètres $p_1, \ldots, p_n$ et $f(p_1, \ldots, p_n)$.

a) L'algorithme A est de <u>complexité temporelle théorique</u> $\mathcal{O}(f(p_1, \ldots, p_n))$ si, pour tout $i \in \{1, \ldots, n\}$, la limite

$$\lim_{p_i \to +\infty} \frac{T(p_1, \ldots, p_n)}{f(p_1, \ldots, p_n)}$$

existe, est finie et diffère de 0.

b) L'algorithme A est de <u>complexité temporelle théorique</u> $\mathcal{O}(f(p_1, \ldots, p_n))$ <u>au plus</u> si, pour tout $i \in \{1, \ldots, n\}$, il existe une constante $C(p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n)$ éventuellement fonction des paramètres $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots p_n$ et une valeur $p_{i,o}$ du paramètre p_i telles que

$$T(p_1, \ldots, p_n) \leqslant C(p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n)\, f(p_1, \ldots, p_n),$$

pour tout $p_i \geqslant p_{i,o}$.

On peut définir de même la notion de complexité spatiale théorique. Toutefois, nous nous limiterons au concept temporel et parlerons parfois, par souci de concision, simplement de complexité.

CHAPITRE II

LES METHODES D'ETIQUETAGE

II.1. Le graphe d'écart

DEFINITION II.1.1. Soient $R = (X,U,c)$ un réseau de transport et f un flot dans R. Le graphe d'écart $R(f)$ associé à f est le graphe $(X,U(f))$ dont l'ensemble des arcs $U(f)$ est obtenu en associant à chaque arc $u = (x,y) \in U$, l'arc $u = (x,y)$ si $f(u) < c(u)$ et l'arc $\overleftarrow{u} = (y,x)$ si $f(u) > 0$.

L'intérêt du graphe d'écart apparaît dans le théorème suivant.

THEOREME II.1.2. Un flot f d'un réseau de transport R est de valeur maximale si et seulement si le graphe d'écart $R(f)$ associé à f ne contient pas de chemin d'extrémité initiale e et d'extrémité terminale t.

Démonstration. La condition est nécessaire. Procédons par l'absurde. Si ce n'est pas le cas, il existe, dans $R(f)$, un chemin $y_1, \ldots, y_p$ d'extrémités initiale e et terminale t grâce auquel on peut construire la fonction $f' : U \to \mathbb{R}^+$ définie par les égalités

$$u \in U : f'(u) = \begin{cases} f(u) + m & \\ & \text{s'il existe } i \in \{1, \ldots, p-1\} \text{ tel que} \\ & u = (y_i, y_{i+1}) \text{ et } (y_i, y_{i+1}) \in U \text{ et } (y_{i+1}, y_i) \notin U \\ f(u) - m & \\ & \text{s'il existe } i \in \{1, \ldots, p-1\} \text{ tel que} \\ & u = (y_i, y_{i+1}) \text{ et } (y_i, y_{i+1}) \notin U \text{ et } (y_{i+1}, y_i) \in U \\ f(u) + \min\{m, c(u) - f(u)\} & \\ & \text{s'il existe } i \in \{1, \ldots, p-1\} \text{ tel que} \\ & u = (y_i, y_{i+1}) \text{ et } (y_i, y_{i+1}) \in U \text{ et } (y_{i+1}, y_i) \in U \\ f(u) - \max\{0, m - c(y_i, y_{i+1}) + f(y_i, y_{i+1})\} & \\ & \text{s'il existe } i \in \{1, \ldots, p-1\} \text{ tel que} \\ & u = (y_{i+1}, y_i) \text{ et } (y_i, y_{i+1}) \in U \text{ et } (y_{i+1}, y_i) \in U \\ f(u) & \\ & \text{sinon} \end{cases}$$

où $m = \min\{A, B, C\}$ avec

$$A = \min_{\substack{i = 1, \ldots, p-1 \\ (y_i, y_{i+1}) \in U \\ (y_{i+1}, y_i) \notin U}} c(y_i, y_{i+1}) - f(y_i, y_{i+1})$$

$$B = \min_{\substack{i = 1, \ldots, p-1 \\ (y_i, y_{i+1}) \notin U \\ (y_{i+1}, y_i) \in U}} f(y_{i+1}, y_i)$$

$$C = \min_{\substack{i = 1, \ldots, p-1 \\ (y_i, y_{i+1}) \in U \\ (y_{i+1}, y_i) \in U}} c(y_i, y_{i+1}) - f(y_i, y_{i+1}) + f(y_{i+1}, y_i)$$

Pour conclure, il suffit de prouver que f' est un flot de valeur strictement supérieure à $|f|$. D'une part, f' est, comme on le vérifie

de suite, un flot. D'autre part, de l'égalité

$$|f'| = |f| + (f'(y_{p-1},y_p) - f(y_{p-1},y_p))$$

et de la définition de $f'(y_{p-1},y_p)$ on déduit de suite qu'on a

$$|f'| = |f| + m$$

ce qui suffit puisqu'on a, vu la définition de $R(f)$, $m > 0$.

La condition est suffisante. Pour le démontrer, considérons l'ensemble A des sommets non accessibles de e dans R(f). On a, bien sûr, par hypothèse, $e \notin A$ et $t \in A$ et, par conséquent, $\omega^-(A)$ est une coupe du réseau R. Prouvons en outre qu'on a

$$\forall (u,v) \in U : u \notin A \text{ et } v \in A \Longrightarrow f(u,v) = c(u,v) \qquad (*)$$

$$\forall (u,v) \in U : u \in A \text{ et } v \notin A \Longrightarrow f(u,v) = 0 \qquad (**).$$

En effet, les égalités (*) résultent de suite du fait que si deux sommets $u \notin A$ et v sont tels que $(u,v) \in U$ et $f(u,v) < c(u,v)$ alors (u,v) appartient à U(f) et, vu la définition de A, v n'appartient pas à A. De plus, les égalités (**) résultent du fait que si deux sommets u et $v \notin A$ sont tels que $(u,v) \in U$ et $f(u,v) > 0$, alors (v,u) appartient à U(f) et, vu la définition de A, u n'appartient pas à A. Dans ces conditions, le lemme I.3.3. établit les égalités

$$\begin{aligned} |f| &= f(\bar{A},A) - f(A,\bar{A}) \\ &= \sum_{\substack{(x,y)\in U \\ x\notin A, y\in A}} f(x,y) - \sum_{\substack{(x,y)\in U \\ x\in A, y\notin A}} f(x,y) \\ &= \sum_{\substack{(x,y)\in U \\ x\notin A, y\in A}} c(x,y) - 0 \\ &= \sum_{u\in\omega^-(A)} c(u) = c(\omega^-(A)). \end{aligned}$$

La thèse résulte alors de l'application du théorème du flot maximum et de la coupe minimum I.3.4.. ///

La démonstration du théorème précédent fournit un algorithme de construction d'un flot de valeur strictement supérieur à la valeur

d'un flot (de valeur non maximum) donné. Nous l'écrivons comme une routine à deux paramètres : le flot f supposé de valeur non maximale et $(y_1, \ldots, y_p)$ un chemin d'extrémités initiale e et terminale t dans R(f).

ALGORITHME AUGMENTATION(f,$(y_1, \ldots, y_p)$) II.1.3. Soient f un flot dans R et $(y_1, \ldots, y_p)$ un chemin d'extrémités initiale e et terminale t dans R(f).

$m := +\infty$
Pour i variant de 1 à p-1 effectuer
  si $(y_i, y_{i+1}) \in U$ *et* $(y_{i+1}, y_i) \notin U$ *alors*
    $min := c(y_i, y_{i+1}) - \ell(y_i, y_{i+1})$
  si $(y_i, y_{i+1}) \notin U$ *et* $(y_{i+1}, y_i) \in U$ *alors*
    $min := \ell(y_{i+1}, y_i)$
  si $(y_i, y_{i+1}) \in U$ *et* $(y_{i+1}, y_i) \in U$ *alors*
    $min := c(y_i, y_{i+1}) - \ell(y_i, y_{i+1}) + \ell(y_{i+1}, y_i)$
  si $m > min$ *alors*
    $m := min$
Pour tout $u \in U$ *effectuer* $\ell'(u) := \ell(u)$
Pour i variant de 1 à p-1 effectuer
  si $(y_i, y_{i+1}) \in U$ *et* $(y_{i+1}, y_i) \notin U$ *alors*
    $\ell'(y_i, y_{i+1}) := \ell(y_i, y_{i+1}) + m$
  si $(y_i, y_{i+1}) \notin U$ *et* $(y_{i+1}, y_i) \in U$ *alors*
    $\ell'(y_{i+1}, y_i) := \ell(y_{i+1}, y_i) - m$
  si $(y_i, y_{i+1}) \in U$ *et* $(y_{i+1}, y_i) \in U$ *alors*
    $\ell'(y_i, y_{i+1}) := \ell(y_i, y_{i+1}) + \min\{m, c(y_i, y_{i+1}) - \ell(y_i, y_{i+1})\}$
    $\ell'(y_{i+1}, y_i) := \ell(y_{i+1}, y_i) - \max\{0, m - c(y_i, y_{i+1}) + \ell(y_i, y_{i+1})\}$

La proposition suivante résulte du théorème II.1.2.

PROPOSITION II.1.4. L'algorithme AUGMENTATION appliqué à un flot f dans R et à un chemin $(y_1, \ldots, y_p)$ d'extrémités initiale e et terminale t dans R(f) détermine un flot f' de valeur strictement supérieure à $|f|$. De plus, f' est à valeurs entières s'il en est ainsi du flot f et de la fonction c.

Démonstration. On prouve cette proposition comme la nécessité de la condition énoncée en II.1.2.. ///

CONVENTION II.1.5. Dans la suite de ce travail, nous appelons

chaîne f-augmentante

toute chaîne de R constituant dans R(f) un chemin d'extrémités initiale e et terminale t. En outre, nous appelons

augmentation de flot le long de la chaîne f-augmentante α

ou plus simplement

augmentation de flot

l'opération qui consiste à remplacer le flot f par le flot f' résultant de l'exécution de l'algorithme AUGMENTATION appliqué à f et α .

II.2. L'algorithme de Ford et Fulkerson

Les résultats précédents suggèrent un algorithme de construction d'un flot de valeur maximum. Partant d'un flot initial f_0 (par exemple en posant $f_0(u) = 0$, pour tout $u \in U$), on détermine de proche en proche les éléments d'une suite $f_0, f_1, \ldots, f_N$ de flots telle que $|f_i| < |f_{i+1}|$ pour tout $i \in \{1, \ldots, N-1\}$, la suite se terminant lorsqu'un flot maximum a été obtenu.

Cet algorithme a été découvert en premier lieu par L.R. Ford et D.R. Fulkerson en 1962. Il est connu sous le nom d'algorithme de Ford et Fulkerson.

ALGORITHME DE FORD ET FULKERSON II.2.1.

Choisir un flot initial f_0 (par exemple, en posant $f(u) = 0$, pour tout $u \in U$);
$k := 0$;
Tant qu'il existe une chaine f_k-augmentante répéter
  choisir une telle chaine, soit $(y_1, \ldots, y_p)$
  $f_{k+1} := AUGMENTATION(f_k, (y_1, \ldots, y_p))$
  $k := k+1$.

THEOREME II.2.2. Si les capacités des arcs du réseau de transport R sont entières alors l'algorithme de Ford et Fulkerson exécuté à partir d'un flot initial à valeurs entières converge et détermine, après exécution, un flot dans R de valeur maximum et à valeurs entières.

Démonstration. Etablissons tout d'abord que l'algorithme de Ford et Fulkerson converge. De fait, tout flot f_k de la suite construite par l'algorithme II.2.1. est, vu la proposition II.1.4., à valeurs entières et vérifie $|f_k| < |f_{k+1}|$ ce qui suffit, puisqu'on a, vu le lemme I.3.3., pour tout flot f_k,

$$|f_k| \leqslant c(\omega^-(\{t\})) .$$

Pour conclure, il reste à établir que le flot f_N déterminé après exécution est de valeur maximum. Ceci résulte de suite du théorème II.1.2. et du fait qu'il n'existe pas de chaine f_N-augmentante. ///

REMARQUE II.2.3. L'algorithme de Ford et Fulkerson reste applicable à un réseau de transport pour lequel les capacités sont mutuellement commensurables. Il suffit, en effet, dans ce cas, de multiplier ces capacités par une constante C assez grande pour obtenir des capacités entières. L'application de l'algorithme II.2.1. au réseau ainsi obtenu détermine un flot de valeur maximum dont les flux et la valeur correspondent aux flux et à la valeur d'un flot maximum du réseau initial multipliés par C

Une façon de résoudre le problème du flot maximum dans le cas contraire consiste à multiplier par une constante C' assez grande les capacités des arcs, à approximer les capacités obtenues aux nombres entiers les plus proches et à appliquer au nouveau réseau l'algorithme II.2.1.. Les flux et la valeur du flot maximum obtenu divisé par C' constituent souvent une excellente approximation des flux et de la valeur d'un flot de valeur maximum du réseau donné.

L'algorithme de Ford et Fulkerson permet aussi de résoudre le problème du flot b-canalisé maximum.

Il semble judicieux dans cette optique d'introduire dans l'algorithme II.2.1. quelques modifications mineures de façon à tenir compte des bornes b(u). Toutefois, ceci suppose l'existence d'un flot b-canalisé de départ, ce dont nous ne sommes pas assuré.

Afin d'apporter une solution à ce problème d'existence, remarquons tout d'abord que si l'on essaye de passer d'un flot dans R à un flot f* dans R en posant $f^*(u) = f(u) - b(u)$ pour tout $u \in U$, l'équation de conservation de la somme des flux en un sommet x_i

$$\sum_{u\in\omega^-(x_i)} f(u) - \sum_{u\in\omega^+(x_i)} f(u) = 0$$

devient

$$\sum_{u\in\omega^-(x_i)} f^*(u) - \sum_{u\in\omega^+(x_i)} f^*(u) - \left(\sum_{u\in\omega^-(x_i)} b(u) - \sum_{u\in\omega^+(x_i)} b(u) \right) = 0$$

ce qui signifie que les réels $f^*(u)$ ne constituent pas dans tous les cas un flot. Cependant l'équation précédente devient une condition de conservation de la somme des flux en x_i s'il existe lorsque

$$\varepsilon_{i,b} = \sum_{u\in\omega^-(x_i)} b(u) - \sum_{u\in\omega^+(x_i)} b(u) > 0 \quad (\text{resp. } \varepsilon_{i,b} < 0)$$

un arc v incident à x_i vers l'intérieur (resp. l'extérieur) sur lequel circule un flux $f^*(v) = \varepsilon_{i,b}$ (resp. $f^*(v) = -\varepsilon_{i,b}$).

De là l'idée d'introduire le réseau b-auxiliaire de R.

CONVENTION II.2.4. Soient $R = (X,U,c)$ un réseau de transport et $b : U \to \mathbb{R}$ une fonction admissible. Dans la suite de ce travail, nous posons, pour tout $i \in \{2, \ldots, n-1\}$

$$\varepsilon_{i,b} = b(\omega^-(x_i)) - b(\omega^+(x_i))$$

et appelons réseau b-auxiliaire $R^* = (X^*,U^*,c^*)$ associé à R le réseau de transport défini par les égalités

$$X^* = X \cup \{x_o, x_{n+1}\}$$

$$U^* = U \cup \{(x_o,x_i) : 2 \leqslant i \leqslant n-1,\ \varepsilon_{i,b} > 0\}$$
$$\cup \{(x_i,x_{n+1}) : 2 \leqslant i \leqslant n-1, \varepsilon_{i,b} > 0\} \cup \{(x_n,x_1)\}$$

$$c^*(u) = c(u) - b(u), \quad \forall u \in U$$

$$c^*(x_o,x_i) = \varepsilon_{i,b}, \quad \forall i \in \{2, \ldots, n-1\} : \varepsilon_{i,b} > 0$$

$$c^*(x_i, x_{n+1}) = -\varepsilon_{i,b}, \quad \forall i \in \{2, \ldots, n-1\} : \varepsilon_{i,b} > 0$$

$$c^*(x_n, x_1) = +\infty .$$

Les sommets x_o et x_{n+1} sont respectivement l'entrée et la sortie de R*.

De plus, sauf mention explicite du contraire, l'expression

réseau b-auxiliaire associé à R

sous-entend que b est une fonction admissible dans R.

RAPPEL II.2.5. Les arcs incidents vers l'extérieur à l'entrée d'un réseau de transport R et les arcs incidents vers l'intérieur à la sortie de R constituent les arcs terminaux de R.

Donnons à présent une condition nécessaire et suffisante pour qu'il existe un flot b-canalisé dans un réseau de transport.

THEOREME II.2.6. Il existe un flot b-canalisé dans R si et seulement s'il existe un flot saturant les arcs terminaux de R*.

Démonstration. La condition est nécessaire. De fait, si f est un flot b-canalisé dans R, il suffit de poser

$$f^*(u) = f(u) - b(u), \quad \forall u \in U,$$

$$f^*(x_o, x_i) = \varepsilon_{i,b}, \quad \forall (x_o, x_i) \in U^*$$

$$f^*(x_i, x_{n+1}) = -\varepsilon_{i,b}, \quad \forall (x_i, x_{n+1}) \in U^*$$

$$f^*(x_n, x_1) = |f|$$

pour obtenir, comme on le vérifie de suite, un flot saturant les arcs terminaux de R*.

La condition est suffisante. En effet, si f* est un flot saturant les arcs terminaux de R*, le vecteur $(f(u_1), \ldots, f(u_m))$ défini par les égalités

$$f(u) = f^*(u) + b(u), \quad \forall u \in U$$

constitue un flot b-canalisé dans R. ///

THEOREME II.2.7. S'il existe dans R un flot de valeur maximum ne saturant pas les arcs terminaux de R, alors il en est ainsi de tout flot de R.

Démonstration. C'est immédiat. ///

Les deux théorèmes précédents suggèrent les modifications à apporter à l'algorithme de Ford et Fulkerson II.2.1. pour qu'il fournisse un flot b-canalisé de valeur maximum.

Remarquons tout d'abord que si l'on détermine un flot maximum ne saturant pas les arcs terminaux de R*, il n'existe pas de flot b-canalisé dans R. Par contre, si le flot f* sature les arcs terminaux de R*, le flot défini par les égalités

$$f(u) = f^*(u) + b(u), \quad \forall u \in U$$

est b-canalisé. Comme il n'est pas nécessairement maximum, il reste à l'améliorer en utilisant l'algorithme de Ford et Fulkerson où l'on prend f comme flot initial. Toutefois, il convient d'assurer que tous les flots f_k de la suite construite soient b-canalisés, ce qui requiert une modification de l'algorithme AUGMENTATION II.1.3.. Dans ce but, nous introduisons la notion de b-graphe d'écart.

DEFINITION II.2.8. Soient $R = (X,U,c)$ un réseau de transport et f un flot b-canalisé dans R. Le b-graphe d'écart $R_b(f)$ associé à f est le graphe $(X,U_b(f))$ dont l'ensemble des arcs $U_b(f)$ est obtenu en associant à chaque arc $u = (x,y) \in U$, l'arc $u = (x,y)$ si $f(u) < c(u)$ et l'arc $\overleftarrow{u} = (y,x)$ si $f(u) > b(u)$.

L'intérêt du b-graphe d'écart apparaît dans le théorème suivant.

THEOREME II.2.9. Un flot b-canalisé f d'un réseau de transport est de valeur maximale si le b-graphe d'écart $R_b(f)$ associé à f ne contient pas de chemin d'extrémités initiale e et terminale t.

Démonstration. La condition est nécessaire. Procédons par l'absurde. Si ce n'est pas le cas, il existe dans $R_b(f)$ un chemin $y_1, \ldots, y_p$ d'extrémité initiale e et d'extrémité terminale t grâce auquel on peut construire à partir de f, la fonction $f' : U \to \mathbb{R}^+$ définie par les égalités

$$\forall u \in U : f'(u) = \begin{cases} f(u) + m \\ \quad \text{s'il existe } i \in \{1, \ldots, p-1\} \text{ tel que} \\ \quad u = (y_i, y_{i+1}) \text{ et } (y_i, y_{i+1}) \in U \text{ et } (y_{i+1}, y_i) \notin U \\ f(u) - m \\ \quad \text{s'il existe } i \in \{1, \ldots, p-1\} \text{ tel que} \\ \quad u = (y_i, y_{i+1}) \text{ et } (y_i, y_{i+1}) \notin U \text{ et } (y_{i+1}, y_i) \in U \\ f(u) + \min\{m, c(u) - f(u)\} \\ \quad \text{s'il existe } i \in \{1, \ldots, p-1\} \text{ tel que} \\ \quad u = (y_i, y_{i+1}) \text{ et } (y_i, y_{i+1}) \in U \text{ et } (y_{i+1}, y_i) \in U \\ f(u) - \max\{0, m - c(y_i, y_{i+1}) + f(y_i, y_{i+1})\} \\ \quad \text{s'il existe } i \in \{1, \ldots, p-1\} \text{ tel que} \\ \quad u = (y_{i+1}, y\ ) \text{ et } (y_i, y_{i+1}) \in U \text{ et } (y_{i+1}, y_i) \in U \\ f(u) \\ \quad \text{sinon} \end{cases}$$

où $m = \min\{A, B, C\}$ avec

$$A = \min_{\substack{i=1,\ldots,p-1 \\ (y_i, y_{i+1}) \in U \\ (y_{i+1}, y_i) \notin U}} c(y_i, y_{i+1}) - f(y_i, y_{i+1})$$

$$B = \min_{\substack{i=1,\ldots,p-1 \\ (y_i, y_{i+1}) \notin U \\ (y_{i+1}, y_i) \in U}} f(y_{i+1}, y_i) - b(y_{i+1}, y_i)$$

$$C = \min_{\substack{i=1,\ldots,p-1 \\ (y_i, y_{i+1}) \in U \\ (y_{i+1}, y_i) \in U}} c(y_i, y_{i+1}) - f(y_i, y_{i+1}) + f(y_{i+1}, y_i) - b(y_{i+1}, y_i)$$

Pour conclure, il suffit de prouver que f' est un flot b-canalisé de valeur strictement supérieure à $|f|$. D'une part, f' est, comme

on le vérifie de suite, un flot b-canalisé. D'autre part, de l'égalité

$$|f'| = |f| + (f'(y_{p-1},y_p) - f(y_{p-1},y_p))$$

et de la définition de $f'(y_{p-1},y_p)$ on déduit de suite qu'on a

$$|f'| = |f| + m$$

ce qui suffit puisqu'on a, vu la définition de $R_b(f)$, $m > 0$.

La condition est suffisante. Pour le démontrer, considérons l'ensemble A des sommets non accessibles de e dans $R_b(f)$. On a, bien sûr, par hypothèse, $e \notin A$ et $t \in A$ et, par conséquent, $\omega^-(A)$ est une coupe du réseau R. Prouvons en outre qu'on a

$$\forall (u,v) \in U : u \notin A \text{ et } v \in A \Longrightarrow f(u,v) = c(u,v) \qquad (*)$$

$$\forall (u,v) \in U : u \in A \text{ et } v \notin A \Longrightarrow f(u,v) = b(u,v) \qquad (**)$$

En effet, les égalités (*) résultent de suite du fait que si deux sommets $u \notin A$ et v sont tels que $(u,v) \in U$ et $f(u,v) < c(u,v)$ alors (u,v) appartient à $U_b(f)$ et, vu la définition de A, v n'appartient pas à A. De plus, les égalités (**) résultent du fait que si deux sommets u et $v \notin A$ sont tels que $(u,v) \in U$ et $f(u,v) > b(u,v)$ alors (v,u) appartient à $U_b(f)$ et, vu la définition de A, u n'appartient pas à A. Dans ces conditions, le lemme I.3.3. établit les égalités

$$\begin{aligned} f &= f(\bar{A},A) - f(A,\bar{A}) \\ &= \sum_{\substack{(x,y)\in U \\ x\notin A, y\in A}} f(x,y) - \sum_{\substack{(x,y)\in U \\ x\in A, y\notin A}} f(x,y) \\ &= \sum_{\substack{(x,y)\in U \\ x\notin A, y\in A}} c(x,y) - \sum_{\substack{(x,y)\in U \\ x\in A, y\notin A}} b(x,y) \\ &= c(\omega^-(A)) - b(\omega^+(A)) \end{aligned}$$

La thèse résulte alors de l'application du théorème du flot b-canalisé maximum I.3.5.. ///

La démonstration de ce théorème détermine les modifications à apporter à l'algorithme AUGMENTATION. Nous l'écrivons de nouveau sous forme d'une routine à deux paramètres : le flot b-canalisé f supposé de valeur nonmaximale et $(y_1, \ldots, y_p)$ un chemin d'extrémités initiale e et terminale t dans $R_b(f)$.

ALGORITHME AUGMENTATION_ADAPTE(f,$(y_1, \ldots, y_p)$) II.2.10.
Soient f un flot b-canalisé dans R et $(y_1, \ldots, y_p)$ un chemin d'extrémités initiale e et terminale t dans $R_b(f)$.

```
m := +∞;
Pour i variant de 1 à p-1 effectuer
  si (y_i, y_{i+1}) ∈ U et (y_{i+1}, y_i) ∉ U alors
    min := c(y_i, y_{i+1}) - f(y_i, y_{i+1})
  si (y_i, y_{i+1}) ∉ U et (y_{i+1}, y_i) ∈ U alors
    min := f(y_i, y_{i+1}) - b(y_{i+1}, y_i)
  si (y_i, y_{i+1}) ∈ U et (y_{i+1}, y_i) ∈ U alors
    min := c(y_i, y_{i+1}) - f(y_i, y_{i+1}) + f(y_{i+1}, y_i) - b(y_{i+1}, y_i)
  si m > min alors
    m := min;
Pour tout u ∈ U effectuer f'(u) := f(u);
Pour i variant de 1 à p-1 effectuer
  si (y_i, y_{i+1}) ∈ U et (y_{i+1}, y_i) ∉ U alors
    f'(y_i, y_{i+1}) := f(y_i, y_{i+1}) + m
  si (y_i, y_{i+1}) ∉ U et (y_{i+1}, y_i) ∈ U alors
    f'(y_{i+1}, y_i) := f(y_{i+1}, y_i) - m
  si (y_i, y_{i+1}) ∈ U et (y_{i+1}, y_i) ∈ U alors
    f'(y_i, y_{i+1}) := f(y_i, y_{i+1}) + min{m, c(y_i, y_{i+1}) - f(y_i, y_{i+1})}
    f'(y_{i+1}, y_i) := f(y_{i+1}, y_i) - max{0, m - c(y_i, y_{i+1}) + f(y_i, y_{i+1})}
```

La proposition suivante résulte du théorème II.2.9.

PROPOSITION II.2.11. L'algorithme AUGMENTATION_ADAPTE appliqué à un flot b-canalisé dans R et à un chemin d'extrémités initiale e et terminale t dans $R_b(f)$ détermine un flot b-canalisé f' de valeur strictement supérieure à |f|. De plus, f' est à valeurs entières s'il en est ainsi du flot f et de la fonction c.

Démonstration. On prouve cette proposition comme la nécessité de la condition énoncée en II.3.9.. ///

Donnons, enfin, l'algorithme adapté de Ford et Fulkerson capable de résoudre le problème du flot b-canalisé maximum.

ALGORITHME_ADAPTE_DE_FORD_ET_FULKERSON II.2.12.

Déterminer grâce à l'algorithme II.2.1. un flot maximum f^* dans le réseau de transport b-auxiliaire R^*;
Si f^* sature tous les arcs terminaux de R^*
  alors
    Pour tout $u \in U$ effectuer $f_0(u) := f^*(u) + b(u)$
    $k := 0$
    Tant qu'il existe un chemin d'extrémités initiale e et terminale t dans $R_b(f)$ répéter
      choisir un tel chemin, soit $(y_1, \ldots, y_p)$.
      $f_{k+1} :=$ AUGMENTATION_ADAPTE$(f_k, (y_1, \ldots, y_p))$
      $k := k+1$
  sinon
    arrêter : le problème ne possède pas de solution.

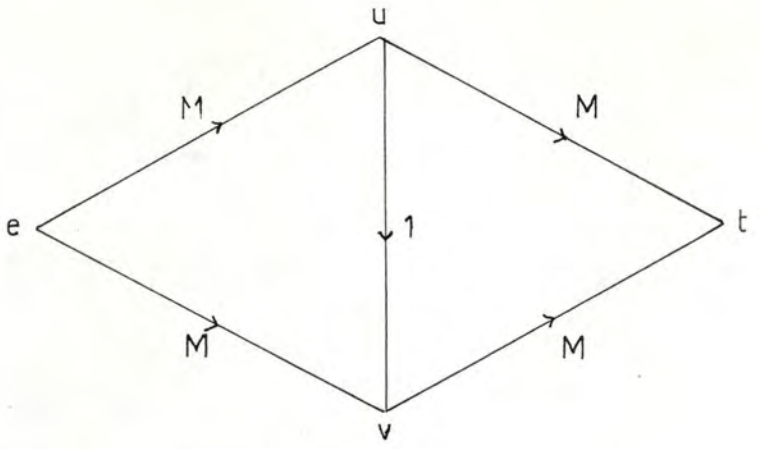
THEOREME II.2.13. Si les fonctions c et b d'un réseau de transport R sont à valeurs entières, alors l'algorithme adapté de Ford et Fulkerson converge et détermine, après exécution, s'il existe un flot b-canalisé dans R de valeur maximum et à valeurs entières.

Démonstration. La démonstration, semblable à celle du théorème II.2.2., repose sur les théorèmes II.2.2., II.2.6., II.2.7., II.2.9. et sur la proposition II.2.11.. ///

REMARQUE II.2.14. On peut bien sûr traiter de la même façon qu'en II.2.3. les réseaux de transport pour lesquels les nombres c(u) et b(u) sont commensurables ou non.

Les algorithmes II.2.1. et II.2.12. n'indiquent d'aucune manière la façon de déterminer les chemins $(y_1, \ldots, y_p)$ joignant les sommets e et t dans R(f) et $R_b(f)$. Plusieurs méthodes ont été développées. Nous les exposons dans les paragraphes qui suivent en donnant pour chacune la complexité des algorithmes qui en découlent.

Montrons dès à présent qu'un choix peu heureux de chemin peut entraîner l'exécution d'un nombre considérable d'itérations. Dans l'exemple suivant,



où M désigne un entier strictement positif et où les valuations des arcs définissent les capacités de ceux-ci, l'exécution de l'algorithme II.2.1. peut entraîner 2M (soit la valeur maximum d'un flot) itérations de la boucle *Tant que* ... si le procédé de recherche de chemins dans R(f) conduit à sélectionner alternativement les chemins (e,u,v,t) et (e,v,u,t).

II.3. La méthode de recherche en profondeur d'abord

Cette méthode est basée sur l'exploration en profondeur d'abord des sommets d'un graphe. De ce fait, les sommets sont visités selon le principe :

- *Ayant déjà visité le sommet* $x_j \in X$, *se déplacer sur un arc* $(x_j, x_k) \in U$ *incident vers l'extérieur à* x_j.
- *Si l'on a déjà visité le sommet* x_k, *revenir en* x_j *et choisir un autre arc incident vers l'extérieur à* x_j. *Sinon, visiter le sommet* x_k *et appliquer le processus ainsi défini de manière récursive à* x_k.
- *Si tous les arcs incidents vers l'extérieur à* x_j *ont été explorés, revenir au sommet* $x_i \in X$ *qui a permis d'atteindre* x_j *et continuer à explorer les arcs incidents vers l'extérieur à* x_i.

L'algorithme ci-dessous en découle. Il permet de résoudre le problème du flot maximum; légèrement modifié, il pourrait, bien sûr, résoudre le problème du flot b-canalisé maximum.

On y tente tout d'abord de déterminer une chaîne augmentante. Dans ce but, une procédure récursive RECHERCHE_CHAINE de spécification

> si le sommet x est visité, alors RECHERCHE_CHAINE(x) recherche dans le sous-graphe de R(f) engendré par l'ensemble des sommets non encore visités uni à {x}, un chemin joignant x à t

a été développée. A l'issue de l'appel RECHERCHE_CHAINE(e), deux cas peuvent se présenter.

Dans le premier, le sommet t a pu être marqué et, par suite, un chaîne f-augmentante α a été découverte. On déduit alors, de même qu'en II.1.3. du flot courant f un flot de valeur strictement supérieure. Toutefois, afin d'améliorer l'efficacité de l'algorithme développé, le nombre m intervenant dans l'algorithme AUGMENTATION II.1.3. est calculé lors de la recherche de la chaîne α. Il reste enfin à essayer d'améliorer le nouveau flot obtenu grâce à une itération supplémentaire.

Dans le deuxième, le sommet t n'a pu être marqué. Il s'ensuit qu'aucune chaîne f-augmentante n'existe. Le flot courant f est dès lors maximum.

Les structures de données suivantes sont utilisées. A chaque sommet x, on associe

- une valeur booléenne, *marqué(x)*, indiquant si le sommet x a été visité,
- un entier, *min(x)*, permettant de calculer m,
- un sommet, *pred(x)*, déterminant le sommet à partir duquel le sommet x a été visité.

L'application *pred* : $X \to X$ détermine, bien entendu, pour tout sommet x marqué un chemin dans R(f) joignant e à x. Le nombre *min(x)* vaut alors le minimum des valuations des arcs de ce chemin, la valuation de l'arc $u = (x,y)$ étant

$c(u) - f(u)$ si $u \in U$ et $\overleftarrow{u} \notin U$

$f(\overleftarrow{u})$ si $u \notin U$ et $\overleftarrow{u} \in U$

$c(u) - f(u) + f(\overleftarrow{u})$ si $u \in U$ et $\overleftarrow{u} \in U$

où $\overleftarrow{u} = (y,x)$. En particulier, $min(t)$ vaut m si t est marqué.

ALGORITHME RECHERCHE_CHAINE(s) II.3.1. Soit s un sommet de X.

```
Pour tout x ∈ X effectuer
  si marqué(t) = non et si marqué(x) = non et si
  { ( (s,x) ∈ U et f(s,x) < c(s,x) ) ou ( (x,s) ∈ U et f(x,s) > 0 ) }
    alors
      marqué(x) := oui;
      pred(x) := s;
      si (s,x) ∈ U et (x,s) ∉ U alors
        min(x) := min{min(s), c(x,s) - f(s,x)};
      si (s,x) ∉ U et (x,s) ∈ U alors
        min(x) := min{min(s), f(x,s)};
      si (s,x) ∈ U et (x,s) ∈ U alors
        min(x) := min{min(s), c(s,x) - f(s,x) + f(x,s)};
      RECHERCHE_CHAINE(x);
Si marqué(t) = non alors
  Pour tout x ∈ X effectuer
    si pred(x) = s alors
      marqué(x) := non;
```

ALGORITHME DE FORD ET FULKERSON II.3.2.

```
Choisir un flot initial f0;
f := f0;
Pour tout x ∈ X\{e} effectuer  marqué(x) := non;
marqué(e) := oui;
min(e) := +∞;
RECHERCHE_CHAINE(e);
Tant que marqué(t) = oui effectuer
  (* amélioration du flot actuel *)
  s := t;
  ps := pred(s);
  répéter
    si (ps,s) ∈ U et (s,ps) ∉ U alors
      f(ps,s) := f(ps,s) + min(t)
    si (ps,s) ∉ U et (s,ps) ∈ U alors
      f(s,ps) := f(s,ps) - min(t)
```

```
si (ps,s) ∈ U et (s,ps) ∈ U alors
  f(ps,s) := f(ps,s) + min{min(t), c(ps,s) - f(ps,s)}
  f(s,ps) := f(s,ps) - max{0, min(t) - c(ps,s) + f(ps,s)}
s := ps
tant que s ≠ e;
(* recherche d'une chaîne f-augmentante *)
Pour tout x ∈ X\{e} effectuer marqué(x) := non;
marqué(e) := oui;
min(e) := +∞;
RECHERCHE_CHAINE(e).
```

Le théorème suivant détermine le nombre maximum d'itérations dans la boucle *Tant que marqué(t) = oui*.

THEOREME II.3.3. Si les capacités des arcs du réseau de transport R sont entières et si M désigne la valeur d'un flot maximum dans R, alors l'algorithme II.3.2. exécuté à partir d'un flot initial à valeurs entières converge après au plus M augmentations de flot.

Démonstration. De fait, d'une part, le théorème II.2.2. affirme, dans ces conditions, que l'algorithme II.3.2. converge et détermine un flot de valeur maximum M. D'autre part, si $(f_0, \ldots, f_N)$ désigne la suite de flot définie par son exécution à partir du flot initial f_0 à valeurs entières, on a, pour tout $k \in \{0, \ldots, N-1\}$,

$$|f_{k+1}| - |f_k| \geqslant 1$$

ce qui suffit, puisque, vu la proposition II.1.4., tout flot f_k $(0 \leqslant k \leqslant N)$ est à valeurs entières. ///

II.4. La méthode de recherche en largeur d'abord

Dans une deuxième approche, la recherche d'une chaîne augmentante s'effectue selon une exploration en largeur d'abord des sommets de R(f).

Rappelons ici que, suivant ce mode d'exploration, les sommets d'un graphe sont visités conformément à l'algorithme

```
Choisir un sommet x;
num(x) := 1;
Tant qu'il existe un sommet numéroté i répéter
    numéroter i+1 tout sommet non encore numéroté
      adjacent à un sommet numéroté i;
    i := i+1.
```

où *num(x)* est un numéro associé à chaque sommet x.

Ainsi, si l'on appelle "scanner un sommet x" l'opération qui consiste à visiter les sommets non encore visités adjacents au sommet x, l'exploration des sommets en largeur d'abord s'effectue selon le principe "premier visité, premier scanné" par opposition au processus d'exploration en profondeur d'abord s'effectuant, quant à lui, selon la règle "dernier visité, premier scanné".

Particularisé à la recherche d'un chemin d'extrémités initiale e et terminale t dans R(f), le mode d'exploration en largeur d'abord donne naissance à l'algorithme II.4.1.. Les structures de données suivantes y sont manipulées : à chaque sommet x, on associe

- un numéro, *num(x)*,
- un sommet, *pred(x)*, déterminant le sommet à partir duquel le sommet x a été visité.

ALGORITHME II.4.1. Soit f un flot dans un réseau de transport R = (X,U,c).

```
Pour tout x∈X \ {e} effectuer  num(x) := 0;
num(e) := 1;
i := 1;
Tant que ( ∃ x∈X : num(x) = i) et (num(t) = 0) répéter
    Pour tout x∈X effectuer
        si num(x) = i alors
            Pour tout y∈X effectuer
                si ( {(x,y)∈U et f(x,y) < c(x,y)}  ou
                     {(y,x)∈U et f(y,x) > 0} ) et  ( (num(t) = 0 ) alors
                        num(y) := i + 1;
                        pred(y) := x;
    i := i + 1.
```

On peut facilement démontrer que les chaînes augmentantes déterminées par l'algorithme II.4.1. sont de longueur minimale. Cette propriété permet de déterminer la complexité de l'algorithme de Ford et Fulkerson II.4.2. adapté à la recherche en largeur d'abord de chemins d' extrémités initiale e et terminale t dans le graphe d' écart R(f). Enonçons le tout d'abord.

ALGORITHME DE FORD ET FULKERSON II.4.2.

```
Choisir un flot initial f_0;
f := f_0;
s := e;
Répéter
  (* amélioration du flot courant f *)
  Tant que s ≠ e effectuer
    ps := pred(s);
    si (ps,s) ∈ U et (s,ps) ∉ U alors
      f(ps,s) := f(ps,s) + min(t);
    si (ps,s) ∉ U et (s,ps) ∈ U alors
      f(s,ps) := f(s,ps) - min(t);
    si (ps,s) ∈ U et (s,ps) ∈ U alors
      f(ps,s) := f(ps,s) + min{min(t), c(ps,s) - f(ps,s)}
      f(s,ps) := f(s,ps) - max{0, min(t) - c(ps,s) + f(ps,s)}
    s := ps;
  (* recherche d'une chaîne f-augmentante *)
  Pour tout x ∈ X\{e} effectuer  num(x) := 0;
  num(e) := 1;
  i := 1;
  min(e) := +∞;
  Répéter
    fin_itérations := vrai;
    Pour tout x ∈ X effectuer
      si num(x) = i alors
        Pour tout y ∈ X effectuer
          si (num(y) = 0)  et  ( {(x,y) ∈ U et f(x,y) < c(x,y)}
          ou  {(y,x) ∈ U et f(y,x) > 0} )  alors
            num(y) := i+1;
            pred(y) := x;
```

```
|  |  |  |  |  |  |  |fin_itérations := faux;
|  |  |  |  |  |  |  |si (x,y)∈U et (y,x)∉U alors
|  |  |  |  |  |  |  | |min(y) := min{min(x), c(x,y) - f(x,y)};
|  |  |  |  |  |  |  |si (x,y)∉U et (y,x)∈U alors
|  |  |  |  |  |  |  | |min(y) := min{min(x), f(y,x)};
|  |  |  |  |  |  |  |si (x,y)∈U et (y,x)∈U alors
|  |  |  |  |  |  |  | |min(y) := min{min(x), c(x,y) - f(x,y) + f(y,x)};
|  |  |  |  |  |  |i := i + 1;
|  |  |  |  |  |tant que (fin_itérations = faux) et (num(t) = 0);
|  |  |  |  |  |s := t;
|  |  |  |  |tant que num(t) ≠ 0.
```

Afin d'obtenir une borne sur le nombre d'augmentations de flot effectuées dans l'algorithme précédent, définissons tout d'abord le concept de goulot et démontrons les lemmes suivants.

DEFINITION II.4.3. Soient f un flot d'un réseau de transport R et ξ une chaîne f-augmentante. Un arc $v \in \xi$ est un <u>goulot de</u> ξ s'il vérifie les égalités

$$ev(v) = \min_{u \in \xi} ev(u)$$

où, pour tout arc $u = (x,y) \in U$, ev(u) est défini par

$$ev(u) = \begin{cases} c(x,y) - f(x,y) & \text{si } (x,y) \in U \text{ et } (y,x) \notin U \\ f(y,x) & \text{si } (x,y) \notin U \text{ et } (y,x) \in U \\ c(x,y) - f(x,y) + f(y,x) & \text{si } (x,y) \in U \text{ et } (y,x) \in U \end{cases}$$

CONVENTION II.4.4. Dans ce paragraphe, sauf mention explicite du contraire,

$$(f_0, \ldots, f_N)$$

désigne la suite de flots déterminée par l'exécution de l'algorithme de Ford et Fulkerson II.2.1. à partir du flot initial f_0.

De plus, nous notons

$$\delta_k(x,y) \quad (k \in \{0, \ldots, N-1\})$$

la distance de x à y dans le réseau d'écart $R(f_k)$, et nous désignons par

$$P_k \quad (k \in \{0, \ldots, N-1\})$$

le chemin d'extrémités initiale e et terminale t dans $R(f_k)$ grâce auquel f_{k+1} est obtenu par augmentation du flot f_k.

LEMME II.4.5. Pour tout $k \in \{0, \ldots, N-1\}$, tout goulot de P_k n'appartient pas au réseau d'écart $R(f_{k+1})$.

Démonstration. Soit (x,y) un goulot de la chaîne f_k-augmentante P_k, $k \in \{0, \ldots, N-1\}$. Deux cas sont à considérer. Dans le premier, l'arc(x,y) appartient à U. Il vient alors, vu l'algorithme AUGMENTATION II.1.3. et la définition II.4.3., si (y,x) n'appartient pas à U,

$$f_{k+1}(x,y) = f_k(x,y) + \min(t)$$

$$= f_k(x,y) + c(x,y) - f_k(x,y) = c(u,v)$$

et si (y,x) appartient à U,

$$f_{k+1}(x,y) = f_k(x,y) + \min\{\min(t), c(x,y) - f_k(x,y)\}$$

$$= f_k(x,y) + \min\{c(x,y) - f_k(x,y) + f_k(y,x), c(x,y) - f_k(x,y)\}$$

$$= c(x,y),$$

$$f_{k+1}(y,x) = f_k(y,x) - \max\{0, \min(t) - c(x,y) + f_k(x,y)\}$$

$$= f_k(y,x) - \max\{0, f_k(y,x)\}$$

$$= 0$$

Dans ce cas, (x,y) ne peut, dès lors, appartenir à $R(f_{k+1})$.

Dans le second, l'arc (x,y) n'appartient pas à U. Comme c'est un goulot, on doit alors nécessairement avoir $(y,x) \in U$. Les égalités

$$\begin{aligned} f_{k+1}(y,x) &= f_k(y,x) - \min(t) \\ &= f_k(y,x) - f_k(y,x) \\ &= 0 \end{aligned}$$

s'ensuivent. Elle prouvent qu'on a $(x,y) \notin R(f_{k+1})$ et, par suite, la thèse. ///

LEMME II.4.6. Tout arc $(x,y) \in R(f_{k+1}) \setminus R(f_k)$ $(k \in \{0, \ldots, N-1\})$ est tel que $(y,x) \in P_k$.

Démonstration. De fait, pour tout arc $(x,y) \in R(f_{k+1}) \setminus R(f_k)$, $k \in \{0, \ldots, N-1\}$, on a nécessairement $f_{k+1}(x,y) \neq f_{k+1}(x,y)$ ou $f_{k+1}(y,x) \neq f_k(y,x)$ et, par conséquent, un des deux arcs (x,y) ou (y,x) doit appartenir à P_k. La thèse résulte alors du fait qu'on ne peut avoir $(x,y) \in P_k$, puisqu'on a, par hypothèse, $(x,y) \notin R(f_k)$. ///

LEMME II.4.7. Pour tout $k \in \{0, \ldots, N-1\}$ et tout $x \in X$, on a $\delta_k(e,x) \leqslant \delta_{k+1}(e,x)$ et $\delta_k(x,t) \leqslant \delta_{k+1}(x,t)$.

Démonstration. Etablissons la première relation, la preuve de la seconde étant similaire. Comme le cas où $\delta_{k+1}(e,x) = +\infty$ est évident, on peut bien sûr supposer avoir $\delta_{k+1}(e,x) = h$, $h \in \mathbb{N}$. Dans ces conditions, il existe un chemin $u_o = e, \ldots, u_h = x$ de longueur h et d'extrémités initiale e et terminale x dans $R(f_{k+1})$ pour lequel on peut établir qu'on a

$$\delta_k(e,u_{i+1}) \leqslant 1 + \delta_k(e,u_i), \quad \forall\, i \in \{0, \ldots, h-1\}.$$

De fait, d'une part, si l'arc (u_i,u_{i+1}), $i \in \{0, \ldots, h-1\}$, appartient à $R(f_k)$ on a $\delta_k(e,u_{i+1}) \leqslant 1 + \delta_k(e,u_i)$. D'autre part, si l'arc (u_i,u_{i+1}), $i \in \{0, \ldots, h-1\}$, n'appartient pas à $R(f_k)$, le lemme précédent affirme que (u_{i+1},u_i) appartient à P_k. Il s'ensuit qu'on a, puisque P_k est un chemin de longueur minimum entre e et t dans $R(f_k)$, $\delta_k(e,u_i) = 1 + \delta_k(e,u_{i+1})$ c'est-à-dire

$$\delta_k(e,u_{i+1}) = -1 + \delta_k(e,u_i) < 1 + \delta_k(e,u_i),$$

ce qui suffit. La thèse résulte alors aussitôt des inégalités

$$\delta_k(e,x) = \delta_k(e,u_h)$$

$$\leqslant \delta_k(e,u_{h-1}) + 1$$

$$\leqslant \delta_k(e,u_o) + h = h = \delta_{k+1}(e,x). \ ///$$

LEMME II.4.8. Pour tout k et $m \in \{0, \ldots, N\}$ tels que $k < m$ et tout goulot (x,y) de P_k et P_m, il existe $\ell \in \{0, \ldots, N\}$ tel que $k < \ell < m$ et $(y,x) \in P_\ell$.

Démonstration. Soient k et m deux entiers de $\{0, \ldots, N\}$ tels que $k < m$ et (x,y) un goulot de P_k et P_m. Prouvons que l'entier

$$\ell = \min\{ t : N \geqslant t > k \text{ et } (x,y) \in R(f_t)\} - 1$$

vérifie la thèse. De fait, de l'appartenance de (x,y) à $R(f_m)$, on déduit de suite que l'ensemble $\{t : N \geqslant t > k \text{ et } (x,y) \in R(f_t)\}$ n'est pas vide et que l'on a $\ell + 1 \leqslant m$. De plus, le lemme II.4.5. affirme que (x,y) n'appartient pas au graphe d'écart $R(f_{k+1})$. Par suite, il vient $\ell + 1 > k + 1$ c'est-à-dire $\ell > k$. La thèse résulte alors du lemme II.4.6. et du fait qu'on a $(x,y) \in R(f_{\ell+1}) \setminus R(f_\ell)$. ///

LEMME II.4.9. Pour toute paire d'entiers $(k,\ell) \in \{0, \ldots, N\}^2$ telle que $k < \ell$, s'il existe un arc $(x,y) \in U$ tel que $(x,y) \in P_k$ et $(y,x) \in P_\ell$, on a $\delta_\ell(e,t) \leqslant \delta_k(e,t) + 2$.

Démonstration. Soient (k,l) un tel couple d'entiers et (x,y) un arc tel que $(x,y) \in P_k$ et $(y,x) \in P_\ell$. Dans ces conditions, la proposition II.4.7. affirme qu'on a

$$\delta_\ell(e,y) \geqslant \delta_k(e,y)$$

et

$$\delta_\ell(x,t) \geqslant \delta_k(x,t) .$$

Par suite, les définitions de P_k et P_ℓ permettent d'écrire

$$\delta_\ell(e,t) = \delta_\ell(e,y) + 1 + \delta_\ell(x,t)$$

$$\geqslant \delta_k(e,y) + 1 + \delta_k(x,t)$$

$$= (1 + \delta_k(e,x)) + 1 + (1 + \delta_k(y,t))$$

$$= 2 + (1 + \delta_k(e,x) + \delta_k(y,t))$$

$$= 2 + \delta_k(e,t),$$

ce qui suffit pour conclure. ///

Le théorème suivant détermine le nombre maximum d'augmentations de flot effectuées dans l'algorithme II.4.2. ainsi que sa complexité.

THEOREME II.4.10. Le nombre maximum d'augmentations de flot effectuées lors d'une exécution de l'algorithme II.4.2. est

$$\frac{n(n-1)(n+2)}{4} .$$

En particulier, la complexité de l'algorithme II.4.2. est $\mathcal{O}(n^6)$ au plus.

Démonstration. Etablissons tout d'abord que le nombre maximum d'occurrences d'un arc comme goulot d'une chaîne f_k - augmentante ($k \in \{0, \ldots, N\}$) est

$$\frac{n+2}{4} .$$

Soit (x,y) un arc. Désignons par $(k_i)_{i=1,\ldots,p}$ la suite d'indices de $\{0, \ldots, N\}$ pour lesquels (x,y) est un goulot du chemin P_{k_i}. Le lemme II.4.8. lui associe une suite $(\ell_i)_{i=1,\ldots,p-1}$ d'indices telle que

$$k_i < \ell_i < k_{i+1} , \quad \forall i \in \{1, \ldots, p-1\}$$

et

$$(y,x) \in P_{\ell_i} , \quad \forall i \in \{1, \ldots, p-1\} .$$

De plus, l'application du lemme II.4.9. à la suite

$$k_1, \ell_1, k_2, \ell_2, \ldots, k_{p-1}, \ell_{p-1}, k_p$$

notée $(m_i)_{i=1,\ldots,2p-1}$, permet d'écrire

$$\delta_{m_{j+1}}(e,t) \geq \delta_{m_j}(e,t) + 2, \quad \forall j \in \{1, \ldots, 2p-2\}$$

et, par suite,

$$\delta_{m_j}(e,t) \geq 2(j-1) + \delta_{m_1}(e,t)$$

$$\geq 2(j-1) + 1 , \quad \forall j \in \{1, \ldots, 2p-1\}.$$

Dans ces conditions, comme tout chemin P_k est de longueur inférieure ou égale à n-1, il vient

$$2(2(p-1))+1 \leqslant \delta_{m_{2(p-1)+1}} \leqslant n-1$$

et, par conséquent,

$$p \leqslant \frac{n+2}{4},$$

ce qui suffit.

Démontrons, à présent, que le nombre maximum d'augmentations de flot effectuées lors de l'exécution de l'algorithme II.4.2. est

$$\frac{n(n-1)(n+2)}{4}.$$

De fait, chaque augmentation de flot détermine un goulot. Il s'ensuit que le nombre d'augmentations de flot effectuées lors de l'exécution de l'algorithme II.4.2. est inférieur au nombre total d'occurrences de goulot. La conclusion résulte alors de ce que ce nombre est lui-même borné, vu ce qui précède, par

$$A_n^2 \frac{n+2}{4} = \frac{n(n-1)(n+2)}{4}.$$

Enfin, la complexité de l'algorithme II.4.2. se déduit de suite de l'assertion précédente et du fait que chaque augmentation de flot nécessite la recherche d'une chaîne augmentante, ce qui s'effectue en II.4.2. par un algorithme de complexité $\mathcal{O}(n^3)$ au plus. //

REMARQUE II.4.10. La démonstration du théorème précédent prouve que l'algorithme de Ford et Fulkerson II.4.2. converge, même s'il est appliqué à un réseau de transport de capacités non entières.

II.5. La méthode d'Edmonds et Karp

Dans leur méthode, J. Edmonds et R. Karp suggérèrent de rechercher parmi l'ensemble des chaînes f - augmentantes, une chaîne fournissant l'augmentation la plus grande possible du flot f. Dans ce but, ils proposèrent l'algorithme II.5.1.. Les structures suivantes y sont utilisées. A chaque sommet $x \in X$, on associe un sommet *pred(x)* déterminant le sommet à partir duquel x a été visité. De plus, deux sous-ensembles *ens_sommets_marqués* et *ens_sommets_non_marqués* constituent, après chaque itération, une partition de X. Ils mémorisent respectivement l'ensemble des sommets déjà visités et l'ensemble des sommets non visités.

ALGORITHME II.5.1. Soit f un flot d'un réseau de transport R.

```
ens_sommets_marqués := {e};
ens_sommets_non_marqués := X\{e};
Répéter
  eval := +∞;
  fin_itérations := oui;
  Pour tout x ∈ ens_sommets_marqués effectuer
    Pour tout y ∈ ens_sommets_non_marqués effectuer
      si { (x,y) ∈ U et f(x,y) < c(x,y) }  ou
         { (y,x) ∈ U et f(y,x) > 0 }  alors
        si (x,y) ∈ U et (y,x) ∉ U alors
          eval' := c(x,y) - f(x,y);
        si (x,y) ∉ U et (y,x) ∈ U alors
          eval' := f(y,x);
        si (x,y) ∈ U et (y,x) ∈ U alors
          eval' := c(x,y) - f(x,y) + f(y,x);
        si eval' < eval alors
          x_retenu := x ;
          y_retenu := y ;
          fin_itérations := non;
          eval := eval';
  Si fin_itérations = non alors
    pred(y_retenu) := x_retenu;
    ens_sommets_marqués := ens_sommets_marques ∪ {y_retenu};
    ens_sommets_non_marqués := ens_sommets_non_marqués \
                                              {y_retenu};
tant que (fin_itérations = non) et (t ∈ ens_sommets_non_marqués).
```

L'idée maîtresse reste, tout comme dans la méthode de recherche en largeur d'abord, d'atteindre de proche en proche, partant de l'entrée e, la sortie t. Toutefois, à chaque itération, l'algorithme ne retient plus tous les sommets non visités adjacents aux sommets visités à l'itération précédente mais un arc qui, parmi les arcs d'origine visitée et d'extrémité terminale non encore retenue, fournirait, s'il était goulot d'une chaîne augmentante, une augmentation de flot maximum.

La proposition suivante caractérise l'algorithme II.5.1..

PROPOSITION II.5.2. Soit f un flot. L'algorithme II.5.1. converge et vérifie, après exécution, l'assertion

" t appartient à *ens_sommets_marqués* si et seulement s'il existe une chaîne f-augmentante " .

De plus, s'il en est ainsi, l'application $pred : X \to X$ détermine une chaîne f-augmentante qui fournit une augmentation maximum du flot f.

Démonstration. Il s'agit d'une simple vérification. ///

L'algorithme d'Edmonds et Karp résulte de l'adaptation de l'algorithme général de Ford et Fulkerson II.2.1. à la recherche d'une chaîne augmentante selon l'algorithme II.5.1.

ALGORITHME D'EDMONDS ET KARP II.5.3.

```
Choisir un flot initial f0;
f := f0;
s := e;
Répéter
  (* amélioration du flot courant f *)
  Tant que s ≠ e effectuer
    ps := pred(s);
    si (ps,s) ∈ U et (s,ps) ∉ U alors
      f(ps,s) := f(ps,s) + min(t);
    si (ps,s) ∉ U et (s,ps) ∈ U alors
      f(s,ps) := f(s,ps) - min(t)
    si (ps,s) ∈ U et (s,ps) ∈ U alors
      f(ps,s) := f(ps,s) + min{min(t), c(ps,s) - f(ps,s)};
      f(s,ps) := f(s,ps) - max{0, min(t) - c(ps,s) + f(ps,s)};
    s := ps;
```

```
(* recherche d'une chaîne f-augmentante *)
ens_sommets_marqués := {e};
ens_sommets_non_marqués := X\{e};
Répéter
  eval := +∞;
  fin_itérations := oui;
  Pour tout x∈ ens_sommets_marqués effectuer
    Pour tout y∈ ens_sommets_non_marqués effectuer
      si { (x,y)∈ U et c(x,y) > f(x,y) }  ou
         { (y,x)∈ U et f(y,x) > 0 }   alors
        si (x,y)∈ U et (y,x)∉ U alors
          eval' := c(x,y) - f(x,y);
        si (x,y)∉ U et (y,x)∈ U alors
          eval' := f(y,x);
        si (x,y)∈ U et (y,x)∈ U alors
          eval' := c(x,y) - f(x,y) + f(y,x);
        si eval' < eval alors
          x_retenu := x;
          y_retenu := y;
          fin_itérations := non;
          eval := eval';
  Si fin_itérations = non alors
    pred(y_retenu) := x_retenu;
    si (x_retenu,y_retenu)∈ U et (y_retenu,x_retenu)∉ U alors
      min(y_retenu) := min{min(x_retenu), c(x_retenu,y_retenu)
                                  -f(x_retenu,y_retenu)} ;
    si (x_retenu,y_retenu)∉ U et (y_retenu,x_retenu)∈ U alors
      min(y_retenu) := min{min(x_retenu),
                                  f(y_retenu,x_retenu)} ;
    si (x_retenu,y_retenu)∈ U et (y_retenu,x_retenu)∈ U alors
      min(y_retenu) := min{min(x_retenu), c(x_retenu,y_retenu)
                                  -f(x_retenu,y_retenu)
                                  +f(y_retenu,x_retenu)} ;
    ens_sommets_marqués := ens_sommets_marqués ∪ {y_retenu};
    ens_sommets_non_marqués := ens_sommets_non_marqués \
                                  {y_retenu} ;
  tant que (fin_itérations = non) et (t∉ ens_sommets_marqués);
  s := t;
tant que (t∈ ens_sommets_marqués).
```

Le théorème suivant précise le nombre maximum d'augmentations de flot effectuées dans l'algorithme d'Edmonds et Karp II.5.3..

THEOREME II.5.4. Soit M un entier positif tel que, pour toute partition des sommets de X en deux sous-ensembles S et $\bar{S}$, le nombre d'arcs ayant une extrémité dans S et une autre dans $\bar{S}$ est plus petit ou égal à M. Soit en outre f* un flot de valeur maximum. L'algorithme d'Edmonds et Karp appliqué à un réseau R de capacités entières converge et ce, après au plus

$$1 + \log_{M/(M-1)} |f^*|$$

augmentations de flot.

Démonstration. Comme le théorème II.2.2. affirme que l'algorithme d'Edmonds et Karp appliqué à un réseau de capacités entières converge, il suffit de prouver que le nombre maximum d'augmentations de flot effectuées lors de toute exécution de cet algorithme à un tel réseau est

$$1 + \log_{M/(M-1)} |f^*| .$$

Soit $(f_0, \ldots, f_N)$ la suite de flots déterminée lors d'une telle exécution. Fixons $k \in \{0, \ldots, N-1\}$, posons $\varepsilon_k = |f_{k+1}| - |f_k|$ et considérons l'ensemble

$$S = \{e\} \cup \{x \in X : \text{il existe un chemin de } e \text{ à } x \text{ dans } R(f_k) \text{ constitué d'arcs } (x,y) \text{ tels que } e(x,y) > \varepsilon_k\}$$

où, pour tout arc $(x,y) \in R(f)$, $e(x,y)$ vaut

$$e(x,y) = \begin{cases} c(x,y) - f(x,y) & \text{si } (x,y) \in U \text{ et } (y,x) \notin U \\ f(y,x) & \text{si } (x,y) \notin U \text{ et } (y,x) \in U \\ c(x,y) - f(x,y) + f(y,x) & \text{si } (x,y) \in U \text{ et } (y,x) \in U. \end{cases}$$

On a, bien sûr, $t \in \bar{S}$ et $e \in S$. De plus, tout arc $(x,y) \in R(f_k)$ tel que $x \in S$ et $y \in \bar{S}$ vérifie l'inégalité $e(x,y) \leqslant \varepsilon_k$. Dans ces conditions, il vient, comme on le vérifie de suite,

$$c(S,\bar{S}) - (f_k(S,\bar{S}) - f_k(\bar{S},S))$$

$$\leqslant \varepsilon_k \, | \{(x,y) : x \in S, \, y \in \bar{S}, \, (x,y) \in U \text{ ou } (y,x) \in U\} |$$

$$\leqslant \varepsilon_k \, M.$$

Par conséquent, comme on a, vu le lemme I.3.3.,

$$|f^*| \leqslant c(S,\bar{S}),$$

$$|f_k| = f_k(S,\bar{S}) - f_k(\bar{S},S)$$

on obtient,

$$|f^*| - |f_k| \leqslant \varepsilon_k M = (|f_{k+1}| - |f_k|)\, M .$$

Il s'ensuit,

$$|f^*| - |f_{k+1}| \leqslant (|f^*| - |f_k|)(1 - M^{-1})$$

et, dès lors, en raisonnant par induction

$$|f^*| - |f_k| \leqslant |f^*|\,(1 - M^{-1})^k .$$

Par ailleurs, comme les capacités du réseau considéré sont entières, chaque flot f_k, $k \in \{0, \ldots, N\}$, est à valeurs entières. Ceci signifie qu'on a, pour tout flot f_k, $k \in \{0, \ldots, N-1\}$,

$$|f^*| - |f_k| \geqslant 1$$

et, par suite, vu ce qui précède, l'inégalité

$$|f^*|\,(1 - M^{-1})^k \geqslant 1$$

c'est-à-dire

$$k \leqslant \log_{M/(M-1)} |f^*| .$$

En particulier, il vient

$$N-1 \leqslant \log_{M/(M-1)} |f^*|,$$

d'où la conclusion. ///

Afin d'évaluer les performances relatives des algorithmes de Ford et Fulkerson II.4.2. et d'Edmonds et Karp II.5.3., tentons de comparer les bornes fournies par les théorèmes II.4.10. et II.5.4.. Dans ce but, désignons par $\bar{c}$ la capacité moyenne d'un arc de U. On a bien sûr, en utilisant des notations connues

$$|f^*| \leqslant \bar{c}\, n^2 .$$

De plus, comme on le vérifie de suite, le nombre M satisfait l'inégalité

$$M \leqslant \frac{n^2}{2} .$$

Dans ces conditions, il vient

$$\log_{M/(M-1)} |f^*| \leqslant \log_{(n^2/n^2-2)} (n^2 \bar{c})$$

$$= \frac{\ln n^2 \bar{c}}{\ln \left(\frac{n^2}{n^2-2}\right)} .$$

Pour conclure, il reste à évaluer le dénominateur. Or, on peut l'estimer en écrivant

$$\ln \left(\frac{n^2}{n^2-2}\right) = \ln \left(1 + \frac{2}{n^2-2}\right)$$

$$\geqslant \ln \left(1 + \frac{2}{n^2}\right)$$

$$\geqslant \frac{2}{n^2} - \frac{1}{2} \left(\frac{2}{n^2}\right)^2 ,$$

la dernière inégalité se déduisant de suite du développement en série du logarithme népérien. Par conséquent, le nombre maximum d'augmentations de flot

$$1 + \log_{M/(M-1)} |f^*|$$

peut être approché par

$$1 + \frac{2 \ln n + \ln \bar{c}}{2(1/n^2 - 1/n^4)} = 1 + \frac{n^4}{2n^2 - 2} (2 \ln n + \ln \bar{c})$$

$$= n^2 \ln n + \frac{1}{2} n^2 \ln \bar{c} + \mathcal{O}(n^2 \ln n + n^2 \ln \bar{c})$$

où $\mathcal{O}(n^2 \ln n + n^2 \ln \bar{c})$ désigne un terme qui, divisé par $n^2 \ln n + n^2 \ln \bar{c}$, converge vers 0 quand n ou $\bar{c}$ tend vers l'infini.

En conclusion, bien que la borne précédente dépende des capacités et requiert leur intégrité, elle est supérieure à la borne du théorème II.4.10. dans, approximativement, l'interval $0 \leqslant \bar{c} \leqslant e^{n/4}$.

II.6. L'algorithme d'Edmonds et Karp de résolution du problème du flot maximum de coût minimum

Nous présentons ici un algorithme de résolution du problème du flot maximum de coût minimum dû à J. Edmonds et R. Karp. Comme il s'agit d'un algorithme applicable uniquement aux réseaux de transport asymétriques, nous formulons, tout au long de ce paragraphe, l'hypothèse II.6.1..

HYPOTHESE II.6.1. Pour tout réseau de transport $R = (X,U,c)$, on a

$$\forall x,y \in X : (x,y) \in U \Longrightarrow (y,x) \notin U.$$

Remarquons qu'on peut toujours s'y ramener en introduisant lorsque deux sommets x,y d'un réseau de transport R sont tels que $(x,y) \in U$ et $(y,x) \in U$ un sommet intermédiaire x* et en remplaçant l'arc (x,y) par deux arcs (x,x*) et (x*,y). La résolution du problème du flot maximum de coût minimum dans R est, comme on le vérifie de suite, équivalente à la résolution du même problème dans le nouveau réseau.

La convention et les définitions suivantes seront utiles dans la suite.

CONVENTION II.6.2. Soit $d : U \to \mathbb{R}^+$ une fonction de coût. Dans ce paragraphe, nous notons, sauf mention explicite du contraire,

$$\Delta_d(x,y)$$

le nombre défini pour tout couple de sommets $x,y \in X$ tels que $(x,y) \in U$ ou $(y,x) \in U$ par l'égalité

$$\Delta_d(x,y) = \begin{cases} d(x,y) & \text{si } (x,y) \in U \\ -d(x,y) & \text{si } (y,x) \in U \end{cases} .$$

De plus, nous appelons,

Δ_d - coût d'un chemin ξ

la somme

$$\sum_{u \in \xi} d(u) .$$

Enfin, la notation $\Delta_d(x,y)$ et l'appellation Δ_d - coût sous-entendent que d est une fonction de U dans $\mathbb{R}^+$.

DEFINITION II.6.3. Une fonction d'étiquetage est une fonction de l'ensemble des sommets d'un réseau de transport sur $\mathbb{R}$.

DEFINITION II.6.4. Soit $d : U \to \mathbb{R}^+$ une fonction de coût. Un flot est d - extrême s'il est de d-coût minimum parmi les flots de même valeur.

DEFINITION II.6.5. Soit $d : U \to \mathbb{R}^+$ une fonction de coût. Un flot f et une fonction d'étiquetage π sont d-compatibles si on a, pour tout $(x,y) \in U$,

$$\pi(x) - \pi(y) + d(x,y) > 0 \Longrightarrow f(x,y) = 0$$

$$\pi(x) - \pi(y) + d(x,y) < 0 \Longrightarrow f(x,y) = c(x,y).$$

CONVENTION II.6.6. Dans la suite de ce travail, sauf mention explicite du contraire, les appellations

d - extrême et d - compatible

sous-entendent que d est une fonction définie de l'ensemble des arcs U du réseau considéré dans $\mathbb{R}^+$.

Le théorème suivant est fondamental. Il repose sur la théorie de la programmation linéaire et tout particulièrement sur les théorèmes de dualité. Afin d'alléger l'exposé, nous l'admettons.

THEOREME II.6.7. Soient f un flot et $d : U \to \mathbb{R}^+$ une fonction de coût. Les assertions suivantes sont équivalentes.

a) Le flot f est d - extrême.

b) Tout circuit du réseau d'écart R(f) est de Δ_d - coût non négatif.

c) Il existe une fonction d'étiquetage π telle que, pour tout arc $(x,y) \in R(f)$, on ait

$$\pi(x) + \Delta_d(x,y) - \pi(y) \geqslant 0.$$

d) Il existe une fonction d'étiquetage π d - compatible avec le flot f.

Démonstration. Résultat admis (cfr. [13]). ///

La proposition suivante est immédiate.

PROPOSITION II.6.8. Soient f un flot et π une fonction d'étiquetage. Soit, en outre, pour tout arc $(x,y) \in R(f)$, le poids

$$\bar{\Delta}_d(x,y) = \pi(x) + \Delta_d(x,y) - \pi(y).$$

a) Pour tout circuit C de R(f), on a

$$\sum_{u \in C} \bar{\Delta}_d(u) = \sum_{u \in C} \Delta_d(u).$$

b) Pour tout chemin P de R(f) d'extrémités initiale e et terminale t, on a

$$\sum_{u \in P} \bar{\Delta}_d(u) = \sum_{u \in P} \Delta_d(u) + \pi(x) - \pi(y).$$

c) Tout circuit de R(f) est de poids positif (resp. négatif) si et seulement s'il est de Δ_d - coût positif (resp. négatif).

d) Tout chemin de R(f) est de poids minimum si et seulement s'il est de Δ_d - coût minimum.

Démonstration. Les assertions a) et b) sont évidentes et les propositions c) et d) s'en déduisent aussitôt. ///

Le théorème ci-dessous constitue aussi un résultat de base.

THEOREME II.6.9. Tout flot obtenu par augmentation d'un flot f d - extrême le long d'une chaîne f-augmentante de Δ_d - coût minimum est d - extrême.

Démonstration. Soient f un flot d-extrême et P un chemin, dans R(f), d'extrémités initiale e et terminale t, de Δ_d - coût minimum.

Pour conclure, il suffit, bien sûr, vu le théorème II.6.7., d'établir l'existence d'une fonction d'étiquetage π' compatible avec le flot f' obtenu par augmentation de f le long de P.

Dans ce but, associons tout d'abord à f, grâce au théorème II.6.7., une fonction d'étiquetage π telle que, pour tout $x,y \in X$,

$$(x,y) \in R(f) \Longrightarrow \bar{\Delta}_d(x,y) = \pi(x) + \Delta_d(x,y) - \pi(y) \geqslant 0.$$

Désignons, en outre, pour tout $x \in X$, par $\delta(x)$, le poids (pour les poids $\bar{\Delta}_d$) d'un chemin d'origine e et d'extrémité terminale x de poids (pour ces mêmes poids) minimum et par I l'ensemble des sommets inaccessibles à partir de e dans R(f). Modifions, enfin, le graphe $G = (X,U)$ en un graphe G' obtenu en introduisant pour tout sommet $i \in I$

un arc (t,i) de poids $\bar{\Delta}_d(t,i)$

$$\bar{\Delta}_d(t,i) = 3 \sum_{u \in U(f)} |\Delta_d(u)|.$$

Dans ces conditions, la fonction d'étiquetage π' définie par les égalités

$$\pi'(x) = \pi(x) + \delta'(x), \quad \forall x \in X$$

où $\delta'(x)$ désigne, pour tout $x \in X$, le poids d'un chemin de poids minimum dans le graphe G', est compatible avec f'.

D'une part, $\pi'(x)$ est défini pour tout $x \in X$. De fait, tout sommet x de G' est accessible de e dans G' et, f étant d-extrême, G' ne contient pas de circuit absorbant.

D'autre part, pour tout arc $(x,y) \in R(f')$, on a

$$\pi'(x) + \Delta_d(x,y) - \pi'(y) \geqslant 0.$$

En effet, comme seuls les flux des arcs d'extrémités appartenant au chemin P ont été modifiés, les arcs (t,i) n'appartenant pas à $R(f)$ n'appartiennent pas non plus à $R(f')$. On a donc pour tout arc $(x,y) \in R(f)$ tel qu'un des deux arcs (x,y) ou (y,x) aumoins appartienne à $R(f')$,

$$\bar{\Delta}_d(x,y) = \pi(x) + \Delta_d(x,y) - \pi(y).$$

Deux cas se présentent. Dans le premier, on a $(x,y) \in R(f') \cap R(f)$. Par suite, les poids $\delta'(x)$ et $\delta'(y)$ vérifient l'inégalité

$$\delta'(y) \leqslant \delta'(x) + \bar{\Delta}_d(x,y)$$

c'est-à-dire l'inégalité

$$\delta'(y) + \pi(y) \leqslant \delta'(x) + \pi(x) + \Delta_d(x,y),$$

ce qui suffit. Dans le second, l'arc $(x,y) \in R(f')$ n'appartient pas à $R(f)$. Du lemme II.4.6., on déduit alors l'appartenance de (y,x) à P et par conséquent l'égalité

$$\delta'(x) = \delta'(y) + \bar{\Delta}_d(y,x)$$

c'est-à-dire l'égalité

$$\delta'(x) + \pi(x) = \delta'(y) + \pi(y) + \Delta_d(y,x),$$

ce qui suffit. ///

Montrons à présent comment on peut mener à bien la recherche d'un chemin de poids minimum dans un graphe. L'algorithme de Moore-Dijkstra résout ce problème. Il suppose le graphe sans circuit de poids strictement négatif et l'ensemble des arcs tel qu'il existe toujours un arc d'un sommet donné vers un autre (ce qui est toujours vérifié grâce à l'ajoute éventuelle d'arcs fictifs de poids infini).

ALGORITHME DE MOORE-DIJKSTRA II.6.10.

```
D := {e};
pi(e) := 0;
Pour tout x ∈ X\{e} effectuer
  |pi(x) := poids(e,x);
Tant que D ≠ X répéter
  |Choisir y ∈ X\D tel que pi(y) = min pi(x);
  |                                x∈X\D
  |D := D∪{y};
  |Pour tout x ∈ X\D effectuer
  | |pi(x) := min{pi(x),pi(y) + poids(y,x)}
```

Le vecteur *pi* délivre, après exécution de cet algorithme, les poids des chemins de poids minimum joignant le sommet e à chaque sommet du réseau. L'égalité pi(x) = +∞ indique qu'il n'existe pas de chemin joignant le sommet e au sommet x. En particulier, l'égalité pi(t) = +∞ implique l'inexistence de chemin entre e et t.

On vérifie de suite que la complexité de l'algorithme II.6.10. est $\mathcal{O}(n^2)$. Ceci est dû à la représentation sous forme de matrice des poids des arcs du graphe ainsi qu'à l'introduction d'arcs fictifs de poids infini. On peut aisément le transformer en un algorithme de complexité $\mathcal{O}(nm)$ répondant aux mêmes spécifications en utilisant les structures de données suivantes. A tout élément $x \in X$ est associé l'ensemble *succ(x)* des successeurs de x. De plus, *elmin* représente à la fin de chaque itération l'élément $y \in X \setminus D$ tel que

$$pi(y) = \min_{z \in X \setminus D} pi(z) .$$

L'algorithme devient

ALGORITHME II.6.11.

```
S := φ;
pi(e) := 0;
elmin := e;
etiq(e) := 0;
Pour tout x∈X\{e} effectuer pi(x) := +∞;
Tant que succ(elmin) ∩ S̄ ≠ φ répéter
  | S := S ∪ {elmin};
  | elaux := elmin;
  | pmin := +∞;
  | Pour tout x ∈ succ(elaux) ∩ S̄ effectuer
  |   | si pi(x) > pi(elaux) + poids(elaux,x) alors
  |   |   | pi(x) := pi(elaux) + poids (elaux,x);
  |   |   | etiq(x) := elaux
  |   | si pmin > pi(x) alors
  |   |   | elmin := x;
  |   |   | pmin := pi(x).
```

De nouveau, après exécution, l'égalité pi(y) = +∞ signifie qu'il n'existe pas de chemin joignant e à y. Par ailleurs, le tableau *etiq* permet de reconstituer, pour tout sommet $y \in \bar{S}$ le chemin de poids minimum joignant e à y.

Donnons pour terminer l'algorithme d'Edmonds et Karp résolvant le problème du flot maximum de coût minimum.

ALGORITHME D'EDMONDS ET KARP II.6.12.

```
f_0 := flot nul;
π_0 := fonction d'étiquetage nulle;
k := 0;
Tant qu'il existe une chaîne f_k - augmentante effectuer
  (* augmentation du flot courant *)
  | Rechercher dans R(f_k) un chemin d'origine e et d'extrémité
  |   terminale t de poids minimum pour les poids
  |   Δ_k(x,y) = π_k(x) + Δ_d(x,y) - π_k(y).
  |   S'il en existe plusieurs, choisir un qui possède le
  |   moins d'arcs. Soit P_k ce chemin de poids σ_k;
  | f_{k+1} := AUGMENTATION(f_k, P_k);
```

(* création d'une nouvelle fonction d'étiquetage *)

$\sigma_k(t) := \sigma_k$;

$\sigma_k(e) := 0$;

Pour tout $x \in X \setminus \{e,t\}$ effectuer

  $\sigma_k(x) :=$ poids d'un chemin d'extrémités initiale e et terminale t dans $R(\ell_k)$ de poids minimum pour les poids Δ_k ($= +\infty$ s'il n'existe pas de chemin dans $R(\ell_k)$ joignant e à t)

Pour tout $x \in X \setminus \{e,t\}$ effectuer

  si $\sigma_k(x) = +\infty$ alors

    $\sigma_k(x) := \sigma_k(t) + 3 \sum_{u \in U(\ell_k)} |\Delta_d(u)|$

Pour tout x X effectuer

  $\pi_{k+1}(x) := \pi_k(x) + \sigma_k(x)$;

$k := k + 1$.

Le théorème suivant le caractérise.

THEOREME II.6.13. Appliqué à un réseau de transport à capacités entières, l'algorithme II.6.12. converge et fournit après exécution un flot maximum de d - coût minimum. En outre, si f* désigne un flot de valeur maximum, au plus $|f^*|$ augmentations de flot sont nécessaires.

De plus, si la fonction de coût d est à valeurs entières et si l'entier D les majore, l'algorithme II.6.12. appliqué au même réseau converge après au plus

$$\frac{1}{4}(n-1)^2 n(n+2)D + 1$$

augmentations de flots.

Démonstration. De fait, le lemme I.3.3. et le théorème I.3.4. impliquent de suite que l'exécution de l'algorithme II.6.12. appliqué à un réseau de transport à capacités entières se termine et ce, après au plus $|f^*|$ augmentations de flot. En outre, le flot déterminé est de valeur maximum. Etablissons qu'il est de coût minimum. Vu le théorème II.6.8., il suffit bien sûr de prouver qu'il est d-compatible

avec une fonction d'étiquetage. Dans ce but, posons $(f_o, \ldots, f_N)$ et $(\pi_o, \ldots, \pi_N)$ respectivement les suites de flots et de fonctions d'étiquetage définies par une telle exécution et prouvons que, pour tout $k \leqslant N$, le flot f_k et la fonction d'étiquetage π_k sont compatibles. Procédons par induction. D'une part, le flot f_o et la fonction d'étiquetage π_o sont trivialement d-compatibles. D'autre part, la démonstration du théorème II.6.9. établit de suite que, pour tout $k > 1$, la d-compatibilité de π_{k-1} et f_{k-1} implique celle de π_k et f_k, ce qui suffit.

Pour conclure, démontrons que si la fonction de coût d est à valeurs entières, l'exécution s'arrête après au plus

$$\frac{1}{4}(n-1)^2 n(n+2) D + 1$$

augmentations de flot. Dans ce but, prouvons tout d'abord que, pour tout $k \in \{1, \ldots, N\}$, $\pi_k(t)$ est le poids d'un chemin de poids minimum entre e et t dans $R(f_{k-1})$ pour les poids Δ_d. De fait, vu les définitions de π_k et de σ_k, on a, pour tout $k \in \{0, \ldots, N-1\}$,

$$\begin{aligned} \sigma_k(t) &= \pi_k(e) - \pi_k(t) + \sum_{u \in P_k} \Delta_d(u) \\ &= -\pi_k(t) + \sum_{u \in P_k} \Delta_d(u), \end{aligned}$$

où P_k est un chemin entre e et t dans $R(f_k)$ de poids minimum pour les poids Δ_k et, par suite,

$$\begin{aligned} \pi_{k+1}(t) &= \pi_k(t) + \sigma_k(t) \\ &= \sum_{u \in P_k} \Delta_d(u). \end{aligned}$$

La proposition II.6.8. permet alors de conclure.

Remarquons, en outre, qu'on a, pour tout $k \in \{0, \ldots, N-1\}$, $\pi_{k+1}(t) \geqslant \pi_k(t)$.

Etablissons à présent la borne ci-dessus. Le calcul entier peut être regardé comme une suite de phases, chacune correspondant à une période pendant laquelle $\pi_k(t)$ est constant. Or, dans une telle phase, les augmentations de flot s'effectuent le long de chemins de longueur minimum appartenant au sous-réseau R' du réseau R constitué

par l'ensemble des arcs des chaînes augmentantes intervenant dans la phase. Le théorème II.4.10. est dès lors applicable. Il affirme qu'il existe au plus, dans chaque phase,

$$\frac{1}{4}(n-1)\,n\,(n+2)$$

augmentations de flot. En outre, excepté pour $k = 0$, $\pi_k(t)$ représente le poids, pour les poids Δ_d, d'un chemin de e à t dans $R(f_{k-1})$ et est, par conséquent, un entier compris entre 1 et $(n-1)D$. Il y a donc au plus $(n-1)D$ phases auxquelles il convient d'ajouter la première augmentation de flot. En conclusion, le nombre total d'augmentations de flot est borné par

$$\frac{1}{4}(n-1)^2\,n\,(n+2)\,D + 1.$$

D'où la conclusion. ///

REMARQUE II.6.14. Comme on le vérifie de suite, l'algorithme II.6.12., modifié en prenant pour flot f_o un flot non nécessairement nul mais à valeurs entières et pour fonction π_o une fonction d'étiquetage d-compatible, appliqué à un réseau de transport à capacités entières converge et fournit après exécution un flot maximum de d-coût minimum.

Bien qu'il soit rassurant de savoir que l'algorithme II.6.12. converge, les bornes sur le nombre d'augmentations de flot ne sont guère performantes. La méthode de graduation développée dans les deux paragraphes suivants constitue une variante de cet algorithme pour laquelle les bornes dépendent de façon logarithmique plutôt que linéaire des capacités.

Nous l'exposons tout d'abord dans le cadre plus simple de la résolution du problème de transfert de Hitchcock. Ensuite, nous montrons comment on peut la modifier afin de solutionner le problème du flot maximum de coût minimum.

II.7. La méthode de graduation comme méthode de résolution du problème de transfert de Hitchcock

Dans la méthode de graduation, la résolution du problème de transfert de Hitchcock résulte de la résolution d'une suite de problèmes de transfert de Hitchcock où les coûts sont les mêmes que les coûts du problème donné mais où les capacités approximent progressivement celles du problème initial. L'efficacité d'une telle méthode se base sur les caractéristiques suivantes:

1) les capacités ainsi que les augmentations de flot se mesurent plus grossièrement que dans le problème initial,

2) la solution finale de chaque problème intermédiaire détermine un bon flot pour le problème intermédiaire suivant.

Afin de l'expliciter, établissons tout d'abord les quelques résultats suivants.

CONVENTION II.7.1. Dans tout ce paragraphe, étant donné un réseau de Hitchcock $(\{e,e_1, \ldots ,e_m,t_1, \ldots ,t_n,t\}),U,c)$, nous posons, pour toute fonction α définie sur U

$$\alpha(e,e_i) = \alpha_{oi} \ , \quad \forall i \in \{1, \ldots ,m\}$$

$$\alpha(t_j,t) = \alpha_{jo} \ , \quad \forall j \in \{1, \ldots ,n\}$$

$$\alpha(e_i,t_j) = \alpha_{ij}, \quad \forall i \in \{1, \ldots ,m\}, \ j \in \{1, \ldots ,n\} \ .$$

THEOREME II.7.2. Un flot f d'un réseau de Hitchcock est d-extrême si et seulement s'il existe des réels $u_o, \ldots ,u_m$ et $v_o, \ldots ,v_n$ tels que

$$u_i - v_j + d_{ij} \geqslant 0 \ , \quad \forall i \in \{1, \ldots ,m\}, \ j \in \{1, \ldots ,n\} \ ,$$

$$u_i - v_j + d_{ij} > 0 \Longrightarrow f_{ij} = 0 \ , \quad \forall i \in \{1, \ldots ,m\}, \ j \in \{1, \ldots ,n\} ,$$

$$u_o > u_i \Longrightarrow f_{oi} = 0 \ , \quad \forall i \in \{1, \ldots ,m\}$$

$$u_i > u_o \Longrightarrow f_{oi} = a_i, \quad \forall i \in \{1, \ldots ,m\}$$

$$v_j > v_o \Longrightarrow f_{jo} = 0 , \quad \forall j \in \{1, \ldots ,n\},$$

$$v_o > v_j \Longrightarrow f_{jo} = b_j, \quad \forall j \in \{1, \ldots ,n\}.$$

Démonstration. La condition est nécessaire. De fait, à tout flot f d-extrême, le théorème II.6.9. associe une fonction d'étiquetage π telle que, pour tout arc $(x,y) \in U$,

$$\pi(x) - \pi(y) + d(x,y) > 0 \Longrightarrow f(x,y) = 0$$

$$\pi(x) - \pi(y) + d(x,y) < 0 \Longrightarrow f(x,y) = c(x,y).$$

Pour conclure, il suffit alors de poser

$$u_o = \pi(e),$$

$$v_o = \pi(t),$$

$$u_i = \pi(e_i), \quad \forall i \in \{1, \ldots ,m\},$$

$$v_j = \pi(t_j), \quad \forall j \in \{1, \ldots ,n\}.$$

La condition est suffisante. En effet, la fonction d'étiquetage définie par les égalités précédentes est d-compatible avec le flot f. Par suite, vu le théorème II.6.7., f constitue un flot d-extrême. ///

Le théorème précédent suggère la notion de flot pseudo-extrême.

DEFINTION II.7.3. Soit $d : U \to \mathbb{R}^+$ une fonction de coût. Un flot f d'un réseau de Hitchcock est d-pseudo-extrême s'il existe des réels $u_1, \ldots ,u_m$ et $v_1, \ldots ,v_n$ tels que

$$u_i - v_j + d_{ij} \geqslant 0, \quad \forall i \in \{1, \ldots ,m\}, j \in \{1, \ldots ,n\},$$

$$u_i - v_j + d_{ij} > 0 \Longrightarrow f_{ij} = 0, \quad \forall i \in \{1, \ldots ,m\}, j \in \{1, \ldots ,n\}.$$

CONVENTION II.7.4. Dans la suite de ce travail, l'appellation

d-pseudo-extrême

sous-entend, sauf mention explicite du contraire, que d est une fonction définie de l'ensemble des arcs U du réseau considéré dans $\mathbb{R}^+$.

PROPOSITION II.7.5. Tout flot maximum d-pseudo-extrême est d-extrême.

Démonstration. De fait, si f désigne un flot maximum d-pseudo-extrême, il existe des réels $u_1, \ldots, u_m$ et $v_1, \ldots, v_n$ tels que, pour tout $i \in \{1, \ldots, m\}$ et $j \in \{1, \ldots, n\}$,

$$u_i - v_j + d_{ij} \geqslant 0$$

et

$$u_i - v_j + d_{ij} > 0 \Longrightarrow f_{ij} = 0 .$$

Posons, en outre,

$$u_o = \min_{i=1,\ldots,m} u_i$$

et

$$v_o = \min_{j=1,\ldots,n} v_j .$$

La thèse résulte de suite de l'application du théorème II.7.2. aux réels $u_o, u_1, \ldots, u_m$ et $v_o, v_1, \ldots, v_n$ et du fait qu'on a, puisque f est de valeur maximum, $f_{oi} = a_i$, pour tout $i \in \{1, \ldots, m\}$ et $f_{jo} = b_j$, pour tout $j \in \{1, \ldots, n\}$. ///

Ce dernier résultat fournit un algorithme alternatif à l'algorithme d'Edmonds et Karp II.6.12. Il consiste essentiellement à construire une suite $(f_o, \pi_o), \ldots, (f_N, \pi_N)$ de paires de flot et de fonction d'étiquetage telle qu'on ait, pour tout $k \in \{0, \ldots, N\}$, tout $i \in \{1, \ldots, m\}$ et tout $j \in \{1, \ldots, n\}$,

$$\pi_k(e_i) - \pi_k(t_j) + d_{ij} \geqslant 0$$

$$\pi_k(e_i) - \pi_k(t_j) + d_{ij} > 0 \Longrightarrow f_{ij} = 0.$$

Par suite, tout flot f_k ($k \in \{0, \ldots, N\}$) est pseudo-extrême.

ALGORITHME II.7.6.

f_0 := flot nul;
π_0 := fonction d'étiquetage nulle;
$k := 0$;
Tant qu'il existe une chaîne f_k-augmentante effectuer
(* augmentation du flot courant *)
Rechercher dans $R(f_k)$ un chemin d'origine e et d'extrémité terminale t de poids minimum pour les poids Δ_k définis, pour tout arc $(x,y) \in R(f_k)$ par l'égalité

$$\Delta_k(x,y) = \begin{cases} 0 \text{ si } (x,y) \in \{(e,e_i) : 1 \leqslant i \leqslant m\} \cup \{(t_j,t) : 1 \leqslant j \leqslant m\} \\ \pi_k(x) - \pi_k(y) + \Delta_d(x,y) \text{ sinon} \end{cases}.$$

S'il en existe plusieurs, choisir un qui possède le moins d'arcs. Soit P_k cette chaîne de poids σ_k;
f_{k+1} := AUGMENTATION(f_k, P_k);
(* création d'une nouvelle fonction d'étiquetage *)
$\sigma_k(t) := \sigma_k$;
$\sigma_k(e) := 0$;
Pour tout $x \in X \setminus \{e,t\}$ effectuer
$\sigma_k(x)$:= poids d'un chemin d'extrémité initiale e et terminale x dans $R(f_k)$ de poids minimum pour les poids Δ_k, $+\infty$ s'il n'existe pas de chemin dans $R(f_k)$ joignant e à t;
Pour tout $x \in X \setminus \{e,t\}$ effectuer
si $\sigma_k(x) = +\infty$ alors
$\sigma_k(x) := \sigma_k(t) + 3 \sum_{u \in U(f_k)} |\Delta_d(u)|$;
Pour tout x X effectuer
$\pi_{k+1}(x) = \pi_k(x) + \sigma_k(x)$;
$k := k + 1$.

Le théorème suivant caractérise l'algorithme II.7.6.

THEOREME II.7.7. L'algorithme II.7.6. appliqué à un réseau d'Hitchcock de capacités a_i ($i \in \{1, \ldots, m\}$) et b_j ($j \in \{1, \ldots, n\}$) entières converge après au plus $\Sigma_{i=1}^{m} a_i$ augmentations de flot et détermine, après exécution, un flot maximum de d-coût minimum. De plus, toute paire (f_k, π_k) construite lors d'une telle exécution vérifie les relations

$$\pi_k(e_i) - \pi_k(t_j) + d_{ij} \geqslant 0,$$

$$\pi_k(e_i) - \pi_k(t_j) + d_{ij} > 0 \Longrightarrow f_k(e_i, t_j) = 0,$$

pour tout $i \in \{1, \ldots, m\}$ et $j \in \{1, \ldots, n\}$.

Démonstration. De fait, on démontre, de même qu'en II.6.13., que l'algorithme II.7.6. appliqué à un réseau d'Hitchcock de capacités a_i ($i \in \{1, \ldots, m\}$) et b_j ($j \in \{1, \ldots, n\}$) entières converge après au plus $\Sigma_{i=1}^{m} a_i$ augmentations de flot et détermine, après exécution, un flot maximum de d-coût minimum. Pour conclure, il reste, bien sûr, à démontrer les relations précédentes.

Soit $(f_o, \pi_o), \ldots, (f_N, \pi_N)$ la suite de paire de flot et de fonction d'étiquetage déterminée par une telle exécution. D'une part, les premières relations se déduisent de suite du fait qu'on a, pour tout $k \in \{0, \ldots, N-1\}$, tout $i \in \{1, \ldots, m\}$ et tout $j \in \{1, \ldots, n\}$, $(e_i, t_j) \in R(f_k)$ (puisque la capacité $c(e_i, t_j)$ est infinie) et par suite

$$\sigma_k(t_j) \leqslant \sigma_k(e_i) + \Delta_k(e_i, t_j)$$

$$\leqslant \sigma_k(e_i) + \pi_k(e_i) - \pi_k(t_j) + d_{ij}$$

c'est-à-dire

$$\pi_{k+1}(e_i) - \pi_{k+1}(t_j) + d_{ij} > 0.$$

D'autre part, on a, pour tout $k \in \{0, \ldots, N\}$, tout $i \in \{1, \ldots, m\}$ et $j \in \{1, \ldots, n\}$,

$$\pi_k(e_i) - \pi_k(t_j) + d_{ij} > 0 \Longrightarrow f_k(e_i, t_j) = 0.$$

Pour l'établir, procédons par l'absurde. Il existe alors $k \in \{0, \ldots, N\}$, $i \in \{1, \ldots, m\}$ et $j \in \{1, \ldots, n\}$ tels que $\pi_k(e_i) - \pi_k(t_j) + d_{ij} > 0$ et $f_k(e_i, t_j) > 0$. Ceci introduit une absurdité

car de la dernière relation on déduit de suite l'appartenance de (t_j,e_i) à $R(f_k)$ et par suite, grâce à un raisonnement analogue au raisonnement ci-avant, l'inégalité

$$\pi_k(t_j) - \pi_k(e_i) - d_{ij} \geqslant 0 \ . \ ///$$

Présentons, à présent, la méthode de graduation. Dans ce but, adoptons tout d'abord la convention suivante.

CONVENTION II.7.8. Dans la suite de ce paragraphe, nous associons à tout problème de transfert de Hitchcock P et à tout entier $n \in \mathbb{N}$, le problème de transfert de Hitchcock P_n obtenu à partir de P en conservant les mêmes coûts mais en imposant aux arcs (e,e_i), $i \in \{1, \ldots ,m\}$ (resp. (t_j,t), $j \in \{1, \ldots ,n\}$), les capacités

$$\left\lfloor \frac{a_i}{2^n} \right\rfloor \quad \text{(resp. } \left\lfloor \frac{b_j}{2^n} \right\rfloor \text{)}$$

où, pour tout réel x, $\lfloor x \rfloor$ désigne le plus grand entier plus petit ou égal à x.

En particulier, si le réseau de Hitchcock considéré est à capacités entières, le problème P_o équivaut au problème P.

Le lemme suivant est fondamental.

LEMME II.7.9. Pour tout entier n, le flot f est d-pseudo-extrême pour le problème P_{n+1} s'il en est ainsi du flot f/2 pour le problème P_n.

Démonstration. Il s'agit d'une simple vérification. ///

Explicitons, à présent, la méthode de graduation. Elle commence par déterminer un entier L tel que

$$\max \{ \max_{i=1,\ldots,m} a_i , \max_{j=1,\ldots,n} b_j \} < 2^L .$$

La résolution du problème P_L est aisée : le seul flot acceptable est, comme on le vérifie de suite, le flot identiquement nul. Ensuite, partant de cette solution, elle résout de proche en proche les problèmes P_{L-1}, ... , P_0 en appliquant le lemme ci-dessus. Etant donné le flot d-pseudo-extrême f, solution du problème P_n ($n \in \{1, \ldots ,L\}$) et la fonction d'étiquetage π d-compatible avec f, le flot 2f est

pris comme flot d-pseudo-extrême initial pour le problème P_{n-1}; il est, bien entendu, d-compatible avec la fonction π.

L'algorithme ci-dessous en résulte.

ALGORITHME DE GRADUATION II.7.10.

Déterminer le plus petit entier L tel que

$$\max \{ \max_{i=1,\ldots,m} a_i , \max_{j=1,\ldots,n} b_j \} < 2^L ;$$

f_{init} := *flot nul;*

Pour ℓ variant de L-1 à 0 effectuer

| *déterminer, selon l'algorithme II.7.6., un flot maximum d-pseudo-extrême f pour le problème P_ℓ en prenant pour flot initial le flot f_{init};*
| $f_{init} := 2f$.

Le théorème suivant décrit l'algorithme de graduation.

THEOREME II.7.11. L'algorithme II.7.10. appliqué à un réseau de Hitchcock de capacités a_i ($i \in \{1, \ldots, m\}$) et b_j ($j \in \{1, \ldots, n\}$) entières converge et délivre après exécution un flot maximum d-extrême. De plus, le nombre maximum d'augmentations de flot requis par une telle exécution est

$$\max(m,n) \left(2 + \left\lfloor \log_2 \frac{\Sigma_{i=1}^{m} a_i}{\max(m,n)} \right\rfloor \right) .$$

Démonstration. Seule la borne sur le nombre d'augmentations de flot n'est pas immédiate. Afin de l'établir, désignons, pour tout $\ell \in \{0, \ldots, L\}$ par f^*_ℓ le flot maximum d-pseudo-extrême solution du problème P_ℓ et par B la valeur commune des sommes $\Sigma_{i=1}^{m} a_i$ et $\Sigma_{j=1}^{n} b_j$.

Prouvons tout d'abord qu'on a, pour tout $\ell \in \{0, \ldots, L\}$

$$|f^*_\ell| \geq \max \left(0, \frac{B}{2} - \max(m,n)\right) \qquad (*) .$$

De fait, de l'existence, pour tout $i \in \{1, \ldots, m\}$ d'un nombre $r_{i,\ell}$ tel que

$$0 \leq r_{i,\ell} < 2^\ell$$

et

$$a_i = 2^\ell \left\lfloor \frac{a_i}{2^\ell} \right\rfloor + r_{i,\ell}$$

on déduit de suite les relations

$$\sum_{i=1}^{m} \left\lfloor \frac{a_i}{2^\ell} \right\rfloor = \frac{\sum_{i=1}^{m} a_i - \sum_{i=1}^{m} r_{i,\ell}}{2^\ell}$$

$$\geqslant \frac{\sum_{i=1}^{m} a_i}{2^\ell} - \sum_{i=1}^{m} \frac{2^\ell - 1}{2^\ell}$$

$$\geqslant \frac{\sum_{i=1}^{m} a_i}{2^\ell} - m \quad .$$

De même, on obtient

$$\sum_{j=1}^{n} \left\lfloor \frac{b_j}{2^\ell} \right\rfloor \geqslant \frac{\sum_{j=1}^{n} b_j}{2^\ell} - n \; .$$

Dans ces conditions, comme la définition de f_ℓ^* implique l'égalité

$$|f_\ell^*| = \min \left(\sum_{i=1}^{m} \left\lfloor \frac{a_i}{2^\ell} \right\rfloor , \sum_{j=1}^{n} \left\lfloor \frac{b_j}{2^\ell} \right\rfloor \right) ,$$

il vient

$$|f_\ell^*| \geqslant \min \left(\frac{B}{2^\ell} - m , \frac{B}{2^\ell} - n \right)$$

$$\geqslant \frac{B}{2^\ell} - \max(m,n),$$

ce qui suffit car on a, bien sûr, $|f_\ell^*| \geqslant 0$.

Démontrons ensuite qu'on a

$$\sum_{\ell=1}^{L-1} |f_\ell^*| \geqslant \sum_{\ell=1}^{B^*} \left(\frac{B}{2} - \max(m,n) \right)$$

où

$$B^* = \left\lfloor \log_2 \frac{B}{\max(m,n)} \right\rfloor .$$

En effet, vu les inégalités (*) et l'équivalence

$$\frac{B}{2^{\ell}} - \max(m,n) \leqslant 0 \Longleftrightarrow B^* \leqslant \ell ,$$

il suffit de prouver pour conclure qu'on a $B^* < L$. Or ceci résulte de suite des inégalités

$$\frac{B}{\max(m,n)} \leqslant \sum_{i=1}^{m} \frac{a_i}{m} < \sum_{i=1}^{m} \frac{2^L}{m} = 2^L .$$

Etablissons enfin la thèse. Comme chaque augmentation de flot provoque un accroissement entier de la valeur du flot et puisque le flot initial du problème $P_{\ell-1}$ est $2f^*_{\ell}$ ($\ell \in \{1, \ldots, L\}$), le nombre total d'augmentations de flot est borné par

$$|f^*_{L-1}| + \sum_{\ell=1}^{L-1} (|f^*_{\ell-1}| - |f^*_{\ell}|) = |f^*_o| - \sum_{\ell=1}^{L-1} |f^*_{\ell}| .$$

Or il vient, vu ce qui précède,

$$|f^*_o| - \sum_{\ell=1}^{L-1} |f^*_{\ell}| \leqslant |f^*_o| - \sum_{\ell=1}^{B^*} \left(\frac{B}{2^{\ell}} - \max(m,n) \right)$$

$$\leqslant B - \sum_{\ell=1}^{B^*} \frac{B}{2^{\ell}} + \sum_{\ell=1}^{B^*} \max(m,n)$$

$$= \frac{B}{2^{B^*}} + B^* \max(m,n) .$$

La thèse résulte alors du fait qu'on a, comme on le vérifie aisément

$$\frac{B}{2^{B^*}} \leqslant 2 \max(m,n). \; ///$$

II.8. La méthode de graduation comme méthode de résolution du problème du flot maximum de coût minimum

Tentons à présent de généraliser la méthode développée dans le paragraphe précédent au cas du problème du flot maximum de coût minimum. L'approche générale reste la même. Se donnant un problème de flot maximum de coût minimum P sur un réseau R, nous définissons aussi, pour tout entier n, le problème P_n comme le problème du flot maximum de coût minimum déduit de P en remplaçant la capacité c(u) de chaque arc $u \in U$ par la capacité

$$\left\lfloor \frac{c(u)}{2^n} \right\rfloor .$$

De même, nous choisissons pour valeur de L le plus petit entier i tel que

$$\min_{u \in U} c(u) < 2^i$$

et nous essayons de résoudre de proche en proche les problèmes P_{L-1}, ... , P_o en utilisant l'algorithme II.6.12. en prenant comme flot initial du problème $P_{\ell-1}$ ($\ell \in \{1, \ldots, L\}$) le double du flot solution du problème P_ℓ.

Toutefois, si f^*_ℓ désigne un flot maximum de d-coût minimum pour le problème P_ℓ ($\ell \in \{0, \ldots, L\}$) rien n'assure que le flot $2f^*_\ell$ est d-extrême. Afin de lever cette difficulté, introduisons la notion de flot d-presqu'extrême.

DEFINITION II.8.1. Soit $d : U \to \mathbb{R}^+$ une fonction de coût. Un flot f est d-<u>presqu'extrême</u> s'il existe une fonction d'étiquetage π telle que, pour tout $(x,y) \in U$,

$$\pi(x) + d(x,y) - \pi(y) > 0 \Longrightarrow f(x,y) = 0$$

$$\pi(x) + d(x,y) - \pi(y) < 0 \Longrightarrow f(x,y) \leqslant c(x,y) \leqslant f(x,y) + 1.$$

CONVENTION II.8.2. Dans la suite de ce travail, sauf mention explicite du contraire, l'appellation

d-presqu'extrême

sous-entend que d est une fonction définie de l'ensemble U des arcs du réseau considéré dans $\mathbb{R}^+$.

LEMME II.8.3. Pour tout entier n, si le flot f est d-extrême pour le problème P_{n+1} alors le flot 2f est d-presqu'extrême pour le problème P_n.

Démonstration. De fait, si f est un flot d-extrême pour le problème P_{n+1}, $n \in \mathbb{N}$, le théorème II.6.7. affirme l'existence d'une fonction d'étiquetage π telle que, pour tout arc $(x,y) \in U$, on ait

$$\pi(x) + d(x,y) - \pi(y) > 0 \Longrightarrow f(x,y) = 0$$

$$\pi(x) + d(x,y) - \pi(y) < 0 \Longrightarrow f(x,y) = \left\lfloor \frac{c(x,y)}{2^{n+1}} \right\rfloor ,$$

et, par suite, en utilisant les inégalités

$$2\left\lfloor \frac{c(x,y)}{2^{n+1}} \right\rfloor \leqslant \left\lfloor \frac{c(x,y)}{2^{n}} \right\rfloor \leqslant 2\left\lfloor \frac{c(x,y)}{2^{n+1}} \right\rfloor + 1, \quad \forall (x,y) \in U$$

telle que, pour tout $(x,y) \in U$,

$$\pi(x) + d(x,y) - \pi(y) > 0 \Longrightarrow 2f(x,y) = 0$$

$$\pi(x) + d(x,y) - \pi(y) < 0 \Longrightarrow 2f(x,y) \leqslant \left\lfloor \frac{c(x,y)}{2^{n}} \right\rfloor \leqslant 2f(x,y) + 1,$$

ce qui suffit. ///

Remarquons ici que rien n'assure la d-compatibilité de 2f et π puisque celle-ci requiert qu'on ait, pour tout $(x,y) \in U$,

$$\pi(x) + d(x,y) - \pi(y) < 0 \Longrightarrow 2f(x,y) = \left\lfloor \frac{c(x,y)}{2^{n}} \right\rfloor .$$

Il est donc intéressant de donner une méthode efficace de transformation du flot 2f en un flot de même valeur mais d-extrême pour le problème P_n. Nous décrivons ci-dessous une telle méthode. Elle peut être considérée comme une variante de l'algorithme "out-of-kilter" de Ford et Fulkerson dans laquelle les phases de changement de flot et de changement de potentiel sont combinées en un seul calcul. Afin de l'expliciter, adoptons tout d'abord la convention suivante.

CONVENTION II.8.4. Dans la suite de ce travail, étant donné un flot f, une fonction d'étiquetage π et une fonction de coût d, nous posons, pour tout arc $(x,y) \in U$

$$\bar{d}(x,y) = \pi(x) + d(x,y) - \pi(y)$$

et définissons le nombre $K_{f,\pi}(x,y)$ de l'arc $(x,y) \in U$ relatif à f et par

$$K_{f,\pi}(x,y) = \begin{cases} f(x,y) & \text{si } \bar{d}(x,y) > 0 \\ 0 & \text{si } \bar{d}(x,y) = 0 \\ c(x,y) - f(x,y) & \text{si } \bar{d}(x,y) < 0 \end{cases} .$$

La propositon suivante est immédiate.

PROPOSITION II.8.5. Un flot f et une fonction d'étiquetage π sont d-compatibles si et seulement si tout arc $(x,y) \in U$ possède un nombre $K_{f,\pi}(x,y)$ nul.

Démonstration. C'est immédiat. ///

Remarquons aussi que les nombres $K_{2f,\pi}$ relatifs au flot 2f et à la fonction d'étiquetage π du lemme II.8.3. sont nuls ou égaux à 1. L'algorithme suivant tente de déduire de la paire $(2f,\pi)$ une nouvelle paire (f',π') de flot et de fonction d'étiquetage pour laquelle tous les nombres $K_{f',\pi'}$ sont nuls.

ALGORITHME II.8.6. Soient f un flot et π une fonction d'étiquetage non d-compatibles d'un réseau vérifiant l'hypothèse II.6.1..

(1). Pour tout arc $(x,y) \in U(\ell)$ effectuer

poser

$$\bar{\Delta}_d(x,y) = \begin{cases} \bar{d}(x,y) & \text{si } (x,y) \in U \text{ et } \ell(x,y) < c(x,y) \\ -\bar{d}(x,y) & \text{si } (y,x) \in U \text{ et } \ell(y,x) > 0 \end{cases} ;$$

donner à l'arc (x,y) le poids $\rho(x,y) = \max\{\bar{\Delta}_d(x,y), 0\}$.

(2). Choisir un arc $(x^*,y^*) \in U(\ell)$ tel que

$$(x^*,y^*) \in U \text{ et } \bar{d}(x,y) < 0 \text{ et } \ell(x^*,y^*) < c(x^*,y^*)$$

ou

$$(y^*,x^*) \in U \text{ et } \bar{d}(x,y) > 0 \text{ et } \ell(y^*,x^*) > 0.$$

(3). Poser $N^* = \{x : x = y^*$ *ou il existe un chemin de* y^* *à* x *dans* $R(f)\}$;

Pour tout $x \in N^*$ *effectuer*

$\delta(x)$:= *poids d'un chemin de poids minimum de* y^* *à* x *dans* $R(f)$;

Pour tout $x \in X \setminus N^*$ *effectuer*

$$\delta(x) := \max_{\{(y,x)\in U(f) \,:\, y\notin N^*,\, x\in N^*\}} \left\lfloor \pi(x) - \bar{\Delta}_d(y,x) \right\rfloor ;$$

Pour tout $x \in X$ *effectuer*

$\pi'(x) := \pi(x) + \delta(x)$.

(4). Si $x^* \in N^*$ *alors*

choisir un cycle C de $R(f)$ *constitué de* (x^*, y^*) *et d'un chemin de poids minimum d'extrémités initiale* y^* *et terminale* x^* *et obtenir* f' *à partir de* f *en effectuant une augmentation de flot le long de C*

sinon $f' := f$.

Le théorème suivant décrit l'algorithme II.8.6.. Comme sa preuve sort du cadre de ce travail, nous l'admettons.

THEOREME II.8.7. L'algorithme II.8.6. converge et détermine après exécution un flot f' et une fonction d'étiquetage π' tels que, en utilisant les notations de II.8.6.,

(i) $K_{f',\pi'}(x,y) \leq K_{f,\pi}(x,y)$, pour tout arc $(x,y) \in U$,

(ii) si (x^*,y^*) appartient à U alors

$$K_{f',\pi'}(x^*,y^*) \leq K_{f,\pi}(x^*,y^*) - 1,$$

(iii) si (y^*,x^*) appartient à U alors

$$K_{f',\pi'}(y^*,x^*) \leq K_{f,\pi}(y^*,x^*) - 1.$$

Démonstration. Résultat admis (cfr. [13]). ///

Ainsi par itération de l'algorithme II.8.6. on peut espérer convertir une paire (f,π) de flot et de fonction d'étiquetage non d-compatibles en une paire (f',π') d-compatible telle que $|f'| = |f|$. De plus, comme chaque itération réduit d'au moins une unité la somme des valeurs de la fonction $K_{f,\pi}$, le processus entier requiert au plus

$$\sum_{u \in U} K_{f,\pi}(u)$$

itérations.

En guise d'application, solutionnons le problème soulevé au début de ce paragraphe. Comme nous l'avons déjà noté, si f_ℓ^* désigne un flot maximum de d-coût minimum pour le problème P_ℓ, rien n'assure que le flot $2f_\ell^*$ soit d-extrême. Il est toutefois, vu le lemme II.8.3. d-pseudo-extrême. Dans ces conditions, il reste, pour résoudre ce problème, à appliquer le processus ci-dessus qui, comme nous l'avons établi, fournit un flot de même valeur que $2f_\ell^*$ et d-extrême pour le problème $P_{\ell-1}$ après

$$\sum_{u \in U} K_{f,\pi}(u)$$

itérations c'est-à-dire, comme on a, en utilisant la fonction d'étiquetage π du lemme II.8.3.,

$$K_{2f,\pi}(x,y) \in \{0,1\},$$

pour tout arc $(x,y) \in U$, après

$$m$$

itérations.

Enonçons à présent la méthode de graduation résolvant le problème du flot maximum de coût minimum.

ALGORITHME DE GRADUATION II.8.8.

Déterminer le plus petit entier L tel que

$$\max_{u \in U} c(u) < 2^L ;$$

f_{init} := *flot nul;*

π_{init} := *fonction d'étiquetage nulle;*

Pour ℓ variant de L-1 à 1 effectuer

déterminer, pour le problème P_ℓ, grâce à l'algorithme II.6.12., un flot maximum de d-coût minimum f et une fonction d'étiquetage π d-compatible avec f, à partir du flot initial f_{init} et de la fonction d'étiquetage π_{init};

appliquer l'algorithme II.8.6. à la paire $(2\ell,\pi)$ répétitivement jusqu'à obtenir une paire (g,Θ) de flot et de fonction d'étiquetage d - compatibles;

$\pi_{init} := \Theta$;

$f_{init} := g$;

Déterminer grâce à l'algorithme II.6.12. un flot maximum de d - coût minimum f pour le problème P_o à partir du flot initial f_{init} et de la fonction d'étiquetage π_{init}.

Le théorème suivant le caractérise.

THEOREME II.8.9. Soit R un réseau de transport à capacités entières vérifiant l'hypothèse II.6.1.. Soient, en outre, T, le nombre maximum d'arcs incidents vers l'intérieur à une coupe de R et f* un flot de valeur maximum dans R. L'algorithme II.8.8. appliqué au réseau R converge et fournit, après exécution, un flot maximum de d - coût minimum. De plus, une telle exécution ne requiert pas plus de $m(L-1)$ applications de l'algorithme II.8.6. et plus de

$$\frac{2^L - 1}{2^L} |f^*| + (L-1)\,T$$

augmentations de flot.

Démonstration. Seule la borne sur le nombre d'augmentations de flot n'est pas immédiate. Afin de l'établir, démontrons tout d'abord qu'on a, pour tout $\ell \in \{0, \ldots, L\}$, en désignant par f^* le flot solution du problème P_ℓ,

$$|f^*_\ell| \geq \frac{1}{2^\ell} |f^*_o| - T .$$

De fait, le théorème du flot maximum et de la coupe minimum I.3.4. affirme qu'on a

$$|f^*_\ell| = \min \left\{ \sum_{\substack{x\in S \\ y\in \bar{S}}} \left\lfloor \frac{c(x,y)}{2^\ell} \right\rfloor \;:\; (S,\bar{S}) \text{ constitue une partition de } X \text{ telle que } e \in S,\ t \in \bar{S} \right\}$$

$$= \sum_{\substack{x\in Z \\ y\in \bar{Z}}} \left\lfloor \frac{c(x,y)}{2^\ell} \right\rfloor$$

pour une telle partition $(Z,\bar{Z})$ et, par suite, comme tout arc $u \in U$ vérifie l'inégalité

$$\left\lfloor \frac{c(u)}{2^{\ell}} \right\rfloor \geqslant \frac{c(u)}{2^{\ell}} - 1$$

qu'on a

$$|f^*_{\ell}| \geqslant \frac{1}{2^{\ell}} c(Z,\bar{Z}) - T$$

$$\geqslant \frac{1}{2^{\ell}} |f^*_o| - T,$$

ce qui suffit.

Etablissons à présent la thèse. Comme chaque augmentation de flot provoque un accroissement entier de la valeur du flot et puisque le flot initial du problème $P_{\ell-1}$ est $2f^*_{\ell}$ ($\ell \in \{1, \ldots, L\}$), le nombre total d'augmentations de flot est borné par

$$|f^*_{L-1}| + \sum_{\ell=1}^{L-1} (|f^*_{\ell-1}| - |f^*_{\ell}|) = |f^*_o| - \sum_{\ell=1}^{L-1} |f^*_{\ell}| .$$

Or il vient, vu ce qui précède,

$$|f^*_o| - \sum_{\ell=1}^{L-1} |f^*_{\ell}| \leqslant |f^*_o| - \sum_{\ell=1}^{L-1} \left(\frac{1}{2^{\ell}} |f^*_o| - T\right)$$

$$\leqslant \frac{2^{L-1}}{2^{L}} |f^*_o| + (L-1)\,T .$$

D'où la conclusion. ///

II.9. La méthode de Dinic

Afin d'expliciter la méthode de Dinic, introduisons, tout d'abord, la notion de réseau de référence.

DEFINITION II.9.1. Le réseau de référence associé au réseau de transport $R = (X,U,c)$ et à un flot f de ce réseau est le réseau $(X,U_{R,f},c)$ dont l'ensemble des arcs $U_{R,f}$ est constitué de l'ensemble des arcs composant les chemins de R(f) d'extrémité initiale e et de longueur inférieure ou égale à la distance de e à t dans R(f).

CONVENTION II.9.2. Dans la suite de ce travail, la notation

$$(X,U_{R,f},c)$$

désigne le réseau de référence associé au flot f et au réseau de transport $R = (X,U,c)$.

La proposition suivante énonce une propriété fondamentale du réseau de référence.

PROPOSITION II.9.3. Soient f un flot d'un réseau de transport $R = (X,U,c)$ et P une chaîne f-augmentante de longueur minimum L. Soit, en outre, f' le flot obtenu par augmentation du flot f le long de P. Toute chaîne f'-augmentante est de longueur supérieure ou égale à L. De plus, toute chaîne f'-augmentante de longueur égale à L appartient au réseau de référence $(X,U_{R,f},c)$.

Démonstration. Notons pour tout couple de sommets $x,y \in X$, d(x,y) la distance séparant x de y dans $(X,U_{r,f},c)$ c'est-à-dire la longueur d'un chemin de longueur minimum dans le réseau de référence $(X,U_{r,f},c)$ joignant x à y.

Comme le lemme II.4.6. affirme que tout arc $(x,y) \in R(f')\backslash R(f)$ est tel que $(y,x) \in P$ et puisque l'augmentation du flot f le long de P ne modifie que les flux des arcs ayant leurs extrémités sur P, il suffit, bien sûr, de prouver que l'addition à $U_{R,f}$ d'un arc (a,b) tel que $(b,a) \in P$ n'introduit que des chemins d'extrémités initiale e et terminale t de longueur strictement supérieure à L.

De fait, tout chemin élémentaire d'extrémités initiale e et terminale t contenant l'arc (a,b) est constitué d'un chemin de $(X,U_{R,f},c)$ joignant e à a de longueur nécessairement supérieure ou égale à d(e,a), de l'arc (a,b) de longueur l et d'un chemin de $(X,U_{R,f},c)$ joignant b à t de longueur supérieure ou égale à d(b,t). Par suite, la longueur totale de ce chemin est supérieure ou égale à

$$d(e,a) + l + d(b,t)$$

c'est-à-dire à

$$d(e,a) + d(a,t) + 2$$

puisque de l'appartenance de (b,a) à P, on déduit de suite l'égalité $d(b,t) = d(a,t) + 1$. Dans ces conditions, la thèse résulte immédiatement de l'égalité $d(e,a) + d(a,t) = L$. ///

Donnons à présent un algorithme de construction de réseau de référence. L'algorithme II.9.4. répond à cette spécification. Il utilise une technique d'exploration des sommets en largeur d'abord ainsi que les structures de données suivantes. A chaque sommet x est associé

- un numéro, *num(x)*
- l'ensemble *succ(x)* des successeurs de x dans le réseau de référence.

De plus, la valeur booléenne *echec* détermine si la sortie t est accessible de l'entrée e dans $(X, U_{R,f}, c)$.

ALGORITHME CONSTRUCTION_RESEAU_REFERENCE(f,echec) II.9.4.
Soit f un flot du réseau de transport R = (X,U,c).

```
Pour tout x ∈ X \ {e} effectuer
  num(x) := 0;
  succ(x) := ϕ;
num(e) := 1;
i := 1;
Répéter
  fin_itérations := vrai;
  Pour tout x ∈ X effectuer
    si num(x) = i alors
      Pour tout y ∈ X effectuer
        si { ( (x,y) ∈ U et f(x,y) < c(x,y) )   ou
             ( (y,x) ∈ U et f(y,x) > 0 ) }   et  num(y) ∈ {0, i+1}
          alors
            num(y) := i + 1;
            succ(x) := succ(x) ∪ {y};
            fin_itérations := faux;
  i := i + 1;
tant que (fin_itérations = faux) et (num(t) = 0);
```

```
Si num(t) = 0
  alors echec := vrai
  sinon echec := faux.
```

Le théorème suivant caractérise l'algorithme II.9.4.

THEOREME II.9.5. Soit f un flot du réseau de transport R = (X,U,c). L'algorithme CONSTRUCTION RESEAU REFERENCE(f,echec) appliqué au flot f converge. Son exécution établit les propriétés suivantes

(1) L'application *num* : X → ℕ induit un tri topologique des sommets du réseau de référence $(X,U_{R,f},c)$. De plus, pour tout $x \in X$, la valeur num(x) - 1 si elle diffère de -1 donne la longueur d'un chemin de longueur minimum entre e et x dans $(X,U_{R,f},c)$ et sinon indique que le sommet x est inaccessible de e dans $(X,U_{R,f},c)$.

(2) Pour tout $x \in X$, *succ(x)* est l'ensemble des successeurs de x dans $(X,U_{R,f},c)$.

(3) La variable booléenne, *echec*, possède la valeur vrai si et seulement si la sortie t est accessible de l'entrée e dans $(X,U_{R,f},c)$.

Démonstration. Il s'agit d'une simple vérification. ///

Décrivons à présent la méthode de Dinic. Plutôt que de construire, comme dans les méthodes précédentes, à chaque itération le graphe d'écart du flot courant, E.A. Dinic propose d'effectuer certaines augmentations successives de flot sur un même réseau, le réseau de référence associé au flot courant. Sa méthode repose donc sur une suite de phases.

Chaque phase commence par la construction du réseau de référence, défini ci-dessus, associé au flot courant. Une telle construction peut, comme nous l'avons montré, s'effectuer grâce à un algorithme d'exploration en largeur d'abord des sommets du graphe d'écart que l'on arrête lorsque la sortie a pu être marquée. Bien sûr, si la construction du réseau de référence échoue c'est-à-dire si l'algorithme précédent se termine sans avoir visité t, alors le flot courant est, comme l'établit le théorème II.1.2., de valeur maximale. Dans

la cas contraire, grâce à une exploration en profondeur d'abord des sommets du réseau de référence, une chaîne augmentante est déterminée et le flot courant augmenté. Les arcs du réseau de référence n'appartenant pas au graphe d'écart du nouveau flot sont éliminés et une nouvelle recherche en profondeur d'abord d'une chaîne augmentante commence ... Et ainsi de suite jusqu'à ce que la sortie t soit devenue inaccessible. A ce moment, une nouvelle phase commence.

Avant d'énoncer l'algorithme qui en découle, étudions tout d'abord les algorithmes auxiliaires suivants. Le premier, RECHERCHE_CHEMIN(s) vise à déterminer à partir d'un sommet s, un chemin entre s et t dans le réseau de référence décrit par les ensembles $succ(x)$ ($x \in X$) des successeurs de x dans ce réseau. Le second, AUGMENTATION_FLOT(f) a pour but d'augmenter le flot f le long de la chaîne augmentante définie par l'application $suiv$ et d'éliminer les arcs du réseau de référence n'appartenant pas au graphe d'écart du nouveau flot.

ALGORITHME RECHERCHE_CHEMIN(s) II.9.6. Soit s un sommet.

```
Pour tout x ∈ succ(s) effectuer
  si t_marqué = non alors
    si x = t
    alors
      t_marqué := oui;
      suiv(s) := t;
    sinon
      suiv(s) := x;
      RECHERCHE_CHEMIN(x).
```

ALGORITHME AUGMENTATION_FLOT(f) II.9.7. Soit f un flot.

```
x := e;
min := +∞;
Répéter
  y := suiv(x);
  si (x,y) ∈ U et (y,x) ∉ U alors
    m := c(x,y) - f(x,y);
  si (x,y) ∉ U et (y,x) ∈ U alors
    m := f(y,x);
  si (x,y) ∈ U et (y,x) ∈ U alors
    m := c(x,y) - f(x,y) + f(y,x);
```

```
    si min > m alors
      min := m;
    x := y;
  tant que x ≠ t;
  x := e;
  Répéter
    y := suiv(x);
    si (x,y) ∈ U et (y,x) ∉ U alors
      ℓ(x,y) := ℓ(x,y) + min;
      si ℓ(x,y) = c(x,y) alors succ(x) := succ(x)\{y};
    si (x,y) ∉ U et (y,x) ∈ U alors
      ℓ(y,x) := ℓ(y,x) - min;
      si ℓ(y,x) = 0 alors succ(x) := succ(x)\{y};
    si (x,y) ∈ U et (y,x) ∈ U alors
      ℓ(x,y) := ℓ(x,y) + min{min, c(x,y) - ℓ(x,y)};
      ℓ(y,x) := ℓ(y,x) - max{0, min - c(x,y) + ℓ(x,y)};
      si ℓ(x,y) = c(x,y) et ℓ(y,x) = 0 alors
        succ(x) := succ(x)\{y};
    x := y;
  tant que x ≠ t.
```

Les deux théorèmes suivants les caractérisent.

THEOREME II.9.8. Soit G un graphe partiel d'un réseau de référence. Si, pour tout $x \in X$, l'ensemble $succ(x)$ est l'ensemble des successeurs de x dans G, alors toute exécution de RECHERCHE_CHEMIN(s) se termine et établit les propriétés

(i) *t_marqué* a la valeur oui si et seulement s'il existe un chemin entre s et t dans G;

(ii) si *t_marqué* a la valeur oui, alors l'application $suiv : X \to X$ définit un chemin entre s et t dans G.

De plus, la complexité de l'algorithme II.9.6. est $\mathcal{O}(n)$ au plus et $\mathcal{O}(m)$ au plus.

Démonstration. Comme d'une part, les propriétés (i) et (ii) sont immédiates et que, d'autre part, la convergence de l'algorithme II.9.6. se déduit de suite du fait que tout graphe partiel d'un réseau de référence peut être décomposé en niveau, il suffit pour conclure d'établir que la complexité l'algorithme II.9.6. est $\mathcal{O}(n)$ au plus et $\mathcal{O}(m)$ au plus. Or si T est le temps d'exécution de RECHERCHE_CHEMIN(x)

et T' le temps d'exécution de l'instruction conditionnelle *si t_marqué = non alors ...* où l'on ne prend en considération que l'appel de la procédure RECHERCHE_CHEMIN, le temps d'exécution de RECHERCHE_CHEMIN(s) est borné par $(n-1)(T+T')$ et par $m(T+T')$. D'où la conclusion. ///

THEOREME II.9.9. Soient f un flot du réseau de transport R et G un graphe partiel d'un réseau de référence associé à R. Si l'application *suiv* : $X \to X$ définit une chaîne f-augmentante P dans G et si, pour tout $x \in X$, l'ensemble *succ(x)* est l'ensemble des successeurs de x dans G, alors l'algorithme II.9.7. appliqué au flot f converge et son exécution établit les propriétés

(i) f_{term} est le flot obtenu par augmentation du flot f_{init} le long de la chaîne P;

(ii) pour tout $x \in X$, *succ(x)* désigne l'ensemble des successeurs de x dans $G \setminus R(f_{term})$;

où f_{init} et f_{term} représentent respectivement le flot f avant et après exécution. De plus, la complexité de l'algorithme II.9.7. est $\mathcal{O}(n)$ au plus et $\mathcal{O}(m)$ au plus.

Démonstration. Il s'agit d'une simple vérification. ///

Enonçons à présent l'algorithme de Dinic.

ALGORITHME DE DINIC II.9.10.

```
Choisir un flot initial f0;
f := f0;
Répéter
  CONSTRUCTION_RESEAU_REFERENCE(f, echec);
  si echec = faux alors
    Répéter
      t_marqué := non;
      RECHERCHE_CHEMIN(e);
      si t_marqué = oui alors
        AUGMENTATION_FLOT(f);
    tant que t_marqué = oui;
tant que echec = faux.
```

Le théorème ci-dessous le décrit.

THEOREME II.9.11. Si les capacités des arcs du réseau de transport R sont entières, alors l'algorithme de Dinic II.9.10. exécuté à partir d'un flot initial à valeurs entières converge et, détermine après exécution un flot dans R de valeur maximum et à valeurs entières. Sa complexité est $\mathcal{O}(n^4)$ au plus.

Démonstration. De fait, d'une part, la convergence et la détermination d'un flot de valeur maximum et à valeurs entières résultent aussitôt des théorèmes II.2.1., II.9.5., II.9.8., II.9.9. et de la proposition II.9.3..

D'autre part, la complexité est $\mathcal{O}(n^4)$ au plus. Afin de l'établir, prouvons tout d'abord que la complexité de la boucle *Répéter ... tant que t_marqué = oui* est $\mathcal{O}(n^3)$ au plus. De fait, à chaque itération de cette boucle un arc au moins est retiré du réseau de référence. Dès lors, comme les algorithmes RECHERCHE_CHEMIN et AUGMENTATION_FLOT ont une complexité de $\mathcal{O}(n)$ au plus, la complexité de la boucle est $\mathcal{O}(n^3)$ au plus.

Pour conclure, il reste, comme la complexité de l'algorithme CONSTRUCTION_RESEAU_REFERENCE est $\mathcal{O}(n^3)$ au plus, à prouver que toute exécution de l'algorithme de Dinic requiert au plus n itérations de la boucle *Répéter ... tant que echec = faux*. Ceci résulte de suite du fait que chacune augmente, vu la proposition II.9.3., d'une unité au moins la longueur minimum des chaînes augmentantes et du fait qu'à tout chemin non élémentaire on peut toujours associer un chemin élémentaire (de longueur inférieure ou égale à n) ayant mêmes extrémités. //

On peut encore établir une meilleure complexité temporelle dans le cas particulier d'un réseau de transport à capacités unitaires. Afin d'y parvenir, montrons tout d'abord comment on peut modifier l'algorithme CONSTRUCTION_RESEAU_REFERENCE pour obtenir un algorithme répondant à la même spécification mais de complexité $\mathcal{O}(n^2)$ au plus. Dans ce but, associons à chaque sommet x, l'ensemble $S(x)$ des successeurs de x dans U et $P(x)$ des prédécesseurs de x dans U. De plus, construisons à la i^{eme} itération l'ensemble $V(i+1)$ des sommets numérotés i + 1. L'algorithme CONSTRUCTION_RESEAU_REFERENCE devient

ALGORITHME CONSTRUCTION_RESEAU_REFERENCE(f,echec) II.9.12.
Soit f un flot d'un réseau de transport R = (X,U,c).

```
Pour tout x ∈ X\{e} effectuer
  num(x) := 0;
  succ(x) := ϕ;
num(e) := 1;
V(1) := {e};
i := 1;
Répéter
  V(i+1) := ϕ;
  fin_itérations := vrai;
  Pour tout x ∈ V(i) effectuer
    Pour tout y ∈ S(x) effectuer
      si f(x,y) < c(x,y) et num(y) ∈ {0, i+1} alors
        num(y) := i+1;
        succ(x) := succ(x) ∪ {y};
        V(i+1) := V(i+1) ∪ {y};
        fin_itérations := faux;
    Pour tout y ∈ P(x) effectuer
      si f(y,x) > 0 et num(y) ∈ {0,i+1} alors
        num(y) := i+1;
        succ(x) := succ(x) ∪ {y};
        V(i+1) := V(i+1) ∪ {y};
        fin_itérations := faux;
  i := i + 1;
tant que (fin_itérations = faux) et (num(t) = 0);
Si num(t) = 0
  alors echec := vrai;
  sinon echec := faux.
```

Bien que l'algorithme reste apparemment de complexité $\mathcal{O}(n^3)$ au plus, on peut démontrer le théorème suivant.

THEOREME II.9.13. La complexité de l'algorithme CONSTRUCTION_RESEAU REFERENCE(f,echec) II.9.12. (où f désigne un flot) est $\mathcal{O}(n^2)$ au plus et $\mathcal{O}(m)$ au plus.

Démonstration. De fait, le théorème II.9.5. affirme que l'algorithme II.9.12. converge. Soit donc N le nombre d'itérations de la boucle

Répéter ... tant que (fin_itérations = faux) et (num(t) = 0). Soit en outre T le temps maximum d'exécution des instructions conditionnelles *si* $\ell(x,y) < c(x,y)$ ... et *si* $\ell(y,x) > 0$ ... . Posons de plus, pour tout $x \in X$, $n(P(x))$ et $n(S(x))$ respectivement le nombre d'éléments de $P(x)$ et $S(x)$.

Etablissons à présent la thèse. Le temps passé à l'examen du sommet $x \in V(i)$ $(i \in \{1, \ldots, n\})$ est au plus

$$n(S(x))\,T + n(P(x))\,T$$

et, par suite, le temps maximum requis pour l'analyse des sommets de $V(i)$ $(i \in \{1, \ldots, n\})$ est

$$\sum_{x \in V(i)} \left(n(S(x)) + n(P(x)) \right) T .$$

Dès lors, le temps total d'exécution de cette boucle est, au plus,

$$\left[\sum_{i=1}^{N} \sum_{x \in V(i)} n(S(x)) + \sum_{i=1}^{N} \sum_{x \in V(i)} n(P(x)) \right] T.$$

Ceci conclut la démonstration car comme tout arc est identifié par son extrémité initiale et son extrémité terminale et puisque, par construction, les ensembles $V(i)$ sont deux à deux disjoints, les deux sommes ci-dessus sont majorées par le nombre m d'arcs du réseau et par conséquent, le temps total d'exécution de la boucle est majoré par $2m\,T$ et par $n(n-1)\,T$. ///

Nous pouvons à présent démontrer le théorème

THEOREME II.9.14. Appliqué à des réseaux de transport de capacités unitaires et à des flots initiaux nuls, l'algorithme de Dinic a une complexité de $\mathcal{O}(n^3)$ au plus.

Démonstration. Vu l'algorithme précédent et la démonstration du théorème II.9.11., il suffit de prouver que le nombre d'itérations de la boule *Répéter...tant que t_marqué = oui* de l'algorithme II.9.10. est n au plus. De fait, comme tous les arcs ont une capacité unité, à chaque itération, l'augmentation de la valeur du flot est 1 et l'arc de la chaîne augmentante utilisée d'origine e est éliminé du réseau de référence. ///

On peut même aller plus loin! Dans ce but, introduisons la notion de réseau d'écart.

DEFINITION II.9.15. Soit un flot f d'un réseau de transport $R = (X,U,c)$ à capacités unitaires. Le réseau d'écart associé au réseau R et au flot f est le réseau obtenu en associant à tout arc $u = (x,y) \in U$ l'arc $\vec{u}$ d'extrémités initiale x et terminale y de capacité $c(u) - f(u)$ si on a $c(u) - f(u) > 0$ et l'arc $\overleftarrow{u}$ d'extrémités initiale y et terminale x de capacité $f(y,x)$ si on a $f(y,x) > 0$.

CONVENTION II.9.16. Dans la suite de ce travail, nous notons

$$(X,U_f,c_f)$$

le réseau d'écart associé au réseau (X,U,c) et au flot f. En outre, cette notation sous-entend que (X,U,c) est un réseau de transport et f un flot de ce réseau.

REMARQUE II.9.17. Il peut exister deux arcs distincts entre deux sommets du réseau d'écart. C'est le cas notamment du réseau d'écart associé au flot f et au réseau (X,U,c) pour lesquels il existe une paire de sommets $x,y \in X$ telle que $(x,y) \in U$, $(y,x) \in U$, $c(x,y) - f(x,y) > 0$ et $f(y,x) > 0$. Toutefois, comme on le vérifie de suite, il n'existe jamais plus de deux arcs ayant mêmes extrémités initiales et mêmes extrémités terminales.

Les lemmes suivants seront utiles dans la suite.

LEMME II.9.18. La valeur d'un flot maximum dans le réseau d'écart (X,U_f,c_f) est $M-|f|$, où M désigne la valeur d'un flot maximum dans (X,U,c).

Démonstration. De fait, pour tout sous-ensemble S de X tel que $e \in S$ et $t \notin S$, on a

$$\sum_{\substack{x\in S \\ y\in \overline{S} \\ (x,y)\in U_f}} c_f(x,y) = \sum_{\substack{x\in S \\ y\in \overline{S} \\ (x,y)\in U}} c(x,y) - f(x,y) + \sum_{\substack{x\in S \\ y\in \overline{S} \\ (y,x)\in U}} f(y,x)$$

c'est-à-dire, vu le lemme I.3.3.,

$$\sum_{\substack{x\in S\\ y\in \bar{S}\\ (x,y)\in U_f}} c_f(x,y) = \sum_{\substack{x\in S\\ y\in \bar{S}\\ (x,y)\in U}} c(x,y) - |f| .$$

Dans ces conditions, toute coupe minimale dans (X,U_f,c_f) correspond à une coupe minimale dans (X,U,c) et réciproquement. La thèse résulte alors du théorème du flot maximum et de la coupe minimum. ///

LEMME II.9.19. Soient (X,U,c) un réseau de transport de capacités unitaires et f un flot de ce réseau à valeurs entières. S'il en existe, la longueur minimum d'un chemin d'origine e et d'extrémité terminale t dans le graphe d'écart associé au flot nul et au réseau d'écart (X,U_f,c_f) est au plus

$$\frac{2\sqrt{2}\,n}{M}$$

où M désigne la valeur d'un flot maximum dans le réseau d'écart.

Démonstration. Soit L la longueur d'un chemin de longueur minimum d'origine e et d'extrémité terminale t dans le graphe d'écart G associé au flot nul et au réseau d'écart (X,U_f,c_f). Soient, en outre, pour tout $i \in \{0, \ldots, L\}$, V_i l'ensemble des sommets accessibles de e dans G par un chemin de longueur minimum i et V_{L+1} l'ensemble des sommets n'appartenant pas aux ensembles précédents.

Etablissons tout d'abord qu'on a, pour tout $k \in \{0, \ldots, L-1\}$,

$$M \leqslant 2|V_k||V_{k+1}| \qquad (*) .$$

De fait, pour tout $k \in \{0, \ldots, L-1\}$, l'ensemble

$$\bigcup_{i=0}^{k} V_i$$

de complémentaire dans G, $\bigcup_{i=k+1}^{L+1} V_i$, contient e et ne contient pas t. Dès lors, le théorème du flot maximum et de la coupe minimum I.3.4. affirme, pour tout $k \in \{0, \ldots, L-1\}$, que la valeur M d'un flot maximum est inférieure ou égale à la capacité de la coupe associée à $\bigcup_{i=k+1}^{L+1} V_i$ c'est-à-dire, comme les arcs ont une capacité unitaire, au nombre d'arcs incidents vers l'intérieur à $\bigcup_{i=k+1}^{L+1} V_i$ soit encore, vu la remarque II.9.17., au double d'arcs d'origine dans V_k et d'extrémité terminale dans V_{k+1}. D'où la conclusion.

Démontrons à présent la thèse. Les égalités (*) impliquent qu'on a, pour tout $k \in \{0, \ldots, L-1\}$,

$$|V_k| \leqslant \sqrt{\frac{M}{2}} \implies |V_{k+1}| \geqslant \sqrt{\frac{M}{2}}$$

et dès lors, selon que l'on a $|V_o| \leqslant \sqrt{M/2}$ ou non, pour tout k impair ou pair de $\{0, \ldots, L-1\}$,

$$|V_k| \geqslant \sqrt{\frac{M}{2}} \ .$$

Dans ces conditions, de l'inégalité $\sum_{i=0}^{L} |V_i| \leqslant n$, on déduit de suite

$$\begin{aligned} n &\geqslant \sum_{i=0}^{L} |V_i| \\ &\geqslant \sum_{\substack{i=0 \\ i \text{ impair}}}^{L} |V_i| \qquad \left(\text{resp. } \sum_{\substack{i=0 \\ i \text{ pair}}}^{L} |V_i|\right) \\ &\geqslant \sqrt{\frac{M}{2}} \sum_{\substack{i=0 \\ i \text{ impair}}}^{L} 1 \qquad \left(\text{resp. } \sum_{\substack{i=0 \\ i \text{ pair}}}^{L} 1\right) \\ &\geqslant \sqrt{\frac{M}{2}} \left\lfloor \frac{L+1}{2} \right\rfloor \\ &\geqslant \sqrt{\frac{M}{2}} \ \frac{L}{2} \end{aligned}$$

ce qui suffit. ///

Le résultat suivant fournit une complexité inférieure au théorème II.9.14..

THEOREME II.9.20. Appliqué à des réseaux de transport de capacités unitaires et à des flots initiaux nuls, l'algorithme de Dinic a une complexité de $\mathcal{O}(n^{8/3})$ au plus.

Démonstration. Comme la complexité temporelle de l'algorithme CONSTRUCTION_RESEAU_REFERENCE est $\mathcal{O}(n^2)$ au plus et puisque, comme le prouve la démonstration du théorème II.9.14., la boucle *répéter ... tant que t_marqué = oui* a une complexité temporelle de $\mathcal{O}(n^2)$ au plus, il suffit d'établir qu'une exécution de l'algorithme de Dinic appliqué à un réseau de transport à capacités unitaires $R = (X,U,c)$ requiert

au plus, $n^{2/3}$ itérations de la boucle *répéter ... tant que echec = faux*. Deux cas se présentent.

Dans le premier, la valeur M d'un flot maximum dans R est inférieure ou égale à $n^{2/3}$. La thèse est alors établie car toute itération de cette boucle augmente d'une unité au moins la valeur du flot courant.

Dans le second, on a $M > n^{2/3}$. Désignons alors par F le flot f tel qu'il est défini au début de l'itération pendant laquelle la valeur du flot courant atteint la valeur $M - n^{2/3}$. On a, bien sûr,

$$|F| < M - n^{2/3}$$

et, par suite, vu le lemme II.9.18., la valeur M_F d'un flot maximum dans le réseau d'écart (X, U_F, c_F) vérifie les inégalités

$$\begin{aligned} M_F &= M - |F| \\ &> M - (M - n^{2/3}) \\ &= n^{2/3} . \end{aligned}$$

Dans ces conditions, le lemme II.9.19. affirme que la longueur minimum L d'une chaîne F-augmentante dans R - étant comme on le vérifie de suite égale à la longueur minimum d'un chemin d'origine e et d'extrémité terminale t dans le graphe d'écart associé au réseau (X, U_F, c_F) et au flot nul - vérifie les relations

$$L \leqslant \frac{2\sqrt{2}\, n}{\sqrt{M_F}} = \frac{2\sqrt{2}\, n}{\sqrt{n^{2/3}}} = 2\sqrt{2}\, n^{2/3} .$$

Dès lors, puisque la proposition II.9.3. implique que la longueur minimum des chaînes augmentantes augmente d'une unité au moins à chaque itération, le nombre d'itérations effectuées jusqu'à l'obtention de F est au plus

$$2\sqrt{2}\, n^{2/3} .$$

La thèse résulte aors du fait que la valeur du flot courant augmente aussi d'une unité au moins à chaque itération. ///

CHAPITRE III

LA METHODE DE REDISTRIBUTION DE FLUX

III.1. Les fonctions de poids

Ce chapitre décrit la méthode de redistribution de flux proposée par B. Kinariwala et A.G. Rao dans [34]. Elle repose essentiellement sur l'accroissement et la diminution locale de flux, les équations de conservation de la somme des flux aux sommets étant temporairement délaissées. Toutefois, afin de mémoriser, en tout sommet, la différence entre la somme des flux des arcs qui lui sont incidents vers l'extérieur et la somme des flux des arcs qui lui sont incidents vers l'intérieur, nous introduisons la notion de fonction de poids et démontrons quelques unes de ses propriétés.

DEFINITION III.1.1. La fonction de poids associée à la fonction admissible f du réseau de transport $R = (X,U,c)$ est la fonction w_f définie par les égalités

$$\forall x \in X : w_f(x) = f(x,X) - f(X,x).$$

Pour tout $x \in X$, la quantité $w_f(x)$ est appelée le f-poids de x.

CONVENTION III.1.2. Dans la suite de ce travail, la notation

$$w_f$$

désigne la fonction de poids associée à la fonction f. Elle sous-entend que f est une fonction admissible d'un réseau (X,U,c). De plus, pour tout sous-ensemble $Y \subset X$, nous posons

$$Y_f^+ = \{y \in Y : w_f(y) > 0\},$$

$$Y_f^- = \{y \in Y : w_f(y) < 0\},$$

$$Y_f^o = \{y \in Y : w_f(y) = 0\}.$$

Enfin, quand le contexte permet d'omettre la fonction f, nous écrivons w, Y^+, Y^-, Y^o en lieu et place respectivement de w_f, Y_f^+, Y_f^-, Y_f^o.

PROPOSITION III.1.3. On a

$$\sum_{x \in X} w_f(x) = 0 .$$

Démonstration. De fait, vu la définition III.1.1., on a

$$\sum_{x \in X} w_f(x) = \sum_{x \in X} \Big(\sum_{u \in \omega^+(x)} f(u) - \sum_{u \in \omega^-(x)} f(u) \Big)$$

$$= \sum_{x \in X} \sum_{u \in \omega^+(x)} f(u) - \sum_{x \in X} \sum_{u \in \omega^-(x)} f(u)$$

$$= \sum_{u \in U} f(u) - \sum_{u \in U} f(u) = 0. \ ///$$

COROLLAIRE III.1.4. A tout sommet de poids strictement positif correspond un sommet de poids strictement négatif et inversément.
Démonstration. Cela résulte immédiatement de la proposition précédente. /

PROPOSITION III.1.5. Pour tout sous-ensemble Y de X, on a

$$\sum_{x \in Y} w_f(x) = f(Y,\bar{Y}) - f(\bar{Y},Y) .$$

Démonstration. De fait, la définition III.1.1. implique, pour tout $x \in X$, les égalités

$$w_f(x) = f(x,X) - f(X,x)$$

$$= f(x,Y) + f(x,\bar{Y}) - f(Y,x) - f(\bar{Y},x)$$

$$= f(x,\bar{Y}) - f(\bar{Y},x) + f(x,Y) - f(Y,x)$$

Dans ces conditions, il vient

$$\sum_{x\in Y} w_f(x) = \sum_{x\in Y} \left(f(x,\bar{Y}) - f(\bar{Y},x) \right) + \sum_{x\in Y} \left(f(x,Y) - f(Y,x) \right)$$

$$= f(Y,\bar{Y}) - f(\bar{Y},Y) + \left(\sum_{x\in Y} \sum_{y\in Y} f(x,y) - \sum_{x\in Y} \sum_{y\in Y} f(y,x) \right)$$

$$= f(Y,\bar{Y}) - f(\bar{Y},Y),$$

ce qui suffit. ///

DEFINITION III.1.6. Pour tout sous-ensemble $Y \subset X$,

$$\sum_{x\in Y} w_f(Y)$$

est le f-poids de Y.

CONVENTION III.1.7. Dans la suite de ce travail, la notation

$$w_f(Y)$$

désigne le f-poids de Y. Elle sous-entend que f est une fonction admissible.

DEFINITION III.1.8. Soit f une fonction admissible d'un réseau de transport R.

a) Un f-chemin dans R ou, plus simplement, chemin de flot est un chemin dans R tel que tous les arcs qui le composent donnent à f une valeur strictement positive.

b) Un f-chemin d'élimination ou, plus simplement, chemin d'élimination est un f-chemin dans R dont l'origine possède un poids strictement positif et l'extrémité terminale un poids strictement négatif.

c) Une f-paire d'élimination ou, plus simplement, paire d'élimination est une paire de sommets x,y pour laquelle il existe un f-chemin d'élimination d'extrémités initiale x et terminale y.

CONVENTION III.1.9. Dans la suite de ce travail, les appellations

f-chemin, f-chemin d'élimination, f-paire d'élimination

sous-entendent que f est une fonction admissible.

PROPOSITION III.1.10. Pour tout sommet $x \in X_f^+$, il existe un sommet $y \in X_f^-$ et un f-chemin P d'extrémités initiale x et terminale y tels qu'une réduction non nulle de $w_f(x)$ et de $-w_f(y)$ puisse être obtenue par réduction de la même quantité de flux aux arcs qui composent P, les poids des sommets intermédiaires sur P restant inchangés.

Démonstration. Soit $x \in X_f^+$. Etablissons tout d'abord qu'il existe un sommet $y \in X_f^-$ et un f-chemin d'extrémités initiale x et terminale y. Procédons par l'absurde. Dans ce but, désignons par Y l'ensemble des sommets inaccessibles de x dans le graphe partiel $(X,\{u \in U : f(u) > 0\})$. L'évaluation du f-poids de Y introduit une contradiction. De fait, d'une part, la définition III.1.6. implique l'égalité

$$w_f(Y) = \sum_{y \in Y} w_f(y) = w_f(x) + \sum_{y \in Y \setminus \{x\}} w_f(x)$$

et, dans ces conditions, de l'égalité $Y \cap X_f^- = \phi$, on déduit de suite

$$w_f(Y) > 0.$$

D'autre part, la proposition III.1.5. implique qu'on a

$$w_f(Y) = f(Y,\bar{Y}) - f(\bar{Y},Y)$$

c'est-à-dire

$$w_f(Y) = -f(\bar{Y},Y) \leqslant 0$$

puisque la définition de Y implique l'égalité $f(Y,\bar{Y}) = 0$.

La conclusion est dès lors immédiate. De fait, étant donné un tel chemin P entre $x \in X_f^+$ et $y \in X_f^-$, il suffit de soustraire du flux de chaque arc de P la quantité

$$\min_{u \in P} f(u) > 0$$

pour obtenir une fonction admissible f' telle que

$$0 \leqslant w_{f'}(x) < w_f(x)$$

$$0 \geqslant w_{f'}(y) > w_f(y)$$

et

$$w_{f'}(z) = w_f(z), \quad \forall z \in P \setminus \{x,y\},$$

ce qui suffit. ///

Etablissons à présent une propriété intéressante des coupes.

PROPOSITION III.1.11. Pour tout sous-ensemble Y de X tel que $e \in Y$, $t \notin Y$, $f(Y,\bar{Y}) = c(Y,\bar{Y})$, $f(\bar{Y},Y) = 0$ et $w_f(x) \leqslant 0$, pour tout $x \in Y\backslash\{e\}$ et tout sous-ensemble $Y' \subset Y$ tel que $e \in Y'$, on a $c(Y',\bar{Y}') \geqslant c(Y,\bar{Y})$.

Démonstration. Soient Y et Y' de tels sous-ensembles. Associons au réseau (X,U,c) et à la fonction admissible f, le réseau $R' = (Y \cup \{x_o,x_1\}, U \cap Y\times Y \cup E, c')$ et le flot f' dans R' obtenus comme suit. Tout d'abord, deux sommets auxiliaires $x_o, x_1 \notin X$ sont associés à Y pour former l'ensemble des sommets du nouveau réseau. L'entrée est de nouveau e et la sortie x_1. Ensuite, pour tout $x \in Y$, tous les arcs de la forme (x,y), $y \in \bar{Y}$ sont remplacés par un arc unique (x,x_1) de capacité

$$\sum_{\substack{y\in\bar{Y} \\ (x,y)\in U}} c(x,y)$$

sur lequel circule le flux

$$\sum_{\substack{y\in\bar{Y} \\ (x,y)\in U}} f(x,y) .$$

En outre, pour tout $x \in Y\backslash\{e\}$, l'arc (x,x_o) de capacité $|w_f(x)|$ est introduit; le flux $|w_f(x)|$ y circule. Enfin, l'arc (x_o,x_1) de capacité et de flux

$$\sum_{x\in Y\backslash\{e\}} |w_f(x)|$$

achève la construction.

Comme on le vérifie de suite, la coupe associée à $\{x_o,x_1\}$ dans R' vérifie l'équation

$$c'(\omega^-(\{x_o,x_1\})) = f'[Y\backslash\{x_o,x_1\},\{x_o,x_1\}] - f'[\{x_o,x_1\},Y\backslash\{x_o,x_1\}]$$

et, par suite, vu le théorème du flot maximum et de la coupe minimum I.3.4., est minimum dans R'. Dès lors, la coupe associée à l'ensemble $C_{R'}, Y'$ vérifie l'inégalité

$$c'[\omega^-(C_{R'}, Y')] \geqslant c'[\omega^-(\{x_o,x_1\})]$$

c'est-à-dire, vu ce qui précède,

$$\sum_{\substack{x\in Y' \\ y\in\bar{Y}' \\ (x,y)\in U}} c(x,y) + \sum_{x\in Y'} |w_f(x)| \geqslant \sum_{\substack{x\in Y \\ y\in\bar{Y} \\ (x,y)\in U}} c(x,y) + \sum_{x\in Y} |w_f(x)| .$$

La thèse résulte alors de l'inégalité

$$\sum_{x \in Y'} |w_f(x)| \leqslant \sum_{x \in Y} |w_f(x)| \ . \ ///$$

III.2. L'algorithme ELIMINATION

Introduisons dès à présent l'algorithme ELIMINTATION. Son utilité provient de la remarque suivante. Supposons avoir résolu le problème : obtenir un sous-ensemble X' du réseau (X,U,c) et une fonction admissible f tels que

$$f(X', \bar{X}') = c(X', \bar{X}') \qquad (1)$$

$$f(\bar{X}', X') = 0 \qquad (2)$$

$$w_f(x) \leqslant 0, \quad \forall x \in X' \setminus \{e\}$$

et

$$w_f(y) \geqslant 0, \quad \forall y \in \bar{X}' \setminus \{t\}.$$

Comme $f(\bar{X}', X')$ est nul, les poids négatifs dans X' peuvent être associés aux chemins d'élimination d'origine e et d'extrémité terminale distincte de t et les poids positifs aux chemins d'élimination d'origine distincte de e et d'extrémité terminale t. Supposons que nous soyons capables d'éliminer ces classes de chemin. Comme ces éliminations ne modifient pas les flux des arcs de $X' \times \bar{X}'$, les égalités (1) et (2) restent valables. Par suite, le théorème du flot maximum et de la coupe minimum affirme que la coupe associée à $\bar{X}'$ est minimum et que le flot obtenu est de valeur maximum. Le problème du flot maximum est donc résolu!

L'algorithme ELIMINATION ci-dessous permet d'effectuer ces éliminations. Appliqué à un sous-ensemble Y de X, il détermine puis supprime les paires d'élimination de Y. On procède comme suit. On tente tout d'abord partant d'un sommet de poids strictement positif de tracer un chemin d'élimination ayant ce sommet pour origine, puis on réduit les flux des arcs qui le compose comme exposé dans la démonstration de la proposition III.1.10.. Ce processus est répété jusqu'à ce qu'il n'existe plus de sommet dans Y de poids strictement

positif ou qu'il n'existe plus de chemin d'élimination. Dans ce dernier cas, s'il subsiste des sommets de poids strictement positif, l'algorithme ELIMINATION identifie en plus l'ensemble Z des sommets de Y extrémités terminales d'un chemin de flot d'origine appartennant à Y^+.

ALGORITHME ELIMINATION (Y,Z) III.2.1. Soient f une fonction admissible du réseau de transport (X,U,c) et Y sous-ensemble de X.

```
Z := ∅ ;
lg-chemin := 0 ;
y_aux := y ;
Tant que y+_aux ≠ ∅ effectuer
    si lg-chemin = 0 alors
        choisir x ∈ X+ ;
        chemin(1) := x ;
        lg-chemin := 1 ;
    RECHERCHE_CHEMIN (chemin, lg-chemin, état-chemin) ;
    si état-chemin = bloqué alors
        Z := Z U {chemin (lg-chemin)} ;
        y_aux := y_aux \ {chemin (lg-chemin)} ;
        lg-chemin := lg-chemin - 1 ;
    si état-chemin = terminé alors
        θ := min{W_f(chemin (1)), |W_f(chemin(lg-chemin))|,
                                                        ;
                 min      f(chemin (i), chemin (i+1))}
              1≤i<lg-chemin
        W_f(chemin (1)) := W_f(chemin (1)) - θ ;
        W_f(chemin (lg-chemin)) := W_f(chemin (lg-chemin)) + θ ;
        Pour i variant de 1 à (lg-chemin - 1) répéter
            f (chemin (i), chemin (i+1)) := f (chemin (i),
                                              chemin (i+1)) - θ;
        lg-chemin := 0 .
```

où la procédure RECHERCHE_CHEMIN (chemin,lg-chemin, état-chemin) est

ALGORITHME RECHERCHE_CHEMIN (chemin, lg-chemin, état-chemin) III.2.2. Soient une fonction admissible f du réseau de transport (X,U,c) et un chemin de ce réseau de longueur *lg-chemin* mémorisé dans le tableau *chemin*.

```
état-chemin := normal ;
Tant que (∃y ∈ Y_aux : ℓ (chemin (lg-chemin),y) > 0) et
état-chemin ≠ terminé, répéter
    choisir y ∈ Y_aux tel que ℓ (chemin (lg-chemin),y) > 0 ;
    si W_ℓ(y) < 0
      alors
        état-chemin := terminé
      sinon
        si y ∈ chemin
          alors
            indice := min{i: 1 ⩽ i < lg-chemin
                             et y = chemin (i)} ;
            δ_ℓ := min{ℓ(chemin(i),chemin(i+1):
                       indice ⩽ i < lg-chemin};
            Pour i variant de indice à
            (lg-chemin - 1) effectuer ;
              ℓ(chemin(i),chemin(i+1)) :=
              ℓ(chemin(i),chemin(i+1)) - δ_ℓ ;
            lg-chemin := indice ;
          sinon
            lg-chemin := lg-chemin + 1 ;
            chemin(lg-chemin) := y ;
Si état-chemin ≠ terminé alors
    état-chemin := bloqué .
```

Enonçons tout d'abord la spécification de l'algorithme RECHERCHE_CHEMIN.

THEOREME III.2.3. Soient f une fonction admissible d'un réseau de transport R = (X,U,c) et Y_{aux} un sous-ensemble de X. L'algorithme RECHERCHE_CHEMIN appliqué au f-chemin élémentaire $(x_1, \ldots, x_q)$ de R tel que $W_f(x_i) \geqslant 0$, pour tout $i \in \{1, \ldots, q\}$, converge et fournit, après exécution, un chemin élémentaire de flot $(y_1, \ldots, y_k)$ où $y_1 = x_1$, vérifiant une seule des deux propriétés suivantes.

(i) $W(Y_k) < 0$ et $W(Y_i) \geq 0$, $\forall i \in \{1, \ldots, k-1\}$ et *état-chemin = terminé*

(ii) $W(Y_i) \geq 0$, $\forall i \in \{1, \ldots, k\}$ et $\nexists y \in X: f(y_k, y) > 0$ et *état-chemin = bloqué*

Démonstration. De fait, comme on le vérifie aisément, si f a pour valeur $f^{(init)}$, si le tableau *chemin* mémorise le $f^{(init)}$- chemin

$$(z_1^{(init)}, \ldots, z_{K(init)}^{(init)})$$

tel que

$$w_{f^{(init)}}(z_i^{(init)}) \geq 0$$

pour tout $i \in \{1, \ldots, K(init)\}$ et si la variable *état_chemin* possède la valeur normal, l'exécution du corps de la boucle *tant que* ... affecte à f la valeur $f^{(final)}$ et détermine, dans *chemin*, un $f^{(final)}$-chemin élémentaire de flot

$$(z_1^{(final)}, \ldots, z_{K(final)}^{(final)})$$

tel que

$$z_1^{(final)} = z_1^{(init)},$$

$$w_{f^{(final)}}(z_i^{(final)}) \geq 0, \quad \forall i \in \{1, \ldots, K(final) - 1\},$$

$$w_{f^{(final)}}(z_{K(final)}^{(final)}) < 0 \quad \text{si après exécution on a } etat\_chemin = terminé,$$

$$w_{f^{(final)}}(z_{K(final)}^{(final)}) \geq 0 \quad \text{si après exécution on a } etat\_chemin = normal,$$

et, si après exécution on a *etat_chemin = normal*, tel qu'une des deux assertions suivantes soit vérifiée

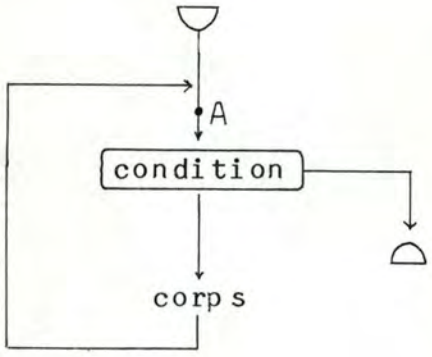
(i) $f^{(init)} = f^{(final)}$ et $K(final) = K(init) + 1$ et

$$(z_1^{(init)}, \ldots, z_{K(init)}^{(init)}) = (z_1^{(final)}, \ldots, z_{K(final)-1}^{(final)}),$$

(ii) $\forall u \in U: f^{(init)}(u) \geq f^{(final)}(u)$ et $\exists u \in U:$

$$f^{(init)}(u) > 0 \text{ et } f^{(final)}(u) = 0.$$

Il s'ensuit que si l'on représente la boucle *Tant que "condition" effectuer "corps"* par l'organigramme



lors de tout passage en A, l'algorithme RECHERCHE_CHEMIN vérifie l'invariant

" la variable *chemin* représente un chemin élémentaire de flot $(z_1, \ldots, z_K)$ tel que

$z_1 = x_1$,

$w_f(z_i) \geqslant 0, \quad \forall i \in \{1, \ldots, K-1\}$

$w_f(z_k) < 0$ si *etat_chemin* = *terminé*

$w_f(z_k) \geqslant 0$ si *etat_chemin* = *normal* .

De plus, la valeur de la variable *lg_chemin* est K. "

Pour conclure, il suffit, dès lors, d'établir que l'algorithme RECHERCHE_CHEMIN converge. Procédons par l'absurde et désignons, pour tout $i \in \mathbb{N}$, par $f^{(i)}$ et $(z_1^{(i)}, \ldots, z_{K(i)}^{(i)})$ respectivement la valeur de f et le chemin élémentaire mémorisé dans *chemin*. Comme le graphe du réseau de transport est fini, pour tout $n \in \mathbb{N}$, on ne peut avoir, pour tout $i \geqslant n$,

$$f^{(i)} = f^{(i+1)} \text{ et } (z_1^{(i)}, \ldots, z_{K(i)}^{(i)}) = (z_1^{(i+1)}, \ldots, z_{K(i+1)-1}^{(i+1)})$$

et $K(i+1) = K(i) + 1$.

Considérons alors la sous-suite des indices $\ell(i)$ pour lesquels cette propriété n'est pas vérifiée. Elle introduit une contradiction. D'une part, elle est infinie car, pour tout i, comme la propriété ci-dessus ne peut être vérifiée pour tout indice $j \geqslant \ell(i)$, il existe nécessairement un indice $\ell(i+1)$. D'autre part, elle est finie car, pour tout i, on a, vu ce qui précède,

$f^{\ell(i)} \geqslant f^{\ell(i)+1}$ et $\exists u \in U : (f^{\ell(i)}(u) > 0$ et $f^{\ell(i)+1}(u) = 0)$

et $\exists y \in Y_{aux} : f(z^{(i)}_{K(i-1)}, y) > 0$. ///

Avant de spécifier l'algorithme ELIMINATION, adoptons tout d'abord la convention suivante.

CONVENTION III.2.4. Dans la suite de ce travail, pour toute partie Y d'un réseau de transport (X,U,c) et toute fonction f admissible de ce réseau, nous notons

$$L_{Y_f^+, Y}$$

ou, plus simplement,

$$L_{Y^+, Y}$$

quand aucun doute n'est possible sur f, l'ensemble des sommets de Y extrémités terminales d'un f-chemin de Y d'origine dans Y_f^+.

Le théorème suivant caractérise l'algorithme ELIMINATION.

THEOREME III.2.4. Soient f une fonction compatible du réseau de transport (X,U,c) à capacités entières et Y un sous-ensemble de X. L'algorithme ELIMINATION(Y,Z) appliqué à Y et f converge et son exécution détermine un ensemble Z et une fonction compatible f' vérifiant les propriétés suivantes.

(i) On a $0 \leqslant w_{f'}(y) \leqslant w_f(y)$, $\forall y \in Y_f^+$.

(ii) On a $w_f(y) \leqslant w_{f'}(y) \leqslant 0$, $\forall y \in Y_f^-$.

(iii) On a $\{y \in Y_f^+$: il ne part aucun f-chemin de y dans $Y\} \subset Z$
$\subset \{y \in Y : w_{f'}(y) \geqslant 0$ et il ne part aucun f'-chemin de y dans $Y\}$.

(iv) On a $L_{Y_{f'}^+, Y} = Z$.

(v) On a $(Y\ Z)^+_{f'} = \phi$. En particulier, $Y \backslash Z$ ne contient aucune f'-paire d'élimination.

(vi) Si le sommet $y \in Y$ possède un demi-degré intérieur nul et est tel que $w_f(y) < 0$, alors tout arc $u \in \omega^+(y)$ vérifie l'égalité $f'(u) = f(u)$ et on a $w_{f'}(y) = w_f(y)$.

(vii) Si le sommet $y \in Y$ possède un demi-degré extérieur nul et est tel que $w_f(y) > 0$, alors tout arc $u \in \omega^-(y)$ vérifie l'égalité $f'(u) = f(u)$ et on a $w_{f'}(y) = w_f(y)$.

(viii) Pour tout arc $u \in U \cap C_{X\times X}\ Y\times Y$, on a $f'(u) = f(u)$.

(ix) Pour tout sommet $x \in \bar{Y}$, on a $w_f(x) = w_{f'}(x)$.

Démonstration. L'algorithme ELIMINATION(Y,Z) converge car à chaque itération de la boucle *Tant que* ... ou bien le nombre d'éléments de Y_{aux} diminue d'une unité et la somme (fonction de la fonction compatible courante)

$$\sum_{y\in Y^+} w(y) + \sum_{y\in Y^-} |w(y)|$$

ne change pas ou bien cette somme diminue d'une unité au moins et, alors, le cardinal de Y_{aux} n'est pas modifié. Pour conclure, établissons les propriétés (i) à (ix). Dans ce but, remarquons, tout d'abord, qu'en représentant la boucle *Tant que* ... comme dans la démonstration précédente, l'invariant

" $Y_{aux} \cup Z = Y$ "

est vérifié lors de tout passage en A.

Cela étant, les propriétés (i), (ii), (v), (viii) et (ix) sont immédiates.

Prouvons la propriété (iii). D'une part, de la forme de l'algorithme, on déduit de suite que le poids d'un sommet introduit dans l'ensemble Z ne varie plus et par suite qu'on a

$$Z \subset \{y \in Y : w_{f'}(y) \geqslant 0 \text{ et il ne part aucun } f'\text{-chemin de de y dans } Y\}.$$

D'autre part, on a

$$\{y \in Y_f^+ : \text{il ne part aucun } f\text{-chemin de } y \text{ dans } Y\} \subset Z.$$

Sinon, il existe un sommet $y \in Y \setminus Z$ tel que $w_f(y) > 0$ duquel ne part aucun f-chemin dans Y. Ceci est absurde car, après exécution, l'ensemble $(Y\setminus Z)^+_{f'}$ est vide.

Démontrons la propriété (iv). L'inclusion $Z \subset L_{Y^+_{f'},Y}$ est est évidente car tout élément de Z est l'extrémité terminale d'une chaîne f'-augmentante dont l'origine x vérifie l'inégalité $w_{f'}(x) > 0$. Pour conclure, il reste à prouver qu'on a $L_{Y^+_{f'},Y} \subset Z$.

Procédons par l'absurde. Il existe alors un sommet $z \notin Z$ extrémité terminale d'un f'-chemin dans Y d'origine appartenant à $Y^+_{f'}$, soit $(x, x_1, \ldots, x_q, z)$. Or le mode d'insertion dans Z implique la non appartenance de $x, x_1, \ldots, x_q$ dans Z et par suite leur appartenance à $Y \setminus Z$, ce qui est absurde car l'ensemble $(Y \setminus Z)^+_{f'}$ est vide.

Prouvons pour terminer les propriétés (vi) et (vii). La deuxième partie de la propriété (vi) résulte de suite du fait que toute modification des poids se fait aux extrémités d'un chemin d'élimination et du fait qu'un tel chemin ne peut avoir pour origine un sommet de poids strictement négatif ni pour extrémité terminale un sommet de demi-degré intérieur nul. Dans ces conditions, comme seuls les flux des arcs des chemins d'élimination sont modifiés et puisqu'un arc $u \in \omega^+(y)$ ne peut appartenir à un tel chemin, on a, pour tout $u \in \omega^+(y)$, $f'(u) = f(u)$. Enfin, la propriété (vii) s'établit de même. ///

III.3. La méthode de redistribution de flux

Décrivons à présent l'algorithme de redistribution de flux de Kinariwala et Rao. Il s'agit d'un algorithme itératif dont chaque itération se compose d'une suite de phases de déplacement de flux suivie d'une phase d'élimination. Il s'applique à toute fonction admissible. Toutefois, avant d'amorcer les itérations, l'exécution d'une phase d'initialisation est requise.

la phase d'initialisation

Lors de celle-ci, les paires d'élimination de $X \setminus \{e\}$ sont tout d'abord supprimées de telle sorte qu'on a, pour tout $x \in X \setminus \{e,t\}$, $w(x) \leqslant 0$; l'égalité $f(t,X) = 0$ permet alors d'interpréter la quantité $w(x)$ comme l'excès de flux d'entrée en x en provenance de e. Ensuite, pour faciliter les calculs, les poids fictifs $-M$ et M, où M désigne une constante arbitrairement grande, sont respectivement affectés aux sommets e et t. Ils assurent les inégalités $w(e) < 0$ et $w(t) > 0$ et,

par suite, aucune exécution de la procédure ELIMINATION ne peut réduire les flux des arcs de $\omega^+(e)$ et $\omega^-(t)$. Enfin, un ensemble auxiliaire X' est créé et initialisé au vide.

Le but des itérations qui suivent la phase d'initialisation est de faire évoluer la fonction admissible f ainsi modifiée et l'ensemble X' de telle sorte qu'on ait, après exécution de celles-ci,

$$e \in X', \quad t \notin X',$$

$$f(X', \bar{X}') = c(X', \bar{X}'),$$

$$f(\bar{X}', X') = 0,$$

$$w_f(x) \leqslant 0, \quad \forall x \in X' \setminus \{e\},$$

$$w_f(x) \geqslant 0, \quad \forall x \in \bar{X}' \setminus \{t\},$$

et

$$0 \leqslant f(u) \leqslant c(u), \quad \forall u \in U,$$

ce qui résout le problème mentionné au début du paragraphe III.2.. Elles reposent sur le déplacement des excès de flux (cités ci-dessus) en t.

Analysons tout d'abord les phases de déplacement de flux.

les phases de déplacement de flux

Avant toute exécution de cette suite de phases, l'ensemble $\bar{X}'^-$ est non vide et l'on a, pour tout $x \in \bar{X}'^- \setminus \{t\}$, $w(x) \leqslant 0$. Ceci est assuré à la première itération par l'exécution de la phase d'initialisation et, pour toute itération ultérieure, par l'itération précédente.

Cela étant, lors de toute phase r, le complémentaire de l'ensemble X' courant est partitionné en deux sous-ensembles X_r et Y_r dont le dernier cité contient les sommets contre lesquels l'excès de flux d'entrée disponible dans $X_r'^-$ est à déplacer. De plus, cette partition vérifie les relations

$$X_r^+ = \phi,$$

$$f(X', Y_r) = c(X', Y_r),$$

$$f(X_r, Y_r) \leqslant c(X_r, Y_r),$$

et

$$f(Y_r, X' \cup X_r) = 0 .$$

Elle donne, en outre, lieu à une des deux situations suivantes. Dans la première, on a $X_r^- = \phi$ ou $f(X_r, Y_r) = c(X_r, Y_r)$. Dans ces conditions, aucun déplacement de flux n'est possible, l'ensemble X_r est ajouté à X' et la phase d'élimination, décrite ci-dessous, est exécutée. Dans la seconde, aucune des deux égalités précédentes n'est vérifiée. Les arcs de $X_r x Y_r$ sont alors saturés, augmentant ou diminuant ainsi respectivement les poids des sommets de X_r et Y_r. Des paires d'élimination peuvent maintenant exister entre X_r^+ et X_r^-. Leur suppression réduit les poids des sommets de X_r^+ et l'excès de flux d'entrée disponible dans X_r^-. La quantité correspondante de flux est ensuite déplacée contre Y_r et l'ensemble

$$L_r = L_{X_r^+, X_r}$$

est retiré de X_r pour être ajouté à Y_r. Une nouvelle phase, numérotée r+1, commence.

Il convient ici de clarifier le concept de déplacement de flux.

le concept de déplacement de flux

La suppression des pairesd'élimination représente une réduction d'une certaine quantité de flux dans $X_r^+ x X_r^-$. Ceci n'est possible que parce qu'un poids positif fut créé pendant le processus de saturation (représentant une augmentation de flux dans $X_r x Y_r$). Dès lors, une certaine quantité de flux contre X_r^- a été réduite et, au moins, une quantité correspondante de flux contre Y_r a été accrue. Nous interprétons ce changement comme un déplacement de flux puisque le flux contre X_r^- a été déplacé contre Y_r. Dans la suite, nous utiliserons ce terme "déplacement" pour désigner un moyen de réduction de flux entre les paires d'élimination, celle-ci étant compensée par une augmentation correspondante de flux dans une autre direction.

Décrivons à présent la phase d'élimination

la phase d'élimination

Arrivé en ce point, un nombre fini h de phases de déplacement de flux ont été effectuées et on a

$$Y_{h+1} = \{t\} \cup L_1 \cup \ldots \cup L_h,$$

chaque ensemble L_i $(1 \leqslant i \leqslant h)$ pouvant contenir des sommets de poids positifs ou négatifs créés durant l'exécution de celles-ci.

Dans la phase d'élimination, les arcs saturés durant les phases de déplacement de flux sont sélectionnés dans l'ordre inverse de l'ordre de saturation et leur flux est réduit. Ce processus se répète jusqu'à ce que tous les poids positifs soient éliminés.

Nous prouverons dans la suite les inclusions

$$(L_i \times L_{i-1}) \cap U \subset (X_i \times Y_i) \cap U \quad (i = 1, \ldots, h)$$

et démontrerons que l'élimination des poids positifs dans L_i^+ $(i = 1, \ldots, h)$ nécessite une réduction de flux, initialement augmenté dans $(L_i \times L_{i-1}) \cap U$. Toutefois, avant que le flux dans $(L_i \times L_{i-1}) \cap U$ $(i = 1, \ldots, h)$ ne soit réduit, nous prendrons soin de supprimer toutes les paires d'élimination existant dans L_i. Notons qu'une telle suppression entraîne un déplacement successif de flux à travers les ensembles $L_i, L_{i-1}, \ldots, \{t\}$.

Enfin, la phase d'élimination se termine par la réinitialisation des flux dans $\bar{X}'$ de telle sorte qu'on ait $w(x) \leqslant 0$, pour tout $x \in \bar{X}' \setminus \{t\}$.

L'algorithme suivant formalise ce qui précède.

ALGORITHME DE REDISTRIBUTION DE FLUX III.3.1. Soient f une fonction admissible d'un réseau de transport (X,U,c) et M une constante arbitrairement grande.

```
Pour tout x ∈ X\{e} effectuer
  w(x) := ℓ(x,X) - ℓ(X,x);
w(e) := -M;
ELIMINATION(X,Z);
w(t) := M;
X' := ϕ;
```

```
Tant que X̄' ≠ φ effectuer
  | X_0 := X̄' ;
  | y_0 := φ ;
  | L_0 := {t};
  | Pour tout u ∈ U effectuer δℓ(u) := 0;
  | X_1 := X̄' \ {t};
  | y_1 := {t};
  | r := 1;
  | Tant que X_r^- ≠ φ et ℓ(X_r, y_r) < c(X_r, y_r) répéter
  |   | Pour tout (x,y) ∈ (X_r x y_r) ∩ U effectuer
  |   |   | δℓ(x,y) := c(x,y) - ℓ(x,y);
  |   |   | si δℓ(x,y) > 0 alors
  |   |   |   | ℓ(x,y) := c(x,y);
  |   |   |   | w(x) := w(x) + δℓ(x,y);
  |   |   |   | w(y) := w(y) - δℓ(x,y);
  |   | ELIMINATION(X_r, L_r);
  |   | r := r+1;
  |   | X_r := X_{r-1} \ L_{r-1};
  |   | y_r := y_{r-1} ∪ L_{r-1};
  | Si X_r^- ≠ φ alors X' := X' ∪ X_r ;
  | r := r - 1;
  | Tant que r ≠ 0 effectuer
  |   | ELIMINATION(L_r, Z);
  |   | Tant que L_r^+ ≠ φ effectuer
  |   |   | choisir un arc (x,y) ∈ (L_r^+ x L_{r-1}) ∩ U tel que
  |   |   |
  |   |   |   θ = min{δℓ(x,y), w(x)} > 0;
  |   |   | ℓ(x,y) := ℓ(x,y) - θ;
  |   |   | w(x) := w(x) - θ;
  |   |   | w(y) := w(y) + θ;
  |   |   | δℓ(x,y) := δℓ(x,y) - θ;
  |   | r := r - 1;
w(e) := c(X',X̄') + |w(X'^-)|;
w(t) := - c(X',X̄');
ELIMINATION(X',Z).
```

Etablissons à présent que l'algorithme de redistribution de flux résout le problème du flot maximum. Dans ce but, démontrons tout d'abord les deux propositions préliminaires suivantes.

PROPOSITION III.3.2. Soient f une fonction admissible du réseau de transport R = (X,U,c), w une fonction de R et M et M' deux constantes positives arbitrairement grandes telles que

$$w(e) = -M,$$
$$w(t) = M',$$

et

$$w(x) = f(x,X) - f(X,x) \leqslant 0, \quad \forall x \in R.$$

Soit, en outre, X' un sous-ensemble de X tel que $f(\bar{X}',X') = 0$ et $f(X',\bar{X}') = c(X',\bar{X}')$. Le programme

```
X_0 := X̄';
y_0 := φ;
L_0 := {t};
Pour tout u ∈ U effectuer δℓ(u) := 0;
X_1 := X̄' \ {t};
y_1 := {t};
r := 1;
Tant que X_r^- ≠ φ et ℓ(X_r,y_r) < c(X_r,y_r) répéter
    Pour tout (x,y) ∈ (X_r x y_r) ∩ U effectuer
        δℓ(x,y) := c(x,y) - ℓ(x,y);
        si δℓ(x,y) > 0 alors
            ℓ(x,y) := c(x,y);
            w(x) := w(x) + δℓ(x,y);
            w(y) := w(y) - δℓ(x,y);
    ELIMINATION(X_r,L_r);
    r := r+1;
    X_r := X_{r-1} \ L_{r-1};
    y_r := y_{r-1} ∪ L_{r-1};
Si X_r^- ≠ φ alors X' := X' ∪ X_r.
```

appliqué à f, w, X' converge. De plus, si X'_{init}, f_{init} désignent respectivement la valeur de X' et f avant exécution et si h, X'_{final}, f_{final}, w_{final}, f_{final} désignent respectivement la valeur de r, X', f, w, f après exécution, on a

1) $X_h^- \neq \phi \Longrightarrow f_{final}(X_h, Y_h) = c(X_h, Y_h)$ et $X'_{final} = X'_{init} \cup X_h$.

2) $X_h^- = \phi \Longrightarrow X'_{final} = X'_{init}$.

3) $\forall r \in \{1, \ldots, h\} : f_{final}(Y_r, X_r) = 0$ et $X_r \cup Y_r = \bar{X}'_{init}$ et

$$X_r \cap Y_r = \phi.$$

4) $f_{final}(X'_{final}, \bar{X}'_{final}) = c(X'_{final}, \bar{X}'_{final})$.

5) $f_{final}(\bar{X}'_{final}, X'_{final}) = 0$.

6) $\forall x \in X'_{final} : w_{final}(x) \leqslant 0$.

7) $X'_{init} = \phi \Longrightarrow X'_{final} \ni e$

8) $\forall r \in \{1, \ldots, h-1\}$ $\forall (x,y) \in (X_r \times Y_r) \cap U : \delta f^r(x,y) > 0 \Rightarrow y \in L_{r-1}$

où δf^r $(1 \leqslant r < h)$ désigne la valeur de δf dans la r^{eme} itération de la boucle *Tant que ...* .

9) $w_{final}(e) = -M''$, où M'' est une constante arbitrairement grande.

10) $w_{final}(t) = M'''$, où M''' est une constante arbitrairement grande.

11) $f_{init}(e,X) = f_{final}(e,X)$.

12) $f_{init}(X,t) = f_{final}(X,t)$.

13) $X_{h,f_{final}}^+ = \phi$.

14) $Y_h = L_{h-1} \cup \ldots \cup L_1 \cup \{t\}$.

15) $\forall i,j \in \{1, \ldots, h-1\} : i \neq j \Longrightarrow L_i \cap L_j = \phi$.

16) $\forall r \in \{1, \ldots, h-1\}$ $\forall x \in L_r$ $\exists w^*(x) \leqslant 0$:

$$w_{final}(x) = w^*(x) + f_{final}(x, L_{r-1}) - \theta - f_{final}(X_{r+1}, x)$$

où θ est le flux total réduit le long des chemins d'élimination de x à X_r^- par l'exécution de ELIMINATION(X_r, L_r).

17) $\forall r \in \{1, \ldots, h-1\} \quad \forall (x,y) \in (L_r \times L_{r-1}) \cap U :$

$$\delta f^r(x,y) = \delta f_{final}(x,y).$$

18) $\forall r \in \{0, \ldots, h-1\} \quad \forall (x,y) \in (X_{r+1} \times L_r) \cap U :$

$$\delta f_{final}(x,y) \leqslant \delta f^{r+1}(x,y).$$

19) $\forall r \in \{1, \ldots, h-1\} \quad \forall (x,y) \in (L_r \times L_{r-1}) \cap U :$

$$\delta f_{final}(x,y) = 0 \quad \underline{ou} \quad f_{final}(x,y) = c(x,y).$$

20) f_{final} est une fonction admissible.

21) $\forall x \in X \setminus \{e,t\} : w_{final}(x) = f_{final}(x,X) - f_{final}(X,x).$

Démonstration. Etablissons tout d'abord la convergence du programme. Deux cas se présentent. Dans le premier, pour toute itération r, l'ensemble L_r est non vide. Dans ces conditions, comme l'ensemble $X_o = \bar{X}'$ est fini et puisque, pour tout r, on a $X_{r+1} = X_r \setminus L_r$, il existe nécessairement un indice h pour lequel on a $X_h = \phi$, ce qui suffit. Dans le second, il existe une itération r pour laquelle l'ensemble L_r est vide. Dans ce cas, il vient

$$X_{r+1} = X_r \setminus L_r = X_r ,$$

$$Y_{r+1} = Y_r \cup L_r = Y_r$$

et, par conséquent,

$$f(X_{r+1},Y_{r+1}) = f(X_r,Y_r)$$

et

$$c(X_{r+1},Y_{r+1}) = c(X_r,Y_r) .$$

Dès lors, il reste à établir, pour conclure, l'égalité $f(X_r,Y_r) = c(X_r,Y_r)$. Or ceci résulte de suite du fait que les flux des arcs d'origine dans X_r et d'extrémité terminale dans Y_r n'ont pu être modifiés par l'exécution de la procédure ELIMINATION(X_r,L_r).

Etablissons à présent les propriétés 1) à 21).

1) et 2) sont immédiats.

3) Raisonnons par récurrence. D'une part, les relations

$$X_1 = \{t\},\ Y_1 = \bar{X}'_{init} \setminus \{t\},$$

$$0 \leqslant f_{final}(Y_1, X_1) \leqslant f_{init}(\{t\}, \bar{X}' \setminus \{t\}) = 0$$

établissent la propriété pour $r = 1$. D'autre part, cette propriété est vérifiée pour $r = r_o \in \{1, \ldots, h\}$ si elle l'est pour tout $r \in \{1, \ldots, r_o - 1\}$. De fait, on a, bien sûr,

$$X_{r_o} \cup Y_{r_o} = \bar{X}'_{init},$$

$$X_{r_o} \cap Y_{r_o} = \phi.$$

De plus, il vient, en désignant, pour tout $r \in \{1, \ldots, h-1\}$, par $f^{(r)}$ la valeur de f à la fin de la r^{eme} itération,

$$0 \leqslant f_{final}(Y_{r_o}, X_{r_o})$$

$$\leqslant f^{(r_o-1)}(Y_{r_o-1} \cup L_{r_o-1}, X_{r_o-1} \setminus L_{r_o-1})$$

$$= f^{(r_o-1)}(Y_{r_o-1}, X_{r_o-1} \setminus L_{r_o-1}) + f^{(r_o-1)}(L_{r_o-1}, X_{r_o-1} \setminus L_{r_o-1}).$$

Dès lors, puisque l'hypothèse de récurrence implique

$$0 = f_{final}(Y_{r_o-1}, X_{r_o-1}) = f^{(r_o-1)}(Y_{r_o-1}, X_{r_o-1}),$$

il reste pour conclure à prouver que le deuxième terme de la somme précédente est nul. Sinon, il existe un arc $(x,y) \in (L_{r_o-1}, X_{r_o-1} \setminus L_{r_o-1})$ tel que $f(x,y) > 0$. Ceci introduit une contradiction car, comme le théorème III.2.5. implique l'égalité

$$L_{r_o-1} = L_{X^+_{r_o-1}, f_{r_o-1}, X_{r_o-1}},$$

le sommet y appartient à L_{r_o-1}.

4) Comme la propriété s'établit de suite au cas où $X_h^- = \phi$, il suffit de l'établir dans le cas où $X_h^- \neq \phi$. Dans ce cas, il vient, vu la propriété 2),

$$X'_{final} = X'_{init} \cup X_h$$

et, par suite, vu la propriété précédente,

$$f_{final}(X'_{final}, \overline{X'_{final}}) = f_{final}(X'_{init} \cup X_h, \overline{X'_{init} \cup X_h})$$
$$= f_{final}(X'_{init}, \overline{X'_{init}} \cap \overline{X_h}) +$$
$$f_{final}(X_h, \overline{X'_{init}} \cap \overline{X_h})$$
$$= f_{final}(X'_{init}, \overline{X'_{init}} \cap \overline{X_h}) + c(X_h, Y_h)$$
$$= f_{final}(X'_{init}, \overline{X'_{init}} \cap \overline{X_h}) +$$
$$c(X_h, \overline{X'_{init} \cup X_h}) .$$

Pour conclure, il reste à prouver qu'on a

$$f_{final}(X'_{init}, \overline{X'_{init}} \cap \overline{X_h}) = c(X'_{init}, \overline{X'_{init}} \cap \overline{X_h}).$$

De fait, de l'égalité

$$f_{init}(X'_{init}, \overline{X'_{init}}) = c(X'_{init}, \overline{X'_{init}})$$

on déduit de suite que toute partie $A \subset \overline{X'_{init}}$ vérifie les égalités

$$f_{final}(X'_{init}, A) = f_{init}(X'_{init}, A)$$
$$= c(X'_{init}, A)$$

ce qui suffit.

5) Comme la propriété est immédiate quand $X_h^- = \phi$, on peut, bien sûr, supposer avoir $X'_{final} = X'_{init} \cup X_h$. Dès lors, il vient, vu la propriété 3),

$$f_{final}(\overline{X'_{final}}, X'_{final}) = f_{final}(\overline{X'_{init} \cup X_h}, X'_{init} \cup X_h)$$
$$\leqslant f_{final}(\overline{X'_{init}}, X'_{init}) +$$
$$f_{final}(\overline{X'_{init}} \cap \overline{X_h}, X_h)$$
$$\leqslant f_{init}(\overline{X'_{init}}, X'_{init}) + f_{final}(Y_h, X_h)$$
$$= 0 + 0 .$$

D'où la conclusion.

6) De fait, les valeurs de w et f ne sont modifiées que dans $\overline{X'_{init}}$. Dès lors, si l'ensemble X_h^- est vide, la propriété est immédiate. Dans le cas contraire, il suffit, alors, pour conclure de

prouver qu'on a, pour tout $x \in X_h$, $w_{final}(x) \leqslant 0$. Or du théorème III.2.4., on déduit de suite, en désignant par w' la valeur de w avant exécution de ELIMINATION(X_{h-1},L_{h-1})

$$L_{h-1} = L_{X_{h-1}^+, X_h}$$

$$\supset \{x \in X_{h-1} : w'(x) > 0\}$$

$$\supset \{x \in X_{h-1} : w_{final}(x) > 0\}$$

ce qui suffit car X_h vaut $X_{h-1} \setminus L_{h-1}$.

7) De fait, comme le demi-degré intérieur de e est nul et puisqu'on a $w(e) = -M$, où M désigne une constante positive arbitrairement grande, l'entrée du réseau ne peut être l'extrémité d'un chemin d'élimination. Dès lors, il vient $e \notin L_r$ pour tout $r \in \{1, \ldots, h-1\}$ et puisque e appartient à X, X_r^- est non vide et e appartient à X_r pour tout $r \in \{1, \ldots, h\}$. Dans ces conditions, la propriété 1) implique l'égalité $X'_{final} = X'_{init} \cup X_h$ et par conséquent la thèse.

8) Procédons par l'absurde. Il existe alors $r \in \{1, \ldots, h-1\}$, $(x,y) \in (X_r, Y_r \setminus L_{r-1})$ tels que $\delta f^r(x,y) > 0$. Dès lors, les égalités $X_r = X_{r-1} \setminus L_{r-1}$ et $Y_r = Y_{r-1} \cup L_{r-1}$ impliquent que l'arc (x,y) appartient à (X_{r-1}, Y_{r-1}). Ceci introduit une contradiction car, comme l'exécution de ELIMINATION(X_{r-1},L_{r-1}) ne peut modifier la valeur du flux de (x,y), il vient alors

$$f^r(x,y) = c(x,y)$$

et, par suite,

$$\delta f^r(x,y) = c(x,y) - f^r(x,y) = 0.$$

9) et 10) résultent de la valeur de w(e) et w(t) et du théorème III.2.5..

11) résulte de la valeur de w(e), du fait que $\omega^-(e) = \phi$ et du théorème II.2.5..

12) résulte de la valeur de w(t), du fait que $\omega^+(t) = \phi$ et du théorème III.2.5..

13) De fait, vu le théorème III.2.5. on a après exécution de ELIMINATION(X_{h-1},L_{h-1})

$$(X_{h-1} \setminus L_{h-1})^{+} = \phi.$$

14) et 15) En effet, on établit aisément par récurrence les égalités

$$Y_r = L_{r-1} \cup \ldots \cup \{t\} \ , \quad \forall r \in \{1, \ldots, h\},$$

$$L_i \cap L_j = \phi, \quad \forall i,j \in \{1, \ldots, h-1\}, \ i \neq j \ .$$

16) Soient $r \in \{1, \ldots, h-1\}$ et $x \ L_r$. Comme on le vérifie de suite, on a, au début de la r^{eme} itération, $w(x) \leqslant 0$. Appelons $w^*(x)$ cette valeur de $w(x)$. L'exécution de l'instruction itérative *Pour tout* $(x,y) \in (X_r \times Y_r) \cap U$ ... affecte alors à $w(x)$ la valeur

$$w(x) = w^*(x) + \delta f^r(x,Y_r)$$

c'est-à-dire, vu la propriété 8), la valeur

$$w(x) = w^*(x) + \delta f^r(x,L_{r-1}) \ .$$

Ensuite, l'appartenance de x à L_r et le théorème III.2.5. impliquent, après exécution de la procédure ELIMINATION(X_r,L_r), l'inégalité

$$w(x) \geqslant 0$$

ce qui ne se peut que si on a

$$w^*(x) + \delta f^r(x,L_{r-1}) \geqslant 0 \ .$$

Par conséquent, en utilisant des notations connues, la valeur de $w(x)$ devient

$$w^*(x) + \delta f^r(x,L_{r-1}) - \theta.$$

Enfin, puisqu'on a $x \in Y_{r+1}$, l'itération suivante affecte à w la valeur

$$w^*(x) + \delta f^r(x,L_{r-1}) - \theta - \delta f^{r+1}(X_{r+1},x) \ .$$

La thèse résulte alors de la propriété 8).

17) De fait, comme on a, pour tout $i \in \mathbb{N}_o$ et tout $r \in \{1, \ldots, h-1-i\}$,

$$X_{r+i} \cap L_r = \phi,$$

tout arc $(x,y) \in (L_r,L_{r-1})$ ne peut appartenir à (X_{r+i},Y_{r+i}) ($i \in \mathbb{N}_o$, $r \in \{1, \ldots, h-1-i\}$).

18) Soient $r \in \{0, \ldots, h-1\}$ et (x,y) un arc de $X_{r+1} \times L_r$. D'une part, si le sommet x appartient à L_{r+1}, l'arc (x,y) appartient à $L_{r+1} \times L_r$ et, vu la propriété précédente, on a

$$\delta f^{r+1}(x,y) = \delta f_{final}(x,y) .$$

D'autre part, si x appartient à $X_{r+1} \setminus L_{r+1}$, la seule affectation que le programme puisse imposer à $\delta f(x,y)$ après la $(r+1)^{eme}$ itération est $\delta f(x,y) := 0$. De fait, pour tout arc $(x,y) \in X_{r+i} \times Y_{r+i}$, l'inégalité $\delta f^{r+i}(x,y) > 0$ $(i \in \mathbb{N})$ et la propriété 8) impliquent l'appartenance de y à L_{r+i-1}. La disjonction des ensembles L_i $(i = 1, \ldots, h-1)$ permet de conclure.

19) se démontre de suite par récurrence en tenant compte de la propriété précédente.

20) et 21) sont immédiats. ///

PROPOSITION III.3.3. Le programme

```
r := r-1;
Tant que r ≠ 0 effectuer
  | ELIMINATION(L_r, Z);
  | Tant que L_r^+ ≠ φ effectuer
  |   | choisir un arc (x,y) ∈ (L_r^+ × L_{r-1}) ∩ U tel que
  |   |
  |   |   θ = min{δℓ(x,y), w(x)} > 0;
  |   | ℓ(x,y) := ℓ(x,y) - θ;
  |   | w(x) := w(x) - θ;
  |   | w(y) := w(y) + θ;
  |   | δℓ(x,y) := δℓ(x,y) - θ;
  | r := r-1.
```

appliqué à $r = h$ et aux êtres mathématiques vérifiant les assertions 1) à 21) de la proposition III.3.2. converge et modifie uniquement δf_{final}, f_{final}, w_{final} en δf_{fin}, f_{fin}, w_{fin} de telle sorte qu'on ait après exécution

1) $w_{fin}(x) \leqslant 0 \quad \forall\, x \in X \setminus \{t\}$,

2) $w_{fin}(e) = -M''''$,

3) $w_{fin}(t) = M''''',$

4) $w_{fin}(x) = f_{fin}(x,X) - f_{fin}(X,x), \quad \forall x \in X \setminus \{e,t\},$

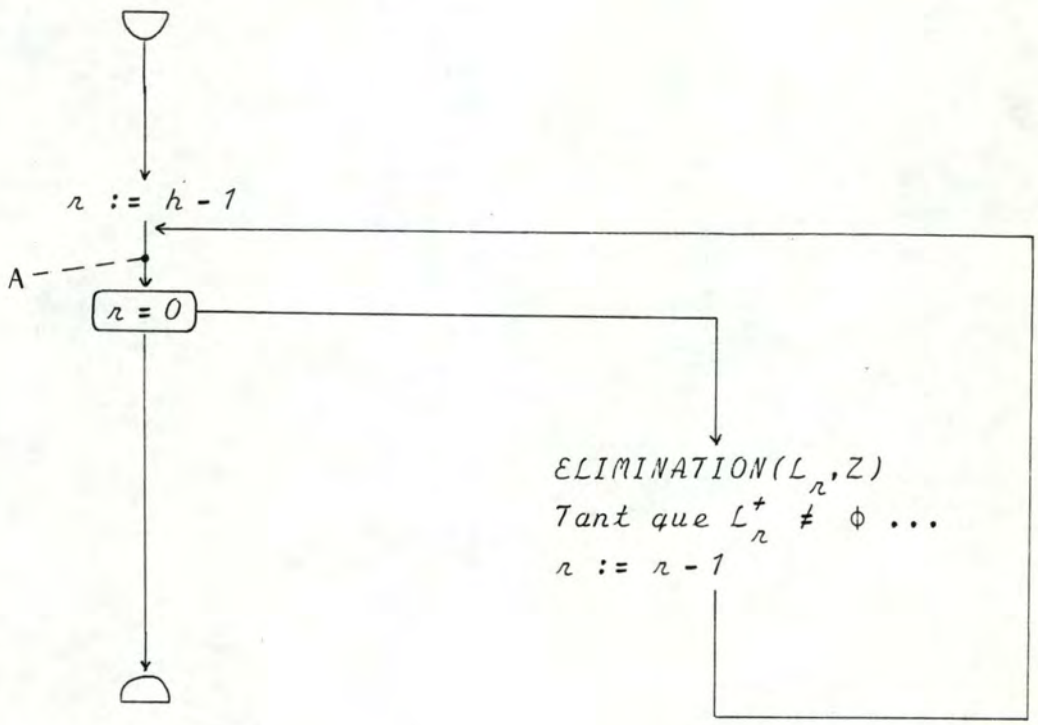
5) $f_{fin}(\overline{X'_{final}}, X'_{final}) = 0,$

6) $f_{fin}(X'_{final}, X'_{final}) = c(X'_{final}, \overline{X'_{final}}),$

7) $0 \leqslant f_{fin}(u) \leqslant c(u), \quad \forall u \in U,$

où M'''' et M''''' désignent deux constantes arbitrairement grandes.

Démonstration. Etablissons tout d'abord par récurrence qu'en représentant ce programme par l'organigramme



l'invariant

$$I_A \equiv " \quad \forall j \in \{r+1, \ldots, h-1\}, \ \forall x \in L_j : w(x) \leqslant 0 \qquad (1)$$

$$\forall x \in L_r : w(x) \leqslant \delta f(x, L_{r-1}) \qquad (2)$$

$$\forall j \in \{1, \ldots, r-1\}, \ \forall x \in L_j \ \exists\, w^*(x) \leqslant 0, \ \theta \geqslant 0 :$$

$$w(x) \leqslant w^*(x) + \delta f_{final}(x, L_{j-1}) - \theta - \delta f_{final}(X_{j+1}, x) \qquad (3)$$

$$\forall x \in X' : w(x) = w_{final}(x) \qquad (4)$$

$$w(e) = -M, \tag{5}$$

$$w(t) = -M', \tag{6}$$

$$\forall x \in X \setminus \{e,t\} : w(x) = f(x,X) - f(X,x), \tag{7}$$

$$f(\overline{X'_{final}}, X'_{final}) = 0 \tag{8}$$

$$f(X'_{final}, \overline{X'_{final}}) = c(X'_{final}, \overline{X'_{final}}) \tag{9}$$

$$\forall u \in U : 0 \leqslant f(u) \leqslant c(u) \tag{10}$$

où M et M' désignent deux constantes positives arbitrairement grandes "

est vérifié lors de tout passage en A. D'une part, lors du premier passage en A on a r = h-1 et comme la seule instruction exécutée est $r := r - 1$, les relations (1) à (10) se déduisent de suite de l'hypothèse. D'autre part, si l'invariant I_A est vérifié lors d'un passage en A et s'il existe un passage successif, l'invariant est encore vérifié lors de ce dernier passage. De fait, soit r_o la valeur de r lors du premier passage. Au second, on a alors $r = r_o - 1$. Dans ces conditions, seules les relations

$$w(x) \leqslant 0, \quad \forall x \in L_{r_o}$$

et

$$w(x) \leqslant \delta f(x, L_{r_o-2}), \quad \forall x \in L_{r_o-1}$$

ne sont pas évidentes. Démontrons les. Par hypothèse de récurrence, il vient

$$w(x) \leqslant w^*(x) + \delta f(x, L_{r_o-2}) - \theta - \delta f(X_{r_o-1}, x), \quad \forall x \in L_{r_o-1}$$

$$w(x) \leqslant \delta f(x, L_{r_o-1}), \quad \forall x \in L_{r_o}$$

et dès lors, vu le théorème III.2.5., après exécution de ELIMINATION(L_{r_o},Z),

$$w(x) \leqslant \delta f(x, L_{r_o-1}), \quad \forall x \in L_{r_o} .$$

Dans ces conditions, la boucle *Tant que* $\mathcal{L}_r^+ \neq \phi$ ... converge et on a, après exécution,

$$w(x) \leqslant 0, \quad \forall x \in L_{r_o} .$$

En outre, comme, pour tout $x \in L_{r_o-1}$, les valeurs de $\delta f(x, L_{r_o-2})$ et $\delta f(L_{r_o-1}, x)$ ne changent pas lors de cette exécution, il vient encore, pour tout $x \in L_{r_o-1}$,

$$w(x) \leqslant w^*(x) + \delta f_{final}(x, L_{r_o-2}) - \theta - \delta f_{final}(X_{r_o-1}, x) + \delta f(L_{r_o-1}, x)$$

$$\leqslant \delta f(x, L_{r_o-2})$$

ce qui suffit.

De plus, la convergence du programme et les relations 2) à 7) résultent de suite de ce qui précède. Prouvons pour conclure la relation 1). D'une part, si on a $X'_{final} = X'_{init}$, il vient $X'_{final} = X_h \cup Y_h$ et par suite, vu l'invariant I_A particularisé à $r = 0$ et les égalités $X_h^+ =$, $Y_h = L_{h-1} \cup \ldots \cup L_1 \cup \{t\}$, les relations

$$w(x) \leqslant 0, \quad \forall x \in X'_{final} \setminus \{t\} .$$

D'autre part, si l'ensemble X'_{final} vaut $X'_{init} \cup X_h$, son complémentaire vaut Y_h et ce qui précède établit les relations

$$w(x) \leqslant 0, \quad \forall x \in X'_{final} \setminus \{t\}.$$

La thèse résulte alors du fait que le programme ci-dessus ne modifie pas les poids des sommets de X'_{init} . ///

Le théorème suivant établit que l'algorithme de redistribution de flux permet de résoudre le problème du flot maximum.

THEOREME III.3.4. S'il converge, l'algorithme de redistribution de flux III.3.1. appliqué à la fonction admissible f modifie f de telle sorte qu'elle représente après exécution un flot de valeur maximum.

Démonstration. De fait, avant exécution de la dernière instruction, on a, vu les deux propositions précédentes,

$$e \in X', \; t \in \bar{X}',$$

$$f(X', \bar{X}') = c(X', \bar{X}'),$$

$$f(\bar{X}', X') = 0,$$

$$w(x) \leqslant 0, \quad \forall x \in X' \setminus \{e\},$$

$$w(y) = 0, \quad \forall x \in \bar{X}' \setminus \{t\},$$

$$0 \leqslant f(u) \leqslant c(u), \quad \forall u \in U,$$

$$w(e) = c(X', \bar{X}') + |w(X'^{-})|,$$

$$w(t) = -c(X', \bar{X}').$$

De plus, la fonction w est la fonction de poids associée à f. En effet, les propositions III.3.2. et III.3.3. établissent qu'on a, pour tout $x \in X \setminus \{e,t\}$, $w(x) = f(x,X) - f(X,x)$. En outre, la propostion III.1.5. appliquée à $Y = X'$, $\bar{X}'$ successivement implique que toute fonction de poids w' pour f vérifie

$$w'(e) = c(X', \bar{X}') - w'(X'^{-}) = c(X', \bar{X}') + |w(X'^{-})|$$

et

$$w'(t) = -c(X', \bar{X}').$$

Dès lors, comme on a $\omega^{-}(e) = \phi = \omega^{+}(t)$, il vient

$$f(e,X) = c(X', \bar{X}') + |w(X'^{-})|$$

et

$$f(X,t) = -c(X', \bar{X}'),$$

ce qui suffit.

Etablissons à présent que l'exécution de la dernière instruction ELIMINATION(X',Z) modifie les poids des sommets de X' de telle sorte qu'on ait

$$w(x) = 0, \quad \forall x \in X' \setminus \{e\}$$

sans altérer pour autant les flux de $X' \times \bar{X}'$. La dernière partie résulte de suite du théorème III.2.5.. Pour démontrer la première, procédons par l'absurde. Il existe alors $y \in X' \setminus \{e\}$ tel que $w(y) < 0$. Par suite, il vient, après exécution de ELIMINATION(X',Z), vu la proposition III.1.5.,

$$w(e) = c(X', \bar{X}') + |w(X'^{-})| > 0$$

et, par conséquent, puisqu'on a $(X' \setminus Z)^{+} = \phi$, $e \in Z$. Il s'ensuit qu'il n'existe pas de chemin de flot partant de e dans X'. Dans ces conditions, l'ensemble A des sommets de X' inaccessibles de e par un chemin de flot introduit une contradiction. D'une part, il vient

$$w(A) = f(A,\bar{A}) - f(\bar{A},A)$$

c'est-à-dire puisque la définition de A implique l'égalité $f(\bar{A},A) = 0$,

$$w(A) = f(A,\bar{A}) \geqslant 0.$$

D'autre part, comme on a $A = X' \setminus \{e\}$ et puisque tout sommet x de $X' \setminus \{e\}$ vérifie l'inégalité $w(x) \leqslant 0$, il vient

$$w(A) = w(X' \setminus \{e\}) = w(y) + w(X' \setminus \{e,y\}) < 0,$$

ce qui est absurde.

Démontrons enfin la thèse. Ce qui précède établit que la fonction admissible f vérifie en tout sommet distinct de e et t l'équation de conservation de la somme des flux. Il s'agit donc d'un flot. De plus, sa valeur est maximum car elle vaut la capacité $c(X',\bar{X}')$ de la coupe associée à $\bar{X}'$. ///

Le théorème suivant assure la convergence de l'algorithme III.3.1.. De plus, il détermine sa complexité. Comme sa démonstration sort du cadre de ce mémoire, nous l'admettons.

THEOREME III.3.5. L'algorithme de redistribution de flux III.3.1. appliqué à un réseau de transport à capacités entières et à une fonction admissible à valeurs entières converge. Sa complexité temporelle est $\mathcal{O}(n^5)$ au plus.

Démonstration. Résultat admis. (cfr. [34]) ///

CHAPITRE IV

LA METHODE DES PREFLOTS

IV.1. Le réseau de couches

Ce chapitre décrit la méthode des préflots. Il s'agit d'une méthode itérative dont chaque itération construit à partir d'un flot donné (par l'itération précédente ou, pour la première, par l'initialisation) dans un certain réseau une suite de préflots dont le dernier élément constitue un flot de valeur supérieure. Précisons tout d'abord le réseau que nous utilisons.

DEFINTION IV.1.1. Soient f un flot du réseau de transport $R = (X,U,c)$ et k la longueur minimum d'une chaîne f - augmentante dans $R(f)$. Le réseau de couches associé au réseau R et au flot f est le graphe dont l'ensemble des sommets est composé des couches L_i définie par les relations

$$L_0 = \{e\},$$

$$L_i = \{x \in X : x \notin U_{j=1}^{i-1} L_j \text{ et } \exists y \in U_{j=1}^{i-1} L_j :$$

$$((y,x) \in U \text{ et } f(y,x) < c(y,x)) \text{ ou } ((x,y) \in U \text{ et } f(x,y) > 0$$

$$i = 1, \ldots, k-1 ,$$

$$L_k = \{t\}$$

et dont l'ensemble des arcs est constitué des arcs $(x,y) \in U$ tels que $f(x,y) < c(x,y)$ pour lesquels il existe $i \in \{0, \ldots, k-1\}$ tel que $x \in L_i$ et $y \in L_{i+1}$ et des arcs $(y,x) \in U$ tel que $f(y,x) > 0$ et pour lesquels il existe $i \in \{0, \ldots, k-1\}$ tel que $x \in L_i$ et $y \in L_{i+1}$.

CONVENTION IV.1.2. Dans la suite de ce travail, la notation

$$(L_0, \ldots, L_k)_{f,c}$$

désigne le réseau de couches associé au réseau de transport (X,U,c) et au flot f de ce réseau.

Le théorème suivant lie les réseaux de référence et de couches.

THEOREME IV.1.3. Quand il existe, le réseau de couches associé au flot f et au réseau de transport $R = (X,U,c)$ s'identifie au graphe $(X,U_{R,f})$ du réseau de référence $(X,U_{R,f},c)$.

Démonstration. Cela résulte aussitôt des définitions II.9.1. et IV.1.3.. ///

Bien sûr, l'existence du réseau de couches associé au réseau de transport $R = (X,U,c)$ et à un de ses flots f dépend de l'existence d'une chaîne f - augmentante dans R. L'algorithme suivant, résultant d'une légère modification de l'algorithme CONSTRUCTION_RESEAU_REFERENCE II.9.12., permet de déterminer l'existence de ce réseau et, dans l'affirmative de le construire. Il repose aussi sur une exploration en largeur d'abord des sommets de R(f) et requiert les structures de données suivantes. Outre les couches L_i et la variable *echec*, sont associés à tout $x \in X$, l'ensemble $P(x)$ des prédécesseurs de x dans U, l'ensemble $S(x)$ des successeurs de x dans U ainsi que les ensembles $\omega_e(x)$, $\omega_i(x)$, $\rho_e(x)$ et $\rho_i(x)$ définissant comme spécifié dans la proposition IV.1.5. ci-dessous le réseau de couches. Nous l'écrivons à nouveau sous la forme d'une procédure à deux paramètres : f et echec.

ALGORITHME CONSTRUCTION_RESEAU_COUCHES(f,echec) IV.1.4. Soit f un flot d'un réseau de transport.

```
Pour tout x ∈ X effectuer
    num(x) := 0;
    ω_e(x) := ϕ;
    ω_i(x) := ϕ;
    r_e(x) := ϕ;
    r_i(x) := ϕ;
L_0 := {e};
num(e) := 1;
i := 1;
Répéter
    L_i := ϕ;
    fin_itérations := vrai;
    Pour tout x ∈ L_{i-1} effectuer
        Pour tout y ∈ S(x) effectuer
            si f(x,y) < c(x,y) et num(y) ∈ {0, i+1} alors
                num(y) := i + 1;
                ω_e(x) := ω_e(x) ∪ {(x,y)};
                ω_i(x) := ω_i(x) ∪ {(x,y)};
                L_i := L_i ∪ {y};
                fin_itérations := faux;
        Pour tout y ∈ P(x) effectuer
            si f(y,x) > 0 et num(y) ∈ {0, i+1} alors
                num(y) := i + 1;
                r_i(x) := r_i(x) ∪ {(y,x)};
                r_e(y) := r_e(y) ∪ {(y,x)};
                L_i := L_i ∪ {y};
                fin_itérations := faux;
    i := i + 1;
tant que (fin_itérations = faux) et (num(t) = 0);
Si num(t) = 0
    alors
        echec := vrai;
    sinon
        echec := faux;
        Pour tout x ∈ L_{num(t)-2} effectuer
```

```
Pour tout (x,y) ∈ ω_e(x) effectuer
  | si y ≠ t alors ω_e(x) := ω_e(x) \ {(x,y)};
Pour tout (y,x) ∈ r_i(x) effectuer
  | si y ≠ t alors r_i(x) := r_i(x) \ {(y,x)};
L_{num(t)-1} := {t}.
```

La proposition suivante le décrit.

PROPOSITION IV.1.5. Soit f un flot du réseau de transport R = (X,U,c). L'algorithme CONSTRUCTION RESEAU COUCHES(f,echec) IV.1.4. appliqué à f converge et affecte la valeur vrai à *echec* si et seulement si le flot f admet un réseau de couches. S'il en est ainsi, les propriétés suivantes sont vérifiées.

1) Les ensembles $\mathcal{L}_i$ définissent les couches du réseau de couches.

2) Pour tout $x \in X$, l'ensemble $\omega_e(x)$ est l'ensemble des arcs de $\omega^+(x) \cap U(f)$ dont l'extrémité terminale appartient à une couche d'indice supérieur d'une unité à celui de la couche contenant x.

3) Pour tout $x \in X$, l'ensemble $\omega_i(x)$ est l'ensemble des arcs de $\omega^-(x) \cap U(f)$ dont l'origine appartient à une couche d'indice inférieur d'une unité à celui de la couche contenant x.

4) Pour tout $x \in X$, l'ensemble $r_e(x)$ est l'ensemble des arcs de $\omega^+(x) \cap U(f)$ dont l'extrémité terminale appartient à une couche d'indice inférieur d'une unité à celui de la couche contenant x.

5) Pour tout $x \in X$, l'ensemble $r_i(x)$ est l'ensemble des arcs de $\omega^-(x) \cap U(f)$ dont l'origine appartient à une couche d'indice supérieur d'une unité à celui de la couche contenant x.

De plus, la complexité temporelle de l'algorithme IV.1.4. est $\mathcal{O}(n^2)$ au plus.

Démonstration. Il s'agit d'une simple vérification. ///

Les itérations de la méthode des préflots reposent sur les phases suivantes.

IV.2. Les phases d'augmentation et d'équilibrage

Soit un réseau de couches $(L_0, \ldots, L_k)_{f,c}$ associé au réseau de transport $R = (X,U,c)$ et à un de ses flots f. Soient, en outre, un préflot g de R et un entier $s \in \{0, \ldots, k-1\}$ vérifiant les deux assertions

(1) le sommet e est g-bloqué pour les chemins de longueur k,

(2) pour tout $x \in U_{i=s+1}^{k-1} L_i$, on a $g(x,X) = g(X,x)$.

Soient, enfin, pour tout $x \in X$, trois listes ordonnées $\alpha_e(x)$, $\alpha_i(x)$ et $\beta(x)$ soumises aux propriétés suivantes où les ensembles $\omega_e(x)$, $\omega_i(x)$, $r_e(x)$ et $r_i(x)$ réfèrent aux ensembles de la proposition IV.1.5..

(3) L'ensemble $\alpha_e(x)$ est inclu dans $\omega_e(x)$ et chacun de ses éléments sauf éventuellement le premier annule g. De plus, la non annulation de g en ce premier élément u_1 implique l'inégalité $g(u_1) < c(u_1)$.

(4) L'ensemble $\alpha_i(x)$ est inclu dans $r_i(x)$.

(5) La liste $\beta(x)$ est constituée de couples arc - réel dont les arcs appartiennent à $\omega_i(x) \cup r_e(x)$. En outre, la somme des modules de ces quantités est supérieure ou égale à la fois à $g(X,x) - g(x,X)$ et à la somme

$$(g - f)(\omega_i(x)) + (f - g)(r_e(x)).$$

La première phase - la phase d'augmentation - a pour objectif d'augmenter, tant que faire se peut, en tout sommet $x \in U_{i=s}^{k-1} L_i$ les flux des arcs de $\omega_e(x)$ tout en conservant les inégalités $g(X,x) \geqslant g(x,X)$. Dans ce but, une nouvelle fonction g' dont on espère qu'elle sera un préflot jouissant des propriétés ci-dessus est définie. On procède comme suit. Dans un premier temps, on pose, pour tout arc u U, $g'(u) = g(u)$. Ensuite, on analyse de proche en proche les sommets des couches L_s, L_{s+1}, ... , L_{k-1}. Le processus suivant se répète. Supposons avoir déjà examiné les sommets des couches L_s, ... ,L_j ($j < k-1$). Proposons-nous de traiter le sommet $x \in L_{j+1}$. Deux cas se présentent.

Dans le premier, le sommet x vérifie l'égalité

$$g(\omega^-(x) \setminus \omega_i(x)) + g'(\omega_i(x)) = g(\omega^+(x) \setminus r_e(x)) + g'(r_e(x)).$$

L'équation de conservation de la somme des flux en x est donc vérifiée et l'examen de x s'arrête là.

Dans le deuxième, on a

$$g(\omega^-(x) \setminus \omega_i(x)) + g'(\omega_i(x)) > g(\omega^+(x) \setminus r_e(x)) + g'(r_e(x)).$$

Dans ce cas, on tente tout d'abord d'éliminer les chaînes augmentantes de longueur k contenant des arcs renversés c'est-à-dire des arcs appartenant aux ensembles r_i. Dans ce but, on réduit successivement les flux des arcs de $\alpha_i(x)$ jusqu'à annulation de tous ces flux ou l'obtention de l'équation de conservation de la somme des flux en x. Pour ce faire, le procédé suivant est itéré. Supposons avoir déjà traité les arcs $u_1, \ldots, u_m$ de $\alpha_i(x)$. On pose alors

$$g'(u_{m+1}) = \max\{0, g(\omega^+(x) \setminus r_e(x)) + g'(r_e(x)) - g'(\{u_1, \ldots, u_m\}) - g(r_i(x) \setminus \{u_1, \ldots, u_{m+1}\}) - g'(\omega^-(x) \setminus r_i(x))$$

si l'on a

$$g'(\{u_1, \ldots, u_m\}) + g(r_i(x) \setminus \{u_1, \ldots, u_m\}) + g'(\omega^-(x) \setminus r_i(x)) > g(\omega^+(x) \setminus r_e(x)) + g'(r_e(x))$$

et

$$g'(u_{m+1}) = g(u_{m+1})$$

sinon. De plus, dans la première situation, l'élément $(u_{m+1}, g'(u_{m+1}) - g(u_{m+1}))$ est ajouté à la liste β de l'origine de u_{m+1} et, si l'on a $g'(u_{m+1}) = 0$, l'arc u_{m+1} est soustrait de la liste $\alpha_i(x)$.

Comme on le vérifie de suite, ce qui précède assure que la somme des flux des arcs incidents vers l'intérieur à x est encore supérieure ou égale à la somme des flux des arcs incidents vers l'extérieur à x.

Ensuite, on augmente autant que possible les flux g(u) des arcs de $\alpha_e(x)$. On procède de nouveau de proche en proche en itérant le procédé suivant jusqu'à saturation de tous les arcs de $\alpha_e(x)$ ou obtention de l'équation de conservation de la somme des flux en x.

Supposons avoir déjà inspecté les arcs $v_1, \ldots, v_n$ de $\alpha_e(x)$. D'une part, si l'arc v_{n+1} de $\alpha_e(x)$ vérifie l'inégalité

$$g'(\omega_i(x)) + g'(r_i(x)) + g(\omega^-(x) \setminus (\omega_i(x) \cup r_i(x))$$

$$> g'(r_e(x)) + g'(\{v_1, \ldots, v_n\}) + g(\omega^+(x) \setminus (r_e(x) \cup \{v_1, \ldots, v_n\})$$

on pose

$$g'(v_{n+1}) = \min\{c(v_{n+1}), g'(\omega_i(x)) + g'(r_i(x)) - g'(r_e(x))$$
$$+ g(\omega^-(x) \setminus (r_e(x) \cup \{v_1, \ldots, v_n\})$$
$$- g'(\{v_1, \ldots, v_n\})$$
$$- g(\omega^+(x) \setminus (r_e(x) \cup \{v_1, \ldots, v_{n+1}\}))\}$$

et on ajoute à la liste β relative à l'extrémité terminale de v_{n+1} l'élément $(u_{m+1}, g'(u_{m+1}) - g(u_{m+1}))$. D'autre part, si l'arc v_{n+1} vérifie l'égalité associée à l'inégalité précédente, l'équation de conservation de la somme des flux en x est satisfaite et le procédé s'arrête.

En outre, on modifie comme suit l'ensemble $\alpha_e(x)$. Soit u_p le dernier arc de $\alpha_e(x)$ dont on a pu accroître le flux et soit q le nombre d'éléments de $\alpha_e(x)$. On pose

$$\alpha_e(x) = \begin{cases} \{u_p, \ldots, u_q\} & \text{si } g'(u_p) < c(u_p) \\ \{u_{p+1}, \ldots, u_q\} & \text{si } p < q \text{ et } g'(u_p) = c(u_p) \\ \phi & \text{sinon.} \end{cases}$$

On vérifie de suite que la fonction g' est un pseudo-flot et qu'elle possède les propriétés

$$(6)\quad \forall x \in U_{i=s}^{k-1} L_i \quad \forall u \in \alpha_{e,init}(x) \setminus \alpha_{e,term}(x) : g'(u) = g(u),$$

$$(7)\quad \forall x \in U_{i=s}^{k-1} L_i \quad \forall u \in r_i(x) : g'(u) \leqslant g(u),$$

$$(8)\quad \forall x \in U_{i=s}^{k-1} L_i \quad \forall u \in \omega^+(x) \cup \omega^-(x) \setminus r_i(x) : g'(u) \geqslant g(u)$$

ainsi que les propriétés (3) et (5) écrites pour g', $\alpha_{e,term}(x)$ et $\beta_{term}(x)$ où $\alpha_{e,init}(x)$, $\alpha_{e,term}(x)$ et $\beta_{term}(x)$ désignent respectivement l'ensemble $\alpha_e(x)$ avant et après exécution de la phase

augmentation et l'ensemble $\beta(x)$ après exécution de cette même phase. De plus, l'entrée e est g'-bloquée. De fait, comme tout chemin de longueur k entre e et t est, par hypothèse g bloqué, il suffit, pour conclure, de prouver que tout arc d'un tel chemin g-saturé est aussi g'-saturé. Soit u un tel arc. D'une part, si l'origine de u appartient à $U_{i=0}^{s-1} L_i$, la thèse résulte de suite de l'égalité $g(u) = g'(u)$. D'autre part, si elle appartient à $U_{i=s}^{k-1} L_i$, comme u appartient nécessairement à un ensemble $\omega_e(x) \setminus \alpha_e(x)$, on a encore $g'(u) = g(u)$ et la thèse est à nouveau établie.

La deuxième phase - la phase d'équilibrage - équilibre les sommets de la couche d'indice supérieur dont un sommet au moins viole l'équation de conservation de la somme des flux. Pour ce faire, une nouvelle fonction g" est définie.

Initialement, on pose, pour tout arc $u \in U$, $g"(u) = g'(u)$. Ensuite, les modifications de flux mémorisées dans les listes β sont remises en cause, en commençant par les plus récentes et les réels $g"(u)$ sont transformés par itération du processus suivant. Soit x le sommet à traiter. Supposons avoir déjà examiné les arcs $u_1, \ldots, u_m$ de la liste $\beta(x)$. Deux cas se présentent. Dans le premier, on a

$$\begin{aligned} A = {} & g'(\omega^-(x) \setminus (\omega_i(x) \cup r_i(x)) + g'(\omega_i(x) \setminus \{u_1, \ldots, u_m\}) \\ & + g'(r_e(x)) + g"(\omega^-(x) \cap \{u_1, \ldots, u_m\}) \\ & - g'(\omega^+(x) \setminus \{u_1, \ldots, u_m\}) - g"(\{u_1, \ldots, u_m\} \cap \omega^+(x)) \\ > {} & 0 . \end{aligned}$$

Soit alors u_{m+1} l'arc suivant apparaissant dans la liste $\beta(x)$. Un tel arc ne peut appartenir, vu ce qui précède, qu'à un des deux ensembles $\omega_i(x)$ ou $r_e(x)$. D'une part, s'il appartient à $\omega_i(x)$, on pose

$$g"(u_{m+1}) = \max\{g'(u_{m+1}) - \Delta(u_{m+1}), -A + g'(u_{m+1})\} .$$

D'autre part, s'il appartient à $r_e(x)$, on pose

$$g"(u_{m+1}) = \min\{g'(u_{m+1}) - \Delta(u_{m+1}), A + g'(u_{m+1})\} .$$

De plus, dans les deux cas, l'élément $(u_{m+1}, \Delta(u_{m+1}))$ est enlevé de la liste $\beta(x)$ et remplacé par l'élément

$(u_{m+1}, g''(u_{m+1}) - g'(u_{m+1}) + \Delta(u_{m+1}))$ si cette dernière quantité est strictement positive. Dans le second, on a $A = 0$ et le processus se termine.

Enfin, afin de conclure l'examen de x, tout arc $(y,x) \in \omega_i(x)$ est retiré de l'ensemble $\alpha_e(y)$ et tout arc $(x,y) \in r_e(x)$ est retiré de l'ensemble $\alpha_i(y)$.

On vérifie de suite que la fonction g" ainsi définie est un préflot dans R tel que tout arc $u \in U_{x \in X} r_i(x)$ vérifie l'inégalité $g''(u) \geqslant g(u)$. De plus, elle possède les propriétés (1) à (5) ci-dessus où g, s, $\alpha_i(x)$, $\alpha_e(x)$, $\beta(x)$ sont respectivement remplacés par g", l'indice de la couche ci-dessus diminué d'une unité et les ensembles $\alpha_i(x)$, $\alpha_e(x)$, $\beta(x)$ tels qu'ils apparaissent après exécution des phases d'augmentation et d'équilibrage.

Les algorithmes IV.2.1. et IV.2.2. formalisent les paragraphes qui précèdent. Comme les opérations principales sur les listes α_e, α_i et β qui y interviennent sont l'insertion en tête de liste et la suppression du dernier élément introduit, nous les organisons sous forme de listes chaînées. Plus précisément, tout élément de $\alpha_e(x)$ est constitué

- d'un sommet *et* extrémité terminale d'un arc appartenant à $\omega_e(x)$,
- d'un pointeur *p* vers l'élément suivant,

tout élément de $\alpha_i(x)$

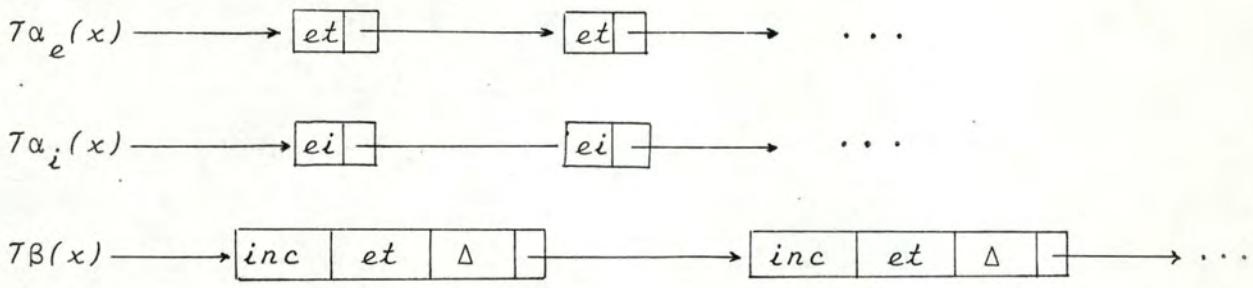
- d'un sommet *ei* origine d'un arc appartenant à $r_i(x)$,
- d'un pointeur *p* vers l'élément suivant

et tout élément de $\beta(x)$

- d'une marque *inc* égale à i ou e,
- d'un sommet *et* origine d'un arc de $\omega_i(x)$ si *inc* vaut i, extrémité terminale d'un arc de $r_e(x)$ sinon,
- d'un réel Δ positif si *inc* vaut i et négatif sinon,
- d'un pointeur *p* vers l'élément suivant.

Ainsi, si l'on désigne par $T\alpha_e(x)$, $T\alpha_i(x)$ et $T\beta(x)$ respectivement le pointeur vers le premier élément de $\alpha_e(x)$, $\alpha_i(x)$ et $\beta(x)$, on peut

schématiser ces listes comme suit



Les opérations suivantes seront utiles dans la suite. Elles permettent respectivement d'insérer l'élément *el* en tête de la liste chaînée *L*, de supprimer l'élément de tête de la liste chaînée *L* et d'éliminer la première occurrence de l'élément *el* dans la liste chaînée *L*. Nous représentons de nouveau par *p* le pointeur de chaque élément vers l'élément suivant de *L* et par *T* le pointeur vers la tête de *L*.

```
L := ajout(L,el) :
  | réserver(el');
  | el' := el;
  | el'.p := T;
  | T↑ := el'.

L := supp(L) :
  | el' := T;
  | T := el'↑.p;
  | élimner(el').

L := elim(L,el) :
  | el_non_trouve := vrai;
  | p' := T;
  | Tant que p' ≠ ϕ et el_non_trouve = vrai effectuer
  |   | si p'↑ = el
  |   |   | alors
  |   |   |   | el_non_trouve := faux;
  |   |   |   | el' := p'↑;
  |   |   |   | p' := p'↑.p;
  |   |   |   | éliminer(el');
  |   |   | sinon p' := p'↑.p.
```

Enonçons à présent les algorithmes IV.2.1. et IV.2.2.. Les ensembles ω_e, ω_i, r_e, r_i, α_e, α_i et β qui y apparaissent réfèrent bien sûr aux ensembles de même nom décrits ci-dessus. En outre, pour tout $x \in X$, la quantité $(in - out)(x)$ détermine la valeur courante de la différence entre la somme des flux circulant sur les arcs incidents vers l'intérieur en x et la somme des flux circulant sur les arcs incidents vers l'extérieur en x.

ALGORITHME PHASE_AUGMENTATION(g,s) IV.2.1. Soient un préflot g et un entier s vérifiant les propriétés (1) à (5) ci-avant.

```
Pour tout u ∈ U effectuer g'(u) := g(u);
Pour i variant de s à k-1 répéter
  Pour tout x ∈ L_i effectuer
    Tant que Γα_i(x) ≠ φ et (in - out)(x) > 0 effectuer
      de := Γα_i(x)↑.ei;
      g'(de,x) := max{0, g(de,x) - (in - out)(x)};
      (in - out)(x) := (in - out)(x) + g'(de,x) - g(de,x);
      (in - out)(de) := (in - out)(de) + g(de,x) - g'(de,x);
      ajoût(β(de),(e,x,g'(de,x) - g(de,x)));
      supp(α_i(x));
    Si g'(de,x) > 0 alors
      ajoût(α_i(x),de);
    Tant que Γα_e(x) ≠ φ et (in - out)(x) > 0 effectuer
      de := Γα_e(x)↑.et;
      g'(x,de) := min{c(x,de), g(x,de) + (in - out)(x)};
      (in - out)(x) := (in - out)(x) + g(x,de) - g'(x,de);
      (in - out)(de) := (in - out)(de) + g'(x,de) - g(x,de);
      ajoût(β(de),(i,x,g'(x,de) - g(x,de)));
      supp(α_e(x));
    Si g'(x,de) < c(x,de) alors
      ajoût(α_e(x),de).
```

ALGORITHME PHASE_EQUILIBRAGE(g',s') IV.2.2. Soit g' un préflot tel que ci-dessus.

```
Pour tout u∈U effectuer g"(u) := g'(u);
i := k;
couche_trouvée := non;
Répéter
  i := i - 1;
  Pour tout x∈L_i effectuer
    si (in - out)(x) ≠ 0 alors couche_trouvée := oui;
tant que couche_trouvée = non;
Si i ≠ 1 alors
  Pour tout x∈L_i effectuer
    Tant que (in - out)(x) > 0 répéter
      el := Tβ(x)↑;
      si el.inc = e alors
        g"(x,el.et) := min{g'(x,el.et) - el.Δ, g'(x,el.et) +
                                              (in - out)(x)};
        (in - out)(x) := (in - out)(x) + g'(x,el.et)
                                       - g"(x,el.et);
        (in - out)(el.et) := (in - out)(el.et) + g"(x,el.et)
                                               - g'(x,el.et);
      si el.inc = i alors
        g"(el.et,x) := max{g'(el.et,x) - el.Δ, g'(el.et,x) -
                                              (in - out)(x)};
        (in - out)(x) := (in - out)(x) + g"(el.et,x)
                                       - g'(el.et,x);
        (in - out)(el.et) := (in - out)(el.et) + g'(el.et,x)
                                               - g"(el.et,x);
      supp(β(x));
    Si el.inc = i alors
      si g"(x,el.et) - g'(x,el.et) + el.Δ > 0 alors
        ajoût(β(x),(i,el.et,g"(x,el.et) - g'(x,el.et) + el.Δ));
    Si el.inc = e alors
      si g"(el.et,x) - g'(el.et,x) + el.Δ > 0 alors
        ajoût(β(x),(e,el.et,g"(el.et,x) - g'(el.et,x) + el.Δ));
    Pour tout (y,x)∈ω_i(x) effectuer
      elim(α_e(y),x);
    Pour tout (x,y)∈r_e(x) effectuer
      elim(α_i(y),x);
s' := i - 1.
```

IV.3. La méthode des préflots

Décrivons à présent la méthode des préflots. Elle consiste en une suite d'itérations dont chacune a pour objectif de transformer un flot donné en un flot de valeur supérieure. Dans ce but, chacune d'entre elles tente tout d'abord de construire le réseau de couches associé au flot courant. En cas d'échec, le théorème II.1.2. affirme que ce flot est de valeur maximum. Sinon, un flot de valeur supérieure peut être défini. Pour ce faire, les arcs incidents vers l'extérieur à l'entrée e sont saturés puis une succession de phases d'augmentation et d'équilibrage modifie progressivement le flot courant, établissant ainsi, comme nous l'avons vu, une suite de préflots dont le dernier élément est le flot cherché.

L'algorithme suivant en découle.

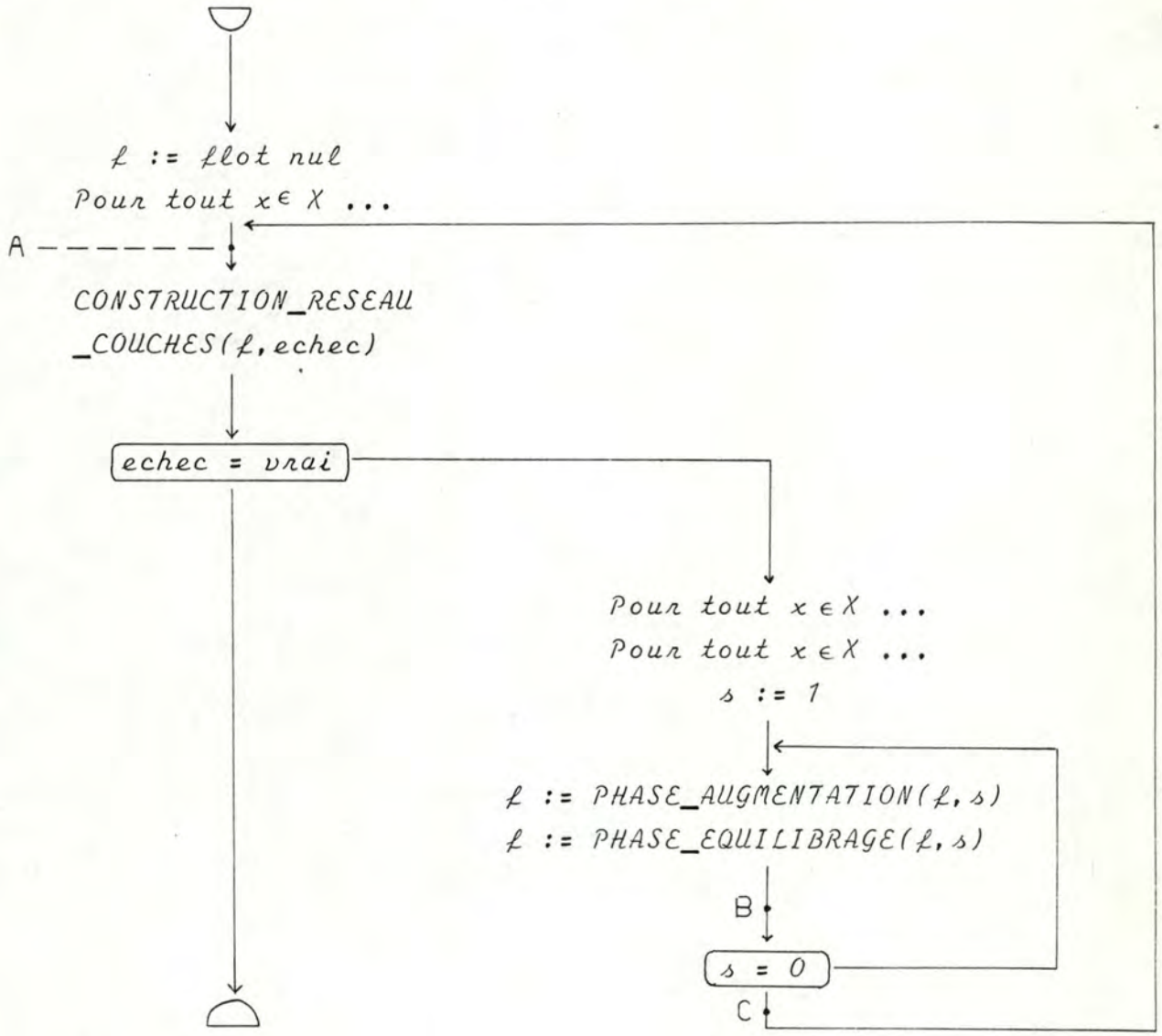
ALGORITHME METHODE_DES_PREFLOTS IV.3.1.

```
f := flot nul;
Pour tout x ∈ X effectuer
  | (in - out)(x) := 0;
Répéter
  | CONSTRUCTION_RESEAU_COUCHES(f, echec);
  | Si echec = faux alors
  |   | Pour tout x ∈ X effectuer
  |   |   | α_i(x) := r_i(x);
  |   |   | α_e(x) := ω_e(x);
  |   |   | β(x) := ϕ;
  |   | Pour tout x ∈ L_1 effectuer
  |   |   | (in - out)(x) := (in - out)(x) - f(e,x) + c(e,x);
  |   |   | ajout(β(x), (i, e, c(e,x) - f(e,x));
  |   |   | (in - out)(e) := (in - out)(e) - c(e,x) + f(e,x);
  |   |   | f(e,x) := c(e,x);
  |   | s := 1;
  |   | Répéter
  |   |   | f := PHASE_AUGMENTATION(f, s);
  |   |   | f := PHASE_EQUILIBRAGE(f, s);
  |   | tant que s ≠ 0;
| tant que echec = faux.
```

Le théorème ci-dessous le décrit.

THEOREME IV.3.2. L'algorithme METHODE DES PREFLOTS IV.3.1. converge et détermine après exécution un flot de valeur maximum. Sa complexité temporelle est de $\mathcal{O}(n^3)$ au plus.

Démonstration. Etablissons tout d'abord que, s'il converge, l'algorithme IV.3.1. fournit, après exécution, un flot de valeur maximum. Dans ce but, représentons-le par l'organigramme suivant:



Le paragraphe IV.2. prouve qu'en tout passage en B, f représente un pseudo-flot vérifiant les propriétés (1) à (5) ci-avant. Dès lors, l'inégalité s = 0 implique qu'en tout passage en C et, par suite, en tout passage en A, le préflot f est aussi un flot. Le

théorème II.1.2. permet alors de conclure.

Prouvons ensuite que l'algorithme IV.3.1. converge. D'une part, la conclusion de l'examen du sommet x dans la phase d'équilibrage entraîne que tout sommet ne peut faire l'objet que d'un seul équilibrage. Il s'ensuit que l'instruction itérative *Répéter ... tant que* $s \neq 0$ converge après au plus n itérations. D'autre part, comme on le démontre aisément, le réseau de couches courant en C ne contient pas de chaîne f-augmentante de longueur égale à la distance séparant e à t dans ce réseau. Par conséquent, la boucle *Répéter ... tant que echec = faux* converge aussi après au plus n itérations.

Déterminons enfin la complexité de l'algorithme IV.3.1.. Comme à chaque passage en A, la longueur minimum des chaînes f-augmentantes croit d'une unité au moins, il suffit de prouver que le corps de la boucle *Répéter ... tant que echec = faux* a une complexité temporelle de $\mathcal{O}(n^2)$ au plus. Ceci revient encore à établir, vu les algorithmes IV.2.1. et IV.2.2., que le nombre de changements de flux est inférieur ou égal à un multiple de n^2. De fait, tout arc d'un ensemble $\alpha_e(x)$ (resp. $\alpha_i(x)$) une fois saturé (resp. son flux annulé) est enlevé de cet ensemble. De plus, à chaque exécution de l'instruction f := *PHASE_AUGMENTATION*(f,s), pour tout sommet $x \in X$, au plus un arc de $\alpha_e(x)$ (resp. $\alpha_i(x)$) fait l'objet d'une séquence d'affectations conduisant à une non saturation (resp. à une non annulation) de son flux. La thèse résulte alors immédiatement du fait que cette boucle s'exécute au plus n fois. ///

CHAPITRE V

LA PROGRAMMATION LINEAIRE

Les problèmes du flot maximum, du flot b-canalisé maximum, du flot maximum de coût minimum, du flot b-canalisé maximum de coût minimum et de transfert de Hitchcock peuvent, bien sûr, être résolus grâce à la théorie de la programmation linéaire.

Rappelons que le problème du flot maximum (resp. du flot b-canalisé maximum) consiste en la recherche de m nombres (réels ou entiers) $f(u_1), \ldots, f(u_m)$ maximisant la fonction

$$w = \sum_{u \in \omega^-(t)} f(u)$$

et soumis aux contraintes

$$\sum_{u \in \omega^-(x)} f(u) = \sum_{u \in \omega^+(x)} f(u) , \quad \forall x \in X \setminus \{e,t\},$$

$$0 \leqslant f(u) \leqslant c(u) , \forall u \in U \quad (\text{resp. } b(u) \leqslant f(u) \leqslant c(u), \ \forall u \in U)$$

c'est-à-dire à la résolution du problème de programmation linéaire

$$\min \ - \sum_{u \in \omega^-(t)} f(u)$$

sous les contraintes

$$\sum_{u\in\omega^-(x)} f(u) = \sum_{u\in\omega^+(x)} f(u) \ , \quad \forall x \in X\setminus\{e,t\},$$

$$0 \leqslant f(u) \leqslant c(u) \ , \quad \forall u \in U \quad (\text{resp. } b(u) \leqslant f(u) \leqslant c(u), \quad \forall u \in U)$$

en les inconnues (réelles ou entières) $f(u_1)$, ... ,$f(u_m)$.

Par ailleurs, le problème du flot (resp. flot b-canalisé) maximum de coût minimum revient à la recherche, parmi les flots (resp. flots b-canalisés) maximum, d'un flot minimisant la fonction

$$\sum_{u\in U} d(u)f(u) \ .$$

Un tel flot peut être obtenu par résolution du problème de programmation linéaire en les inconnues $f(u_1)$, ... ,$f(u_m)$:

$$\min \left(-M \sum_{u\in\omega^-(t)} f(u) + \sum_{u\in U} d(u)f(u) \right)$$

$$\text{sous les contraintes} \quad \sum_{u\in\omega^-(x)} f(u) = \sum_{u\in\omega^+(x)} f(u), \ \forall x \in X\setminus\{e,t\}$$

$$0 \leqslant f(u) \leqslant c(u) \ (\text{resp. } b(u) \leqslant f(u) \leqslant c(u) \ ), \quad \forall u \in U$$

où M désigne une constante positive arbitrairement grande.

Enfin, le problème de transfert de Hitchcock, cas particulier du problème précédent, peut, bien sûr, être ramené à un problème de programmation linéaire.

Notre but, ici, n'est nullement de développer la théorie de la programmation linéaire mais, seulement de rappeler les résultats nécessaires à la compréhension des algorithmes que nous proposons. Nous renvoyons le lecteur désireux d'en connaître davantage au livre [35] de Simmonard.

V.1. Rappels et conventions

Tout problème de la programmation linéaire simple s'écrit sous la forme

$$\min z = \sum_{j=1}^{n} c_j x_j$$

sous les contraintes

$$\sum_{j=1}^{n} a_{ij} x_j \geqslant b_i \ , \quad i \in I_1 \ ,$$

$$\sum_{j=1}^{n} a_{ij} \leqslant b_i \ , \quad i \in I_2 \ ,$$

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \ , \quad i \in I_3 \ ,$$

$$x_j \geqslant 0 \ , \quad j \in J_1$$

x_j quelconque, $j \in J_2$

où a_{ij} , b_i , c_j $(i \in I_1 \cup I_2 \cup I_3,\ j \in J_1 \cup J_2)$ sont des constantes données, I_1, I_2, I_3, J_1, J_2 des sous-ensembles, éventuellement vides, de N tels que $i_1 \neq i_2 \Rightarrow I_{i_1} \cap I_{i_2} = \phi$ $(i_1, i_2 \in \{1,2,3\})$ et $J_1 \cap J_2 = \phi$, et x_j $(j \in J_1 \cup J_2)$ les inconnues du problème. Comme on le vérifie aisément, quitte à introduire de nouveaux coefficients a_{ij}, b_i, c_j et de nouvelles variables x_j, on peut toujours ramener un tel problème sous l'une des deux formes suivantes:

a) <u>forme standard</u>

$$\min z = \sum_{j=1}^{n} c_j x_j$$

sous les contraintes

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \ , \quad i = 1, \ldots, m$$

$$x_j \geqslant 0 \ , \quad j = 1, \ldots, n$$

b) <u>forme canonique</u>

$$\min z = \sum_{j=1}^{n} c_j x_j$$

sous les contraintes

$$\sum_{j=1}^{n} a_{ij} x_j \geqslant b_i \ , \quad i = 1, \ldots, m$$

$$x_j \geqslant 0 \ , \quad j = 1, \ldots, n$$

CONVENTION V.1.1. Dans la suite de ce travail, nous supposerons, sauf mention explicite du contraire, tout problème de programmation linéaire simple écrit sous forme standard; les symboles a_{ij}, b_i, c_i y représentant des données, les symboles x_i les inconnues et $z = \Sigma_{i=1}^{n} c_j x_j$ la fonction économique. En outre, nous utiliserons les abréviations

" PLS " pour "problème de programmation linéaire simple"

" s.c. " pour "sous les contraintes".

Donnons à présent quelques définitions particulières à la forme standard de la programmation linéaire simple.

DEFINITIONS V.1.2. Soit le PLS

$$\min z = \sum_{j=1}^{n} c_j x_j \qquad (1)$$

$$\text{s.c.} \quad \sum_{j=1}^{n} a_{ij} x_j = b_j \ , \ i = 1, \ldots, m \qquad (2)$$

$$x_j \geqslant 0 \ , \ j = 1, \ldots, n \qquad (3)$$

a) Le système(2) est <u>compatible</u> s'il possède au moins une solution, il est <u>incompatible</u> dans le cas contraire.

b) Une équation est <u>redondante</u> si elle est combinaison d'une ou plusieurs autre(s) équation(s).

c) Un système est <u>non redondant</u> ou <u>indépendant</u> s'il ne contient aucune équation redondante, il est <u>redondant</u> ou <u>dépendant</u> dans le cas contraire.

d) Une <u>base du système(2)</u> ou <u>base du PLS</u> est une base de l'ensemble des vecteurs colonnes $P_j = (a_{1j}, \ldots, a_{mj})$. Une base du PLS est donc une sous-matrice de la matrice $A = (a_{ij})$ régulière d'ordre m et inversement.

e) Les m variables associées aux colonnes d'une base B sont appelées <u>variables de base</u> tandis que les n-m autres variables sont appelées <u>variables hors-base</u> ou <u>variables indépendantes relativement à B</u>.

CONVENTION V. 1.3. Dans la suite de ce travail,

B

désigne une base du PLS traité et

N

la matrice dont les vecteurs colonnes sont les vecteurs colonnes de la matrice $A=(a_{ij})$ n'appartenant pas à B. En outre,

x_B, x_N, c_B, c_N

désignent respectivement les sous-vecteurs de $x=(x_1, \ldots, x_n)$ correspondant aux variables de base et aux variables indépendantes relativement à B et les sous-vecteurs de $c=(c_1, \ldots, c_n)$ correspondant aux variables de base et hors base relativement à B.

De plus, nous posons

$$b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} ,$$

$$\bar{b} = B^{-1}b ,$$

$$P_j = \begin{bmatrix} a_{1j} \\ \vdots \\ a_{mj} \end{bmatrix} \qquad j = 1, \ldots, n ,$$

$$\bar{P}_j = \begin{bmatrix} \bar{a}_{1j} \\ \vdots \\ \bar{a}_{mj} \end{bmatrix} = B^{-1}P_j \qquad j = 1, \ldots, n ,$$

$$\pi = c_B B^{-1} ,$$

$$\bar{c}_j = c_j - \pi P_j \qquad j = 1, \ldots, n ,$$

$$\bar{z}_o = c_B \bar{b} = c_B B^{-1} b = \pi b$$

Remarquons dès à présent qu'on peut réécrire, en utilisant ces notations, le système(2) ci-dessus sous la forme

$$x_B + B^{-1}Nx_N = \bar{b}$$

Cette dernière suggère une solution toute naturelle du système(2): $x_B = \bar{b}$, $x_N = 0$ <u>solution de base associée à B</u>.

Par ailleurs, comme on a, en désignant par J l'ensemble des indices correspondant aux variables indépendantes relativement à B,

$$z = c_B x_B + \sum_{j \in J} c_j x_j$$

$$= c_B \bar{b} - \sum_{j \in J} x_j c_B \bar{P}_j + \sum_{j \in J} c_j x_j$$

il vient

$$(-z) + \sum_{j \in J} (c_j - c_B \bar{P}_j) x_j = -c_B \bar{b}$$

c'est-à-dire

$$(-z) + \sum_{j \in J} \bar{c}_j x_j = -\bar{z}_o.$$

Il s'ensuit que la solution de base associée à B donne à la fonction économique la valeur $\bar{z}_o$.

Définissons ci-dessous quelques notions relatives aux bases et aux solutions de base associées.

DEFINITION V.1.4.

a) Une solution de base est <u>dégénérée</u> si la valeur d'une ou plusieurs variables de base est nulle.

b) Une <u>solution de base réalisable</u> ou <u>programme de base</u> est une solution de base satisfaisant à $x \geqslant \underline{0}$.

c) Une <u>solution de base réalisable minimale (resp. maximale)</u> ou <u>programme minimal (resp. maximal)</u> est une solution de base réalisable qui donne à la fonction économique une valeur minimale (resp. maximale) finie.

d) Une <u>base réalisable</u> est une base à laquelle correspond une solution de base réalisable, une <u>base non réalisable</u> est une base qui n'est pas réalisable.

e) Une solution réalisable (minimale (resp. maximale)) est une solution du système(2) satisfaisant aux contraintes (3) (et fournissant une valeur minimale (resp. maximale) finie de la fonction économique).

f) Une base minimale (resp. maximale) est une base réalisable à laquelle correspond une solution de base réalisable minimale (resp. maximale).

La proposition suivante se déduit de suite des définitions précédentes.

PROPOSITION V.1.5. La solution de base associée à la base B est réalisable si et seulement si on a, pour tout $i \in \{1, \ldots, n\}$, $\bar{b}_i \geqslant 0$.

Démonstration. C'est immédiat. ///

Enfin, les trois théorèmes suivants seront d'une importance capitale dans la suite.

THEOREME V.1.6. Une solution de base réalisable est minimale si tous les coûts relatifs correspondants aux variables indépendantes sont positifs ou nuls.

Démonstration. Résultat admis (cf. [35]). ///

THEOREME V.1.7. S'il existe un indice s dans le système canonique associé à la base B tel que $\bar{c}_s < 0$ et tel que, pour tout $i \in \{1, \ldots, m\}$ on ait, $\bar{a}_{is} \leqslant 0$, alors le système (2) possède une classe de solutions réalisables ayant (m+1) variables strictement positives, pour laquelle l'ensemble des valeurs prises par la fonction économique n'a pas de borne inférieure finie.

Démonstration. Résultat admis (cf. [35]). ///

THEOREME V.1.8. S'il existe un indice s dans l'équivalent canonique associé à la base B tel que $\bar{c}_s < 0$ et tel qu'il existe $i \in \{1, \ldots, m\}$ tel que $\bar{a}_{is} > 0$ et si la solution de base réalisable associée à B n'est pas dégénérée, alors on peut construire (cf. algorithme primal du simplexe) à partir de celle-ci une nouvelle solution de base réalisable donnant à la fonction économique une valeur inférieure.

Démonstration. Résultat admis (cf. [35]). ///

V.2. La méthode primale du simplexe

V.2.1. L'algorithme primal du simplexe

Afin d'alléger l'écriture de l'algorithme primal du simplexe, adoptons, tout d'abord, la convention suivante.

CONVENTION V.2.1.1. Etant donné une base B, nous représentons le PLS

$$\min z = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.c.} \quad \sum_{j=1}^{n} a_{ij} x_j = b_i$$

$$x_j \geq 0$$

sous la forme d'un tableau, appelé tableau du simplexe, construit comme suit:

	1	x_1	x_2	. . .	x_n
(-z)	$(-\bar{z}_o)$	$\bar{c}_1$	$\bar{c}_2$	. . .	$\bar{c}_n$
1	$\bar{b}_1$	$\bar{a}_{11}$	$\bar{a}_{12}$	. . .	$\bar{a}_{1n}$
.	.	.	.		.
.	.	.	.		.
.	.	.	.		.
m	$\bar{b}_m$	$\bar{a}_{m1}$	$\bar{a}_{m2}$	. . .	$\bar{a}_{mn}$

En outre, nous désignons par "pivotage du tableau du simplexe avec $\bar{a}_{rs}$ $(\neq 0)$ pour pivot" la transformation de ce tableau définie par les affectations

$$\bar{z}_o := \bar{z}_o + \frac{\bar{b}_r}{\bar{a}_{rs}} \bar{c}_s$$

$$b_i := \begin{cases} \bar{b}_i - \bar{a}_{is} \dfrac{\bar{b}_r}{\bar{a}_{rs}} & \text{si } i = 1, \ldots, r-1, r+1, \ldots, m \\ \dfrac{\bar{b}_r}{\bar{a}_{rs}} & \text{si } i = r \end{cases}$$

$$\bar{c}_j := \begin{cases} \bar{c}_j - \bar{c}_s \dfrac{\bar{a}_{rj}}{\bar{a}_{rs}} & \text{si } j = 1, \ldots, r-1, r+1, \ldots, n \\ -\dfrac{\bar{c}_s}{\bar{a}_{rs}} & \text{si } j = r \end{cases}$$

$$\bar{a}_{ij} := \begin{cases} \bar{a}_{ij} - \bar{a}_{is} \dfrac{\bar{a}_{rj}}{\bar{a}_{rs}} & \text{si } j = 1, \ldots, n \text{ et } i = 1, \ldots, r-1, r+1, \ldots, m \\ \dfrac{\bar{a}_{rj}}{\bar{a}_{rs}} & \text{si } j = 1, \ldots, n \text{ et } i = r \end{cases}$$

La convergence de l'algorithme proposé repose sur la notion de vecteur lexicographiquement positif.

DEFINITION V.2.1.2. Un vecteur v est <u>lexicographiquement positif (négatif)</u> si sa première composante non nulle est positive (négative). Il est <u>lexicographiquement supérieur (resp. inférieur)</u> au vecteur w si v-w (resp. w-v) est lexicographiquement positif.

Enonçons à présent l'algorithme primal du simplexe. Il s'agit d'un algorithme itératif qui permet d'évoluer de base réalisable en base réalisable jusqu'à l'obtention d'une base réalisable minimale. Il repose essentiellement sur les théorèmes V.1.6., V.1.7. et V.1.8. et suppose la connaissance a priori d'une base réalisable.

ALGORITHME PRIMAL DU SIMPLEXE V.2.1.3. Soit B une base réalisable.

(1). Mettre l'équivalent canonique associé à la base B sous la forme du tableau du simplexe en rangeant les colonnes de telle façon que toutes les lignes sauf éventuellement celle correspondant à la fonction économique, constituent des vecteurs lexicographiquement positifs de R^{n+1}.

(2). Si tous les coûts relatifs $\bar{c}_j$ sont positifs, arrêter les calculs: la solution de base réalisable actuelle est minimale. Sinon, aller à (3).

(3). Sélectionner l'un des $\bar{c}_j < 0$, par exemple

$$\bar{c}_s = \min_{j:\bar{c}_j<0} \bar{c}_j$$

et aller en (4). (critère d'entrée)

(4). Si pour tout $i \in \{1, \ldots, m\}$ on a $\bar{a}_{is} \leqslant 0$, alors arrêter les calculs: la fonction économique n'a pas de minimum fini. Sinon, aller en (5).

(5). Calculer

$$\min_{i:\bar{a}_{is}>0} \frac{\bar{b}_i}{\bar{a}_{is}} .$$

Si ce minimum se présente pour une seule valeur de i, soit $i = r$, alors poser $\alpha = r$. Sinon, déterminer l'ensemble I_1 des indices pour lesquels le minimum est atteint et calculer

$$\min_{i\in I_1} \frac{\bar{a}_{i1}}{\bar{a}_{is}} .$$

Si ce minimum se présente pour une seule valeur de i, poser α égal à cette valeur. Sinon, déterminer l'ensemble $I_2 \subset I_1$ des indices pour lesquels ce minimum est atteint et calculer

$$\min_{i\in I_2} \frac{\bar{a}_{i2}}{\bar{a}_{is}} , \ldots$$

Et ainsi de suite jusqu'à obtenir un ensemble I_e réduit à un singleton, soit $I_e = \{x_r\}$. Poser alors $\alpha = r$. (critère de sortie) Aller en (6).

(6). Pivoter le tableau du simplexe avec $\bar{a}_{\alpha s}$ pour pivot. Aller en (2).

Le théorème suivant fournit la complexité de l'algorithme primal du simplexe.

THEOREME V.2.1.4. La complexité temporelle de l'algorithme primal du simplexe V.2.1.3. est de $\mathcal{O}(A_n^m\ nm)$.

Démonstration. De fait, d'une part, comme on le vérifie de suite, la complexité temporelle de chaque itération de l'algorithme V.2.1.3. est de $\mathcal{O}(nm)$. D'autre part, il existe A_n^m bases différentes et, par suite, l'algorithme V.2.1.3. converge après au plus A_n^m itérations. ///

V.2.2. Cas des variables bornées supérieurement

Considérons un problème de programmation linéaire simple dans lequel certaines des variables sont astreintes à demeurer inférieures ou égales à des bornes strictement positives v_j:

$$\min z = \Sigma_{j=1}^{n} c_j x_j$$

$$\text{s.c. } \Sigma_{j=1}^{n} a_{ij} x_j = b_i \quad , i = 1, \ldots, m$$

$$0 \leqslant x_j \quad , j = 1, \ldots, n$$

$$x_j \leqslant v_j \quad , j \in J \subset \{1, \ldots, n\} \quad (4)$$

L'application de l'algorithme primal du simplexe V.2.1.3. exigerait la transformation préalable des relations

$$x_j \leqslant h_j \quad , j \in J$$

en équations

$$x_j + x_j^e = h_j \quad , j \in J$$

par addition de variables d'écart x_j^e. Le nombre des équations ainsi que le nombre des variables seraient augmentés. Dès lors, la dimension du tableau et, par suite, le temps d'exécution croîtrait.

La méthode suivante permet de pallier à ces inconvénients en traitant les contraintes du type (4) à l'intérieur même du code d'instructions de l'algorithme primal du simplexe. Elle est basée sur les concepts de solution de base généralisée et de solution réalisable généralisée.

DEFINITION V.2.2.1. Soient le problème de programmation linéaire simple

$$\min z = \Sigma_{j=1}^{n} c_j x_j \qquad (5)$$

$$\text{s.c.} \quad \Sigma_{j=1}^{n} a_{ij} x_j = b_i \ , \ i = 1, \ldots, m \qquad (6)$$

$$0 \leqslant x_j \leqslant v_j \ , \ j = 1, \ldots, n \qquad (7)$$

et B une base du système (6).

a) Une solution de base généralisée est une solution particulière du système $x_B + B^{-1}Nx_N = \bar{b}$ obtenue en annulant certaines des (n-m) variables indépendantes et en égalant les autres variables indépendantes à leurs bornes supérieures.

b) Une solution réalisable généralisée de (6) est une solution de (6) satisfaisant à (7).

REMARQUE V.2.2.2. Remarquons que tout PLS peut être ramené à la forme (5), (6), (7) ci-dessus quitte à égaler certains des v_j à $+\infty$.

Le théorème suivant fournit un critère d'optimalité pour une solution de base réalisable généralisée.

THEOREME V.2.2.3. Une solution de base réalisable généralisée est minimale si les coûts relatifs $\bar{c}_j$ correspondant aux variables indépendantes vérifient les relations

$$\begin{aligned} &\bar{c}_j \geqslant 0 \quad \text{si} \quad x_j = 0 \\ &\bar{c}_j \leqslant 0 \quad \text{si} \quad x_j = v_j \end{aligned} \qquad (8)$$

Démonstration. Soient B la base associée à la solution de base généralisée et I l'ensemble des indices des variables indépendantes. Comme on a, vu V.1.3.,

$$z = c_B B^{-1} b + \sum_{i \in I} \bar{c}_i x_i \qquad (9)$$

il résulte des hypothèses que tout changement des valeurs des variables indépendantes entraîne une augmentation de la valeur de la fonction économique, ce qui établit le théorème. ///

Modifions à présent l'algorithme primal du simplexe. A cet effet, supposons que nous connaissions une solution de base réalisable généralisée de base correspondante B et que l'une des variables indépendantes, soit x_s, ne vérifie pas l'une des deux relations (8). On a donc $\bar{c}_s < 0$ si $x_s = 0$ et $\bar{c}_s > 0$ si $x_s = v_s$ et, en vertu de (9), on peut améliorer la solution de base actuelle en faisant croître x_s à partir de 0 dans le premier cas ou décroître x_s à partir de v_s dans le second cas. Les relations (8) constituent, dès lors, le nouveau critère d'entrée.

Nous obtiendrions une diminution maximale de z si nous faisions croître (ou décroître) la variable indépendante x_s jusqu'à ce qu'elle atteigne sa borne supérieure v_s (ou inférieure 0) ou, jusqu'à ce que l'une des variables de base x_r atteigne l'une de ses bornes. Deux cas sont donc à considérer:

i) La variable indépendante x_s atteint la première une de ses bornes

La base reste inchangée et il faut examiner les coûts relatifs $\bar{c}_j$ des autres variables indépendantes. Aussi longtemps que la condition (8) n'est pas satisfaite, et pour autant que nous restions dans ce cas i), nous modifions la valeur des variables indépendantes violant (8).

Ainsi, après avoir modifié les valeurs d'un certain nombre de variables indépendantes, on arrive à l'une des deux situations suivantes:

1) la condition (8) est satisfaite: la solution de base généralisée calculée en dernier lieu est minimale,

2) l'une des variables de base atteint une de ses bornes avant que la variable indépendante contrôlée n'atteigne sa borne supérieure ou inférieure. On se trouve alors dans le cas ii).

ii) La variable de base x_r atteint la première une de ses bornes

On change alors de base: x_r devient variable indépendante avec la valeur 0 ou v_r qu'elle a atteinte et est remplacée dans l'ensemble des variables de base par x_s qui prend la valeur "critique" pour laquelle x_ℓ atteint sa borne. Le changement de base se fait de façon habituelle par pivotage du tableau du simplexe avec $\bar{a}_{rs}$ pour pivot.

Déterminons maintenant la valeur critique de x_s c'est-à-dire au-delà de laquelle on ne peut plus faire varier x_s soit parce qu'une des variables de base atteint une de ses bornes, soit parce que la variable x_s elle-même atteint une de ses bornes. Dans ce but, distinguons les deux cas suivants:

a) x_s décroît à partir de sa borne supérieure v_s

Dans ce cas, si, pour tout indice $i \in \{1, \ldots, n\}$, $\hat{x}_i$ et $\tilde{x}_i$ désignent respectivement la valeur de la variable x_i avant et après variation de x_s, on a, pour toute variable x_k de base,

$$\tilde{x}_k = \hat{x}_k + \bar{a}_{n(k)s} (v_s - \tilde{x}_s) \qquad (10)$$

où n(k) est l'entier tel que $\bar{P}_k = e_{n(k)}$. Comme on doit bien évidemment avoir,

$$\tilde{x}_s \geqslant 0$$

et

$$0 \leqslant \hat{x}_k + \bar{a}_{n(k)s} (v_s - \tilde{x}_s) \leqslant v_k$$

pour toute variable de base x_k, la valeur crtique de x_s vaut, dans ce cas,

$$\tilde{x}_s = \max_{k \in K} \left\{ 0 \; ; \; v_s + \frac{v_k - \hat{x}_k}{-\bar{a}_{n(k)s}} \text{ si } \bar{a}_{n(k)s} > 0 \; ; \; v_s + \frac{x_k}{\bar{a}_{n(k)s}} \right.$$

$$\left. \text{si } \bar{a}_{n(k)s} < 0 \right\}$$

où k désigne l'ensemble des indices correspondant aux variables de base.

Il convient d'observer les faits suivants.

- si le maximum est 0, x_s atteint sa borne inférieure avant qu'une variable de base atteigne une de ses bornes. Il n'y a pas de changement de base mais la valeur de x_s passe de v_s à 0 et, par suite, les valeurs des variables de base changent en accord avec la formule (10).
- si le maximum est:

$$v_s + \frac{v_r - \hat{x}_r}{-\bar{a}_{n(r)s}} \quad (11), \quad \bar{a}_{n(r)s} > 0$$

la variable de base x_r atteint la valeur v_r la première et sort de l'ensemble des variables de base en gardant cette valeur. Elle y est remplacée par x_s avec la valeur (11).

- si le maximum est:

$$v_s + \frac{\hat{x}_r}{\bar{a}_{n(r)s}} \quad (12), \quad \bar{a}_{n(r)s} < 0$$

la variable de base x_r atteint sa borne inférieure la première et sort de l'ensemble des variables de base avec cette valeur; elle y est remplacée par x_s avec la valeur (12).

- lorsque le maximum n'est pas unique, les trois opérations précédentes restent applicables. Une opération étant terminée, l'une au moins des variables de base x_k ($k \neq r$) a une valeur égale à une de ses bornes. Il s'agit d'un phénomène de dégénérescence qui peut être éliminé, tout comme pour l'algorithme primal du simplexe, en ordonnant les bases suivant un ordre lexicographique.

b) x_s croît à partir de sa borne inférieure 0

Dans ces conditions, en utilisant les notations définies ci-dessus, on a, pour toute variable x_k de base,

$$\tilde{x}_k = \tilde{x}_k - \bar{a}_{n(k)s} \tilde{x}_s$$

Comme on doit bien entendu avoir

$$\tilde{x}_s \leqslant v_s$$

et

$$0 \leqslant \hat{x}_k - \bar{a}_{n(k)s} \tilde{x}_s \leqslant v_k,$$

pour toute variable de base x_k, la valeur critique de x_s vaut

$$\tilde{x}_s = \min_{k \in K} \left\{ v_s ; \frac{\hat{x}_k}{\bar{a}_{n(k)s}} \text{ si } \bar{a}_{n(k)s} > 0; \frac{v_k - \hat{x}_k}{-\bar{a}_{n(k)s}} \text{ si } \bar{a}_{n(k)s} < 0 \right\}$$

Il convient d'observer les faits suivants.

- si le minimum est v_s, la variable x_s atteint sa borne supérieure avant qu'une variable de base atteigne une de ses bornes. La base reste inchangée mais il faut affecter la valeur v_s à x_s et les valeurs $\hat{x}_k - \bar{a}_{n(k)s} v_s$ aux variables de base x_k.
- si le minimum est $\hat{x}_r / \bar{a}_{n(r)s}$ $(\bar{a}_{n(r)s} > 0)$, la variable de base x_r atteint sa borne inférieure la première et sort de l'ensemble des variables de base en gardant cette valeur. Elle y est remplacée par x_s avec la valeur $\hat{x}_r / \bar{a}_{n(r)s}$.
- si le minimum vaut $(v_r - \hat{x}_r) / (-\bar{a}_{n(r)s})$ $(\bar{a}_{n(r)s} < 0)$, la variable de base x_r atteint sa borne supérieure la première et sort de l'ensemble des variables de base avec cette valeur. Elle y est remplacée par x_s avec la valeur $(v_r - \hat{x}_r) / (-\bar{a}_{n(r)s})$.

- lorsque le minimum est atteint pour plusieurs valeurs, les trois opérations précédentes restent applicables. Une opération étant terminée, l'une au moins des variables a une valeur égale à une de ses bornes. A nouveau, ce problème de dégénérescence peut être résolu en ordonnant les bases suivant un ordre lexicographique.

Afin d'alléger l'écriture de l'algorithme primal du simplexe modifié, adoptons la convention suivante.

CONVENTION V.2.2.4. Etant donné une base B, nous représentons le PLS

$$\min z = \Sigma_{j=1}^{n} c_j x_j$$

$$\text{s.c. } \Sigma_{j=1}^{n} a_{ij} x_j = b_i \qquad i = 1, \ldots, m$$

$$0 \leqslant x_j \leqslant v_j \qquad j = 1, \ldots, n$$

sous forme d'un tableau, appelé tableau du simplexe généralisé, construit comme suit:

z	$\bar{c}_1$	...	$\bar{c}_n$
	$\bar{a}_{11}$	...	$\bar{a}_{1n}$
	⋮		⋮
	$\bar{a}_{m1}$	...	$\bar{a}_{mn}$
	$\hat{x}_1$	...	$\hat{x}_n$
	v_1	...	v_n

les symboles $\hat{x}_i$ représentent les valeurs courantes des variables.

Enonçons enfin l'algorithme primal du simplexe modifié.

ALGORITHME PRIMAL DU SIMPLEXE MODIFIE V.2.2.5. Soit x une solution de base réalisable généralisée de base correspondante B.

(1). Mettre l'équivalent canonique associé à la base B et à la solution x sous la forme du tableau du simplexe généralisé en multipliant éventuellement certaines lignes par (-1) de façon à ce qu'elles constituent des vecteurs lexicographiquement positifs, excepté peut-être la première.

(2). Si $z = -\infty$, alors arrêter les calculs: la fonction économique n'a pas de minimum fini. Sinon, aller en (3).

(3). Si tous les coûts $\bar{c}_j$ relatifs aux variables indépendantes vérifient la condition $\bar{c}_j \geqslant 0$ si $x_j = 0$, $\bar{c}_j \leqslant 0$ si $x_j = v_j$, arrêter les calculs: la solution de base réalisable généralisée actuelle est minimale. Sinon, aller à (4).

(4). Sélectionner l'un des $\bar{c}_j$, soit $\bar{c}_s$, tel que $\bar{c}_j < 0$ si $x_j = 0$ ou $\bar{c}_j > 0$ si $x_j = v_j$ (critère d'entrée) et aller en (5).

(5). Si $x_s = v_s$ alors calculer

$$\tilde{x}_s = \max_{k \in K} \{0;\ v_s + \frac{v_k - \hat{x}_k}{-\bar{a}_{n(k)s}} \text{ si } \bar{a}_{n(k)s} > 0 ;$$

$$v_s + \frac{\hat{x}_k}{\bar{a}_{n(k)s}} \text{ si } \bar{a}_{n(k)s} < 0\}$$

(les notations sont celles des pages précédentes) Si ce maximum est atteint pour plusieurs indices de k, alors, en faisant correspondre à x_s le vecteur $(0, \ldots, 0)'$ et à la variable de base x_ℓ le vecteur

$$\frac{-\bar{c}_s}{\bar{a}_{n(\ell)s}} (\bar{a}_{n(\ell)1}, \ldots, \bar{a}_{n(\ell)n})$$

sélectionner, parmi les variables réalisant ce maximum, la variable x_r correspondant au vecteur lexicographiquement inférieur à tous les autres. Sinon, soit x_r la variable réalisant ce maximum.

Effectuer

$$\hat{x}_s := \tilde{x}_s$$

Pour toute variable de base x_k,

$$\hat{x}_k := \hat{x}_k + \bar{a}_{n(k)s}\,(v_s - \tilde{x}_s)$$

$$z := z + (\tilde{x}_s - v_s)\bar{c}_s$$

Si $x_r \neq x_s$ alors
pivoter le sous tableau du tableau du simplexe généralisé correspondant aux éléments du tableau du simplexe avec $\bar{a}_{n(r)s}$ pour pivot.

Si $x_s = 0$, alors

calculer

$$\tilde{x}_s = \min_{k \in K}\left\{v_s;\ \frac{\hat{x}_k}{\bar{a}_{n(k)s}} \text{ si } \bar{a}_{n(k)s} > 0;\right.$$

$$\left.\frac{v_k - \hat{x}_k}{-\bar{a}_{n(k)s}} \text{ si } \bar{a}_{n(k)s} < 0\right\}$$

Si ce minimum est atteint pour plusieurs indices de k, alors en faisant correspondre aux variables x_s et de base les mêmes vecteurs que ci-avant, choisir parmi les variables réalisant ce minimum celle, soit x_r, correspondant au vecteur lexicographiquement inférieur à tous les autres. Sinon, soit x_r la variable réalisant ce maximum.

Effectuer

$\hat{x}_s := \tilde{x}_s$

Pour toute variable de base x_k

$$\hat{x}_k := \hat{x}_k - \bar{a}_{n(k)s}\,\tilde{x}_s$$

$z := z + \bar{c}_s \tilde{x}_s$

Si $x_r \neq x_s$

pivoter le sous tableau du simplexe généralisé correspondant aux éléments du tableau du simplexe avec $\bar{a}_{n(r)s}$ *pour pivot.*

Aller en (2).

V.2.3. Formes révisées des algorithmes V.2.1.3. et V.2.2.5.

Tel que nous l'avons exposé, l'algorithme primal du simplexe nécessite à chaque itération le calcul complet d'un nouveau tableau du simplexe soit le calcul de (m+1)*(n+1) éléments distincts. En réalité, beaucoup d'entre eux ne servent pas, puisque seule la partie de la ligne des coûts relatifs limitée aux variables indépendantes détermine le choix de la nouvelle variable de base et puisque, sauf cas de dégénérescence extraordinaire, un nombre faible des éléménts des lignes 2 à m+1 fournit la variable à sortir de la base. Cette constatation est d'autant plus renforcée par l'étude du nombre d'éléments nuls du tableau du simplexe: il représente selon Orchard-Hays ([32]) pour des PLS de grande taille, jusqu'à 99% du nombre total d'éléments.

Des propos semblables peuvent être tenus au sujet de l'algorithme du simplexe modifié.

Les algorithmes suivants tiennent compte de ces faits. Ils évitent de stocker en mémoire une masse d'information trop importante en ne mémorisant que l'inverse de la base courante et les données du PLS original, à la limite sur mémoire secondaire. Par suite, des problèmes de plus grande taille pourront être traités. Une autre conséquence réside dans l'utilisation de sous-programmes destinés à augmenter la précision des calculs. Enfin, lorsque le tableau du simplexe initial contient un grand nombre d'éléments nuls, moins de calculs seront effectués.

Enonçons la forme revisée de l'algorithme primal du simplexe V.2.1.3.

ALGORITHME PRIMAL DU SIMPLEXE REVISE V.2.3.1. Soit une solution de base réalisable de base correspondante B.

(1). Calculer le vecteur des multiplicateurs du simplexe $\pi = c_B B^{-1}$ *et le vecteur* $\bar{c}_N = c_N - \pi N$ *des coûts relatifs correspondant aux variables indépendantes. (N.B.: on peut les calculer en mémoire centrale, composante par composante)*

(2). Si $\bar{c}_N \geq \underline{0}$ *, arrêter les calculs: la solution de base réalisable actuelle est minimale. Sinon, aller à (3).*

(3). Sélectionner, dans $\bar{c}_N$*, un coût relatif* $\bar{c}_s < 0$ *(le plus négatif, par exemple) et calculer le vecteur*

$$\bar{P}_s = B^{-1} P_s = (\bar{a}_{1s}, \ldots, \bar{a}_{ms})'$$

Aller à l'étape (4).

(4). Si $\bar{a}_{is} \leq 0$ *pour tout* $i \in \{1, \ldots, m\}$ *arrêter les calculs: la fonction économique n'a pas de minimum fini. Sinon, calculer*

$$\min_{i:\bar{a}_{is}>0} \frac{\bar{b}_i}{\bar{a}_{is}} .$$

Si ce minimum se présente pour une seule valeur de i, soit $i = r$ *alors poser* $\alpha = r$.

Sinon, déterminer l'ensemble I_1 des indices pour lesquels le minimum est atteint et pour tout $i \in I_1$, calculer $\bar{a}_{i1}$. Calculer ensuite $\min_{i \in I_1} \bar{a}_{i1} / \bar{a}_{is}$. Si ce minimum se présente pour une seule valeur de i, poser α égal à cette valeur. Sinon, déterminer l'ensemble $I_2 \subset I_1$ des indices pour lequel ce minimum est atteint et calculer pour tout $i \in I_2$, $\bar{a}_{i2}$. Calculer ensuite $\min_{i \in I_2} \bar{a}_{i2} / \bar{a}_{is}$, ... Et ainsi de suite jusqu'à l'obtention d'un ensemble I_ℓ réduit à un singleton, soit $I_\ell = \{x_r\}$. Poser alors $\alpha = r$. Aller en (5).

(5). Calculer l'inverse $\overset{\gamma}{B}^{-1}$ de la nouvelle base $\overset{\gamma}{B}$ (obtenue en remplaçant dans l'ensemble des variables de base, la variable correspondant à la r^{eme} contrainte par la variable x_s, ainsi que la nouvelle solution de base réalisable $(x_{\overset{\gamma}{B}} = \overset{\gamma}{B}^{-1} b = \overset{\gamma}{b},\ x_N = \underline{0})$.
Aller en (1) en substituant $\overset{\gamma}{B}$ à B.

REMARQUE V.2.3.2. Le calcul de $\overset{\gamma}{B}^{-1}$ peut se faire grâce aux opérations de pivotage habituelles mais sur une matrice de taille plus petite ou encore grâce à l'égalité

$$\overset{\gamma}{B}^{-1} = \begin{pmatrix} 1 & 0 & \dots & 0 & -\bar{a}_{1s}/\bar{a}_{rs} & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & -\bar{a}_{2s}/\bar{a}_{rs} & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 1 & -\bar{a}_{r-1,s}/\bar{a}_{rs} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1/\bar{a}_{rs} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & -\bar{a}_{r+1,s}/\bar{a}_{rs} & 1 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 & -\bar{a}_{ms}/\bar{a}_{rs} & 0 & \dots & 1 \end{pmatrix} B^{-1}$$

L'algorithme primal du simplexe modifié V.2.2.5. admet une forme révisée semblable. Nous laissons au lecteur le soin de la rédiger.

Comparons à présent les performances des algorithmes V.2.1.3. et V.2.3.1. sur base du nombre d'opérations arithmétiques élémentaires à effectuer lors de l'application de chacun d'eux. Plus précisément, nous ne retenons que les opérations de multiplication et de division car ce sont celles qui demandent le plus de temps aussi bien manuellement qu'automatiquement.

Désignons par p la proportion des éléments non nuls du tableau du simplexe initial et admettons qu'elle est identique à la moyenne des proportions des éléments non nuls dans chaque colonne. Supposons aussi que nous n'effectuons aucune opération si l'un des opérandes est nul. Dans ces conditions, le nombre maximum d'opérations exécutées lors d'une itération de l'algorithme V.2.3.1.

- pour le calcul de π est m^2
- pour le calcul de $\bar{c}_N$ est $p(n-m)$
- pour le calcul de $\bar{P}_s$ est mp
- pour l'exécution de *(4)* est $(n+1)m + nmp$
- pour le calcul de $\tilde{B}^{-1}$ est m^2
- pour le calcul de $\tilde{b}$ est m^2

soit au total $3m^2 + np(m+1) + (n+1)m$. Par ailleurs, le nombre maximum d'opérations exécutées lors d'une itération de l'algorithme V.2.1.3.

- pour l'exécution de *(4)* est $(n+1)m$
- pour le calcul du nouveau tableau du simplexe est $(n-m+1)(m+1)$

soit au total $2nm + m + n + 1 - m^2$.

On peut, dès lors, raisonnablement considérer que l'algorithme primal du simplexe révisé V.2.3.1. exige moins d'opérations que l'algorithme primal du simplexe V.2.1.3. lorsque la proportion p des éléments non nuls du tableau du simplexe initial est telle que

$$3m^2 + np(m+1) + (n+1)m < 2nm + m + n + 1 - m^2$$

c'est-à-dire telle que

$$p < \frac{nm + n + 1 - 4m^2}{n(m+1)}$$

Cette dernière inégalité permet de décider de l'opportunité d'utiliser l'un ou l'autre algorithme.

Une comparaison semblable peut, bien sûr, être faite pour l'algorithme V.2.2.5. et l'algorithme révisé qui en découle.

V.2.4. La méthode des deux phases et la méthode des pénalités

L'application de l'algorithme primal du simplexe ou de l'algorithme primal du simplexe révisé nécessite la connaissance d'une solution de base réalisable initiale. Une telle solution n'est malheureusement pas souvent connue dans la pratique. De plus, le système des contraintes peut être incompatible ou redondant.

Il est donc utile de disposer d'une méthode permettant à la fois de dépister les redondances et incompatibilités éventuelles et de calculer une solution de base réalisable initiale. Diverses méthodes ont été proposées ([10] , [21] , [23] , [30] , [35]) parmi lesquelles nous avons sélectionné la méthode des deux phases et la méthode des pénalités.

Exposons tout d'abord la première dans le cadre de l'algorithme primal du simplexe. L'adaptation à l'algorithme primal du simplexe révisé est immédiate.

ALGORITHME DES DEUX PHASES V.2.4.1. Soit le PLS

$$\min z = cx$$

$$\text{s.c. } \sum_{j=1}^{n} a_{ij} x_j = b_i \quad , \quad i = 1, \ldots, m$$

$$x_j \geq 0 \quad , \qquad j = 1, \ldots, n$$

(1). Lui associer le PLS auxiliaire

$$\min w = \sum_{i=1}^{m} x_{n+1}$$

$$\text{s.c.} \quad \delta_i \sum_{j=1}^{n} a_{ij}x_j + x_{n+i} = \delta_i b_i \qquad i = 1, \ldots, m$$

$$x_j \geq 0 \qquad j = 1, \ldots, n, n+1, \ldots, n+m$$

$$\text{où } \delta_i = \begin{cases} 1 \text{ si } b_i \geq 0 \\ -1 \text{ sinon} \end{cases}$$

(2). Appliquer l'algorithme primal du simplexe V.2.1.3. au PLS auxiliaire en prenant pour solution de base réalisable initiale la solution définie par

$$x_i = 0, \qquad i = 1, \ldots, n$$

$$x_{n+i} = \delta_i b_i, \quad i = 1, \ldots, m$$

jusqu'à ce que l'une des trois possibilités suivantes se présente :

(i) $\min w > 0$:

cela signifie que les contraintes du PLS original sont incompatibles.

(ii) $\min w = 0$ et aucune des variables $x_{n+1}, \ldots, x_{n+m}$ n'est une variable de base correspondant à ce minimum.

Dans ce cas, la solution réalisable trouvée est une solution de base réalisable du PLS original. De plus, le système

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \qquad i = 1, \ldots, m$$

est non redondant. Il reste, dès lors, à remplacer dans le dernier tableau du simplexe, la ligne relative à w par la ligne relative à z, à amputer ce tableau des colonnes relatives à $x_{n+1}, \ldots, x_{n+m}$ et à appliquer au tableau ainsi obtenu l'algorithme primal du simplexe.

(iii) min w = 0 et au moins une des variables artificielles $x_{n+1}, \ldots, x_{n+m}$ est une variable de base correspondant à ce minimum:

Dans ce cas, la solution réalisable trouvée est une solution réalisable de base ou non du PLS original. Il reste à remplacer dans le dernier tableau du simplexe, la ligne relative à w par la ligne relative à z, à amputer ce tableau des colonnes relatives aux variables $x_{n+1}, \ldots, x_{n+m}$ qui ne sont pas de base ainsi que les lignes ℓ_i telles qu'on a, pour tout $j \in \{1, \ldots, n\}$, $\bar{a}_{ij} = 0$ et à appliquer l'algorithme primal du simplexe modifié comme suit. Soit, lors d'une itération, x_s la variable sélectionnée pour entrer dans l'ensemble des variables de base.

- *si $\bar{a}_{is} \leq 0$, pour tout i correspondant aux variables artificielles de base et si $\bar{a}_{is} < 0$ pour au moins un de ces i, faire sortir de l'ensemble des variables de base n'importe laquelle des variables artificielles telles que $\bar{a}_{is} < 0$. Amputer ensuite le nouveau tableau du simplexe obtenu de la colonne relative à cette variable sortante.*
- *sinon, appliquer normalement le critère de sortie de l'algorithme primal du simplexe.*

Enonçons à présent l'algorithme des pénalités dans le cadre de l'algorithme primal du simplexe, l'adaptation à l'algorithme primal du simplexe révisé étant, de nouveau, immédiate.

ALGORITHME DES PENALITES V.2.4.2. Soit le PLS

$\min z = c x$

$$\text{s.c. } \sum_{j=1}^{n} a_{ij} x_j = b_i \qquad i = 1, \ldots, m$$

$$x_j \geqslant 0 \qquad j = 1, \ldots, n$$

(1). Appliquer l'algorithme primal du simplexe V.2.1.3. au PLS augmenté

$$\min z' = \sum_{i=1}^{n} c_i x_i + M \sum_{i=1}^{m} x_{n+i}$$

$$\text{s.c. } \delta_i \sum_{j=1}^{n} a_{ij} x_j + x_{n+i} = \delta_i b_i \qquad i = 1, \ldots, m$$

$$x_j \geqslant 0 \qquad j = 1, \ldots, n, n+1, \ldots, n+m$$

où M désigne une constante positive arbitrairement grande (c'est-à- dire supérieure à tout nombre fini auquel elle sera comparée au cours des calculs) et où les coefficients δ_i $(i = 1, \ldots, m)$ sont définis par

$$\delta_i = \begin{cases} 1 & \text{si } b_i \geqslant 0 \\ -1 & \text{sinon,} \end{cases}$$

en prenant comme solution de base réalisable la solution

$$x_j = 0 \qquad j = 1, \ldots, n$$

$$x_{n+1} = \delta_i b_i \qquad i = 1, \ldots, m$$

Aller en (2).

(2). Si la fonction économique z' n'a pas de minimum fini, il en est de même pour z.

Sinon,

si certaines des variables $x_{n+m}, \ldots, x_{n+m}$ ont une valeur strictement positive, alors le PLS original n'a pas de solution réalisable;

sinon, la solution réalisable minimum obtenue pour le PLS augmenté est aussi solution réalisable minimum du PLS original.

De même, l'algorithme primal du simplexe modifié et l'algorithme primal du simplexe modifié révisé nécessitent la connaissance d'une solution de base réalisable généralisée. La méthode des deux phases se généralise aisément à ces cas. Nous ne la présentons que pour l'algorithme primal du simplexe modifié, l'extension à l'algorithme primal du simplexe modifié révisé étant évidente.

ALGORITHME DES DEUX PHASES V.2.4.3. Soit le PLS

$$\min z = \sum_{i=1}^{n} c_i x_i$$

$$\text{s.c. } \sum_{j=1}^{n} a_{ij} x_j = b_i \ , \quad i = 1, \ldots, m$$

$$0 \leqslant x_j \leqslant v_j \ , \quad j = 1, \ldots, n$$

(1). Lui associer le PLS auxiliaire

$$\min w = \sum_{i=1}^{n} x_{n+i}$$

$$\text{s.c. } \delta_i \sum_{j=1}^{n} a_{ij} x_j + x_{n+i} = \delta_i b_i \ , \quad i = 1, \ldots, m$$

$$0 \leqslant x_j \leqslant v_j \ , \quad j = 1, \ldots, n$$

$$0 \leqslant x_{n+i} \leqslant +\infty, \quad i = 1, \ldots, m$$

$$\text{où } \delta_i = \begin{cases} 1 & \text{si } b_i \geqslant 0 \\ -1 & \text{sinon} \end{cases}$$

(2). Appliquer l'algorithme primal du simplexe modifié V.2.2.5. au PLS auxiliaire en prenant pour solution de base réalisable généralisée la solution définie par

$$x_i = 0 \qquad i = 1, \ldots, n$$

$$x_{n+i} = \delta_i b_i \ , \quad i = 1, \ldots, m$$

jusqu'à ce que l'une des trois possibilités suivantes se présente:

(i) $\min w > 0$

Cela signifie que les contraintes du PLS original sont incompatibles .

(ii) $\min w = 0$ et aucune des variables $x_{n+1}, \ldots, x_{n+m}$ n'est une variable de base correspondant à ce minimum.

Dans ce cas, la solution réalisable généralisée trouvée est une solution de base réalisable généralisée du PLS original. De plus, le système

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \quad , \quad i = 1, \ldots, m$$

est non redondant. Il reste à remplacer dans le dernier tableau du simplexe généralisé la ligne relative à w par celle relative à z, à amputer ce tableau des colonnes relatives aux variables $x_{n+1}, \ldots, x_{n+m}$ et à appliquer au tableau ainsi obtenu l'algorithme primal du simplexe modifié V.2.2.5.

(iii) $\min w = 0$ et une, au moins, des variables artificielles $x_{n+1}, \ldots, x_{n+m}$ est une variable de base correspondant à ce minimum.

Dans ce cas, la solution réalisable généralisée trouvée est une solution réalisable généralisée de base ou non du PLS original. Il faut remplacer dans le dernier tableau du simplexe généralisé la ligne relative à w par celle relative à z, amputer ce tableau des colonnes relatives aux variables artificielles non en base ainsi que les lignes ℓ_i $(i = 1, \ldots, m)$ telles qu'on ait, pour tout $j \in \{1, \ldots, n\}$, $\bar{a}_{ij} = 0$ et appliquer l'algorithme primal du simplexe modifié V.2.2.5. en respectant la règle suivante. Soit, lors d'une itération, x_s, la variable sélectionnée pour entrer dans l'ensemble des variables de base.

- *S'il existe un indice i correspondant à une variable artificielle de base tel que* $\bar{a}_{is} \neq 0$, *alors faire sortir une telle variable de l'ensemble des variables de base et amputer le nouveau tableau du simplexe généralisé de la colonne qui lui correspond.*
- *Sinon, appliquer normalement le critère de sortie de l'algorithme V.2.2.5.*

Enfin, la méthode des pénalités s'étend tout naturellement aux algorithmes V.2.2.5. et V.2.3.2.. Appliquée au premier, elle donne naissance à l'algorithme suivant:

ALGORITHME DES PENALITES V.2.4.4. Soit le PLS

$$\min z = \sum_{i=1}^{n} c_i x_i$$

$$\text{s.c.} \quad \sum_{j=1}^{n} a_{ij} x_j = b_i \,, \qquad i = 1, \ldots, m$$

$$0 \leqslant x_j \leqslant v_j \,, \qquad j = 1, \ldots, n$$

(1). Appliquer l'algorithme primal du simplexe modifié V.2.2.5. au PLS augmenté

$$\min z' = \sum_{i=1}^{n} c_i x_i + M \sum_{i=1}^{m} x_{n+i}$$

$$\text{s.c.} \quad \delta_i \sum_{j=1}^{n} a_{ij} x_j = \delta_i b_i \,, \qquad i = 1, \ldots, m$$

$$0 \leqslant x_j \leqslant v_j \,, \qquad j = 1, \ldots, n$$

$$0 \leqslant x_{n+i} \leqslant +\infty \,, \qquad i = 1, \ldots, m \,,$$

où M désigne une constante positive arbitrairement grande et où les coefficients δ_i $(i = 1, \ldots, m)$ *sont définis par*

$$\delta_i = \begin{cases} 1 & \text{si } b_i \geqslant 0 \\ -1 & \text{sinon,} \end{cases}$$

en prenant comme solution de base réalisable généralisée la solution

$$x_j = 0 \quad , \quad j = 1, \ldots, n$$

$$x_{n+i} = \delta_i b_i \quad , \quad i = 1, \ldots, m$$

Aller en (2).

(2). Si la fonction économique z' n'a pas de minimum fini, il en est de même pour z.

Sinon,

si certaines des variables $x_{n+1}, \ldots, x_{n+m}$ ont une valeur strictement positive, alors le PLS original n'a pas de solution réalisable généralisée;

sinon, la solution réalisable généralisée minimum obtenue pour le PLS augmenté est aussi solution réalisable généralisée minimum du PLS original.

Nous laissons au lecteur le soin de l'appliquer à l'algorithme V.2.3.2..

V.3. La méthode duale du simplexe

V.3.1. L'algorithme dual du simplexe

Introduisons dès à présent l'algorithme dual du simplexe. Son utilité apparaîtra plus clairement au paragraphe V.5..

ALGORITHME DUAL DU SIMPLEXE V.3.1.1. Soit B une base telle que les vecteurs $(\bar{c}_i, \bar{a}_{1i}, \ldots, \bar{a}_{mi})$, $i \in \{1, \ldots, n\}$ soient non nuls et lexicographiquement positifs.

(1). Construire le tableau du simplexe correspondant à B.

(2). Si on a, pour tout $i \in \{1, \ldots, m\}$, $\bar{b}_i \geq 0$ alors arrêter les calculs: la solution de base actuelle est réalisable et minimum.

Sinon, aller en (3).

(3). Sélectionner un des $\bar{b}_i < 0$, *par exemple, celui pour lequel* $\bar{b}_r = \min_{i:\bar{b}_i<0} \bar{b}_i$ *(critère de sortie)*

Aller en (4).

(4). Si tous les $\bar{a}_{rj}$ *correspondant aux variables indépendantes sont positifs ou nuls, arrêter les calculs: le PLS ne possède pas de solution réalisable. Sinon, aller en (5).*

(5). Sélectionner pour variable entrant dans l'ensemble des variables de base, celle dont l'indice s correspond au vecteur qui, parmi les vecteurs

$$\left(-\frac{\bar{c}_j}{\bar{a}_{rj}}, -\frac{\bar{a}_{1j}}{\bar{a}_{rj}}, \ldots, -\frac{\bar{a}_{mj}}{\bar{a}_{rj}}\right)$$

tels que $\bar{a}_{rj} < 0$ *est lexicographiquement inférieur à tous les autres. (critère d'entrée)*
Aller en (6).

(6). Pivoter le tableau du simplexe avec $\bar{a}_{rs}$ *pour pivot.*
Aller en (2).

REMARQUE V.3.1.2. D'une base B du PLS

$$\min z = \sum_{i=1}^{n} c_i x_i$$

$$\text{s.c.} \quad \sum_{j=1}^{n} a_{ij} x_j = b_i \quad (i = 1, \ldots, m)$$

$$x_j \geqslant 0 \quad , j = 1, \ldots, n$$

pour laquelle les coûts relatifs $\bar{c}_j$ sont tous positifs ou nuls, il est possible de déduire un PLS équivalent ainsi qu'une de ses bases vérifiant l'hypothèse de l'algorithme V.3.1.1.. De fait, il suffit d'ajouter au système des contraintes, l'inégalité

$$\sum_{j \in J} x_j \leqslant M$$

soit après addition d'une variable d'écart,

$$x_o + \sum_{j \in J} x_j = M$$

où J représente l'ensemble des indices correspondant aux variables indépendantes et M une constante positive arbitrairement grande. Le PLS original est dès lors équivalent au PLS

$$\min z = \sum_{i=1}^{n} c_i x_i$$

$$\text{s.c.} \quad x_o + \sum_{j \in J} x_j = M$$

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \,, \quad i = 1, \ldots, m$$

$$x_j \geqslant 0 \,, \quad j = 1, \ldots, n$$

Ce dernier admet pour base

$$\begin{bmatrix} 1 & 0 \\ 0 & B \end{bmatrix}$$

et on a, en s'y référant,

$$\bar{b} = \begin{bmatrix} M \\ B^{-1} b \end{bmatrix}$$

et, pour tout indice j correspondant à une variable indépendante,

$$\bar{c}_j = c_j - c_B B^{-1} P_j \geqslant 0.$$

Remarquons en outre qu'une variable d'un PLS à laquelle correspond le vecteur $(\bar{c}_\ell, \bar{a}_{1\ell}, \ldots, \bar{a}_{m\ell})$ nul n'intervient aucunement dans sa résolution. De fait, de l'égalité

$$0 = \begin{bmatrix} \bar{a}_{1\ell} \\ \cdot \\ \cdot \\ \cdot \\ \bar{a}_{m\ell} \end{bmatrix} = B^{-1} \begin{bmatrix} a_{1\ell} \\ \cdot \\ \cdot \\ \cdot \\ a_{m\ell} \end{bmatrix}$$

on déduit de suite que

$$0 = \begin{bmatrix} a_{1\ell} \\ \cdot \\ \cdot \\ \cdot \\ a_{m\ell} \end{bmatrix}$$

et l'on a, alors,

$$\bar{c}_\ell = c_\ell - c_B B^{-1} P_\ell = c_\ell$$

c'est-à-dire

$$c_\ell = 0.$$

Dès lors, le PLS original,

$$\min z = \Sigma_{i=1}^{n} c_i x_i$$

$$\text{s.c.} \quad \Sigma_{j=1}^{n} a_{ij} x_j = b_i \,, \quad i = 1, \ldots, m$$

$$x_j \geqslant 0 \,, \quad j = 1, \ldots, n$$

est équivalent au PLS

$$\min z = \sum_{\substack{i=1 \\ i \neq \ell}}^{n} c_i x_i$$

$$\text{s.c.} \quad \sum_{\substack{j=1 \\ j \neq \ell}}^{n} a_{ij} x_j = b_i \,, \quad i = 1, \ldots, m$$

$$x_j \geqslant 0 \,, \quad j = 1, \ldots, \ell-1, \ell+1, \ldots, m$$

V.3.2. Forme révisée de l'algorithme dual du simplexe

L'algorithme dual du simplexe admet bien sûr une forme révisée.

ALGORITHME DUAL DU SIMPLEXE REVISE V.3.2.1. Soit une base B d'inverse connu B^{-1} et telle que les vecteurs $(\bar{c}_i, \bar{a}_{1i}, \ldots, \bar{a}_{mi})$, $i \in \{1, \ldots, n\}$ soient non nuls et lexicographiquement positifs.

(1). Calculer le vecteur $\bar{b} = B^{-1} b$.

Si on a, pour tout $i \in \{1, \ldots, m\}$, $\bar{b}_i \geq 0$, *alors arrêter les calculs : la solution de base actuelle est réalisable et minimum.*

Sinon aller en (2).

(2). Sélectionner un des $\bar{b}_i < 0$, *par exemple celui pour lequel* $\bar{b}_r = \min_{i:\bar{b}_i<0} \bar{b}_i$ *(critère de sortie)*

Aller en (3).

(3). Calculer, pour tout indice j correspondant à une variable indépendante

$$\bar{a}_{rj} = \sum_{l=1}^{m} (B^{-1})_{rl} a_{lj} .$$

Si tous sont positifs, arrêter les calculs : le PLS ne possède pas de solution réalisable.

Sinon, calculer pour tout indice j tel que $\bar{a}_{rj} < 0$, *le coût relatif* $\bar{c}_j = c_j - c_B \bar{b}$ *et aller en (4).*

(4). Calculer

$$\min_{j:\bar{a}_{rj}<0} - \frac{\bar{c}_j}{\bar{a}_{rj}}$$

et déterminer l'ensemble J_1 *des indices pour lesquels ce minimum est atteint. Si* J_1 *se réduit à un singleton, soit* $J_1 = \{x_s\}$ *alors poser* $\alpha = s$. *Sinon, calculer, pour tout* $j \in J_1$, $\bar{a}_{1j}$ *et ensuite*

$$\min_{j \in J_1} - \frac{\bar{a}_{1j}}{\bar{a}_{rj}} .$$

Puis, déterminer l'ensemble $\mathcal{J}_2 \subset \mathcal{J}_1$ des indices pour lesquels ce minimum est atteint. Si $\mathcal{J}_2$ se réduit à un singleton, soit $\mathcal{J}_2 = \{x_s\}$, alors poser $\alpha = s$. Sinon, calculer, pour tout $j \in \mathcal{J}_2$, $\bar{a}_{2j}$ et ensuite

$$\min_{j \in \mathcal{J}_2} - \frac{\bar{a}_{2j}}{\bar{a}_{rj}} \cdot \quad \ldots$$

Et ainsi de suite, jusqu'à l'obtention d'un ensemble $\mathcal{J}_\ell$ réduit à un singleton, soit $\mathcal{J}_\ell = \{x_s\}$. Poser alors $\alpha = s$. (critère d'entrée)

Aller en (5).

(5). *Calculer l'inverse $\mathcal{B}^{-1}$ de la nouvelle base $\mathcal{B}$ obtenue en remplaçant dans l'ensemble des variables de base la variable correspondant à la r^{eme} contrainte par la variable x .*

Aller en (1) en substituant $\mathcal{B}$ à B.

V.4. Réinversion de la base

L'application de la méthode primale ou duale du simplexe requiert à chaque itération la connaissance de l'inverse de la base courante B. Comme nous l'avons déjà signalé en V.2.3.2., celle-ci est généralement obtenue par pivotages successifs ou grâce à des produits matriciels. Malheureusement, des erreurs d'arrondi apparaissent à chaque itération et se propagent d'itération en itération. Par suite, il arrive, comme le montre le calcul direct $B^{-1}B$, que les pertes de précision deviennent trop importantes. Il devient alors nécessaire de procéder à une inversion directe et aussi précise que possible de la base.

L'algorithme le plus couramment employé consiste à utiliser la forme produit de l'inverse c'est-à-dire à rechercher l'inverse de la base, supposée de dimension m, sous la forme d'un produit de m matrices élémentaires. Afin de le découvrir, établissons tout d'abord les propositions suivantes.

PROPOSITION V.4.1. Soient $\{c_i : 1 \leqslant i \leqslant m\}$ et $\{\ell_i : 1 \leqslant i \leqslant m\}$ les colonnes et les lignes d'une matrice invertible. Soit en outre $(\nu_1, \ldots, \nu_m)$ une permutation de $(1, \ldots, m)$. On a,

$$(c_1, \ldots, c_m)^{-1} = (e_{\nu_1}, \ldots, e_{\nu_m})(c_{\nu_1}, \ldots, c_{\nu_m})^{-1}$$

et

$$\begin{pmatrix} \ell_1 \\ \cdot \\ \cdot \\ \cdot \\ \ell_m \end{pmatrix}^{-1} = \begin{pmatrix} \ell_{\nu_1} \\ \cdot \\ \cdot \\ \cdot \\ \ell_{\nu_m} \end{pmatrix}^{-1} \begin{pmatrix} e_{\nu_1} \\ \cdot \\ \cdot \\ \cdot \\ e_{\nu_m} \end{pmatrix}$$

Démonstration. C'est immédiat. ///

PROPOSITION V.4.2. Pour toute matrice $C = (c_1, \ldots, c_{i-1}, c_i, c_{i+1}, \ldots, c_m)$ et tout vecteur $x = \sum_{j=1}^{m} \lambda_j c_j$ tel que $\lambda_i \neq 0$, l'inverse de la matrice $(c_1, \ldots, c_{i-1}, x, c_{i+1}, \ldots, c_m) = C'$ obtenue en substituant au vecteur colonne c_i le vecteur x est donné par l'égalité

$$C'^{-1} = \begin{pmatrix} 1 & 0 & \ldots & 0 & -\lambda_1/\lambda_i & 0 & \ldots & 0 \\ \cdot & \cdot & & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot & & \cdot \\ 0 & 0 & \ldots & 1 & -\lambda_{i-1}/\lambda_i & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & 1/\lambda_i & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & -\lambda_{i+1}/\lambda_i & 1 & \ldots & 0 \\ \cdot & \cdot & & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & \cdot & \cdot & & \cdot \\ 0 & 0 & \ldots & 0 & -\lambda_m/\lambda_i & 0 & \ldots & 1 \end{pmatrix} C^{-1}$$

Démonstration. C'est immédiat. ///

Le résultat précédent peut servir à calculer de proche en proche l'inverse de toute matrice invertible. De fait, si $B = (b_1, \ldots, b_m)$ désigne une telle matrice de dimension m, il suffit,

en supposant la proposition V.4.2. applicable à chaque pas, de remplacer successivement les m vecteurs unités de dimension m : $e_1, \ldots, e_m$ par les vecteurs colonnes $b_1, \ldots, b_m$. On obtient alors

$$B^{-1} = J_m \ldots J_1 \mathbb{I}_m$$

c'est-à-dire l'inverse de B comme le produit de m matrices de dimension m.

Cependant, afin d'augmenter la précision des calculs et de diminuer le volume de ceux-ci, il est avantageux de substituer les colonnes dans un ordre différent et même à une place différente. Ainsi, à la t^{eme} itération, la $p(t)^{eme}$ colonne de $\mathbb{I}_m$ sera remplacée par la colonne $b_{k(t)}$ de B. Dans ces conditions, on obtient de même une suite de matrices invertibles $B_{(t)}$, $t = 1, \ldots, m$ vérifiant les relations

$$B_{(0)}^{-1} = \mathbb{I}_m ,$$

$$B_{(1)}^{-1} = J_{p(1)} B_{(0)}^{-1} = J_{p(1)} ,$$

$$\vdots$$

$$B_{(t)}^{-1} = J_{p(t)} B_{(t-1)}^{-1} = J_{p(t)} \ldots J_{p(1)} ,$$

$$\vdots$$

$$B^{-1} = B_m^{-1} = J_{p(m)} B_{(m-1)}^{-1} = J_{p(m)} \ldots J_{p(1)} ,$$

où on a posé, pour tout $t \in \{1, \ldots, m\}$,

$$J_{p(t)} = (e_1, \ldots, e_{p(t)-1}, v_{p(t)}, e_{p(t)+1}, \ldots, e_m) \qquad (10) ,$$

$$v_{p(t)} = (v_1^{(t)}, \ldots, v_{p(t)-1}^{(t)}, v_{k(t)}^{(t)}, v_{p(t)+1}^{(t)}, \ldots, v_m^{(t)})' ,$$

$$v_i^{(t)} = \frac{-y_{i,k(t)}^{(t-1)}}{y_{p(t),k(t)}^{(t-1)}} , \quad i \neq p(t) ,$$

$$v_{p(t)}^{(t)} = \frac{1}{y_{p(t),k(t)}^{(t-1)}} ,$$

$$y_{k(t)}^{(t-1)} = (y_{1,k(t)}^{(t-1)}, \ldots , y_{p(t),k(t)}^{(t-1)}, \ldots , y_{m,k(t)}^{(t-1)})' ,$$

$$= B_{(t-1)}^{-1} b_{k(t)} .$$

Afin de calculer aisément les matrices $Y_{p(i)}$ $(i = 1, \ldots ,m)$, introduisons pour chaque colonne b_k de B les m vecteurs colonnes $y_k^{(t)}$ $(t = 1, \ldots ,m)$ définis comme suit :

$$y_k^{(0)} = B_{(0)}^{-1} b_k = b_k ,$$

$$\vdots$$

$$y_k^{(t)} = B_{(t)}^{-1} b_k ,$$

$$\vdots$$

$$y_k^{(m)} = B_{(m)}^{-1} b_k .$$

Ils vérifient les égalités

$$y_k^{(t)} = J_{p(t)} B_{(t-1)}^{-1} b_k = J_{p(t)} y_k^{(t-1)} \quad , \quad \forall\, k,t \in \{1, \ldots ,m\} \quad ,$$

c'est-à-dire les égalités

$$y_{i,k}^{(t)} = y_{i,k}^{(t-1)} + y_{p(t),k}^{(t-1)} v_i^{(t)} \quad , \quad \forall\, k,t \in \{1, \ldots ,m\}$$
$$\forall\, i \in \{1, \ldots ,m\} \setminus \{p(t)\} \quad ,$$

$$y_{p(t),k}^{(t)} = y_{p(t),k}^{(t-1)} v_{p(t)}^{(t)} \quad \forall\, k,t \in \{1, \ldots ,m\}$$

et dès lors les implications

$$\forall\, k,t \in \{1, \ldots ,m\} \quad (y_{p(t),k}^{(t-1)} = 0 \Longrightarrow y_k^{(t)} = y_k^{(t-1)}).$$

Il s'ensuit que seuls les vecteurs $y_k^{(t)}$ $(k = 1, \ldots ,m)$ correspondant

à un vecteur $y_k^{(t-1)}$ tel que $y_{p(t),k}^{(t-1)} \neq 0$ seront calculés à la t^{eme} itération ($t = 1, \ldots, m$).

Notons encore que puisqu'on a, pour tout $t \in \{1, \ldots, m\}$, $J_{p(t)} y_{k(t)}^{(t-1)} = e_{p(t)}$, il vient à l'itération t, pour tout $i \leqslant t$,

$$y_{k(i)}^{(q)} = e_{p(i)} \quad \forall\, q : i \leqslant q \leqslant m .$$

Dès lors, la matrice $Y^{(t)} = (y_{k(1)}^{(t)}, \ldots, y_{k(m)}^{(t)})$ définie à la t^{eme} itération a la structure

$$Y^{(t)} = (e_{p(1)}, \ldots, e_{p(t)}, y_{k(t+1)}^{(t)}, \ldots, y_{k(m)}^{(t)}) \qquad (11) ;$$

en particulier, $Y^{(m)}$ est la matrice unité $\mathbb{1}_m$.

En conclusion, en supposant les permutations $(p(1), \ldots, p(m))$ et $(k(1), \ldots, k(m))$ de $(1, \ldots, m)$ connues et, en posant, pour tout $i \in \{1, \ldots, m\}$, $p^{-1}(i)$ l'entier $\ell \in \{1, \ldots, m\}$ tel que $p(\ell) = i$, la méthode de calcul de l'inverse de B s'écrit

ALGORITHME V.4.3.

t := 1;
Tant que $t \leqslant m$ *répéter*
  Calculer $J_{p(t)}$ *par les formules (10);*
  Pour tout k $\{k(t+1), \ldots, k(m)\}$, *calculer* $y_k^{(t)}$ *à partir de* $y_k^{(t-1)}$;
  t := t+1;
Effectuer
  $B^{-1} := (e_{k(p^{-1}(1))}, \ldots, e_{k(p^{-1}(m))})\, J_{p(m)} \cdots J_{p(1)}$.

Reste à présent à choisir au mieux la séquence des couples $(p(t),k(t))$, $t \in \{1, \ldots, m\}$. Bien entendu, ce choix constitue dans la pratique le point fondamental puisque c'est sur lui que repose la qualité des résultats fournis par la réinversion.

Généralement on procède en deux phases :

1. phase de triangularisation

Dans la première, on tente de triangulariser la matrice B ou du moins une partie de celle-ci. A cet effet, on permute ses

lignes et ses colonnes de façon à obtenir une matrice du type

$$B' = \begin{bmatrix} T & 0 & 0 \\ X & Y & 0 \\ Z & W & T' \end{bmatrix}$$

où T et T' sont deux matrices carrées triangulaires inférieures.

Pour y parvenir, appliquons tout d'abord l'algorithme suivant à B.

ALGORITHME CONSTRUCTION—T' V.4.4.

S'il existe une colonne n'ayant qu'un élément non nul alors
- *Sélectionner une telle colonne : soit b_k d'élément non nul $b_{i,k}$;*
- *Permuter la k^{eme} colonne et la dernière colonne;*
- *Permuter la i^{eme} ligne et la dernière ligne;*
- *Appliquer CONSTRUCTION_T' au sous tableau obtenu en supprimant la dernière ligne et la dernière colonne;*

Sinon arrêter.

Il délivre après exécution un tableau de la forme

$$\begin{bmatrix} \alpha & 0 \\ * & T' \end{bmatrix}.$$

Il reste donc, pour obtenir le résultat attendu, à faire apparaître la matrice T. On y arrive en appliquant l'algorithme CONSTRUCTION_T à la matrice α.

ALGORITHME CONSTRUCTION_T V.4.5.

S'il existe une ligne n'ayant qu'un élément non nul alors
- *Sélectionner une telle ligne : soit l_k d'élément non nul $l_{i,k}$;*
- *Permuter la k^{eme} ligne et la 1^{ere} ligne;*
- *Permuter la i^{eme} colonne et la 1^{ere} colonne;*
- *Appliquer CONSTRUCTION_T au sous-tableau obtenu en supprimant la 1^{ere} ligne et la 1^{ere} colonne;*

Sinon arrêter.

La matrice obtenue, après exécution, a bien la forme désirée. On remarquera, en outre, que tous les éléments de la matrice Y sont non nuls. Ceci implique que parmi les matrices possédant la forme adéquate, la matrice obtenue est telle que les sous-matrices T et T' sont de dimension maximum. En ce sens, il s'agit d'une "triangularisation partielle maximum" de B.

2. phase d'inversion

Dans la seconde, on procède à l'inversion proprement dite de B ou, ce qui revient au même, à la recherche des suites $(k(t))_{t\in\{1, \ldots ,m\}}$ et $(p(t))_{t\in\{1, \ldots ,m\}}$.

Afin d'alléger l'exposé, adoptons tout d'abord la convention suivante.

CONVENTION V.4.6. Dans la suite de ce travail, nous désignons par

B'

la matrice obtenue à partir de la matrice B comme décrit ci-dessus. En outre, nous notons

$\kappa(t)$ (resp. $\pi(t)$), $t \in \{1, \ldots ,m\}$

l'indice de la colonne (resp. de la ligne) de B correspondant à la t^{eme} colonne (resp. ligne) de B' et par

δ (resp. δ')

la dimension de T (resp. T').

La proposition suivante en découle.

PROPOSITION V.4.7. Soient $(\nu(1), \ldots ,\nu(m))$ et $(\mu(1), \ldots ,\mu(m))$ deux permutations de la suite $(1, \ldots ,m)$. Soient en outre, pour tout $i \in \{1, \ldots ,m\}$, b_i et b'_i respectivement la i^{eme} colonne de B et de B'. Si, pour tout $t \in \{1, \ldots ,m\}$, la matrice $Y(t)$ (resp. $Y'(t)$) est obtenue à partir de la matrice $Y^{(t-1)}$ (resp. $Y'^{(t-1)}$) par pivotage avec $y^{(t-1)}_{\pi(\mu(t)),\kappa(\nu(t))}$ (resp. $y'^{(t-1)}_{\mu(t),\nu(t)}$) pour pivot et si on a $Y^{(0)} = (b_{\nu(1)}, \ldots ,b_{\nu(m)})$ et $Y'^{(0)} = (b'_{\nu(1)}, \ldots ,b'_{\nu(m)})$, alors, pour tout $t \in \{1, \ldots ,m\}$, on a

$$Y'^{(t)} = \begin{bmatrix} e'_{(1)} \\ \vdots \\ e'_{(m)} \end{bmatrix} Y^{(t)} \, (e_{k(1)}, \ldots, e_{k(m)})$$

Démonstration. C'est immédiat. ///

Elle signifie qu'en procédant comme indiqué dans les hypothèses, les permutations des lignes et des colonnes existant entre B et B' se reproduisent entre chaque $Y^{(t)}$ et $Y'^{(t)}$. Par conséquent, nous pouvons déterminer les différents pivots et la séquence des couples $(k(t),p(t))$ en raisonnant sur la matrice $Y'^{(t)}$ plutôt que sur la matrice $Y^{(t)}$.

Construisons à présent cette séquence. Procédons en trois étapes. A la première, nous posons, pour tout $t \in \{1, \ldots, \delta\}$, $\nu(t) = \mu(t) = t$ ce qui revient à substituer successivement les vecteurs colonnes $b_{k(1)}, \ldots, b_{k(\delta)}$ aux vecteurs unités $e_{\pi(1)}, \ldots, e_{\pi(\delta)}$. On a donc, pour tout $t \in \{1, \ldots, \delta\}$, $k(t) = \kappa(t)$, $p(t) = \pi(t)$ et, vu la formule (11), $Y'^{(\delta)} = (e_1, \ldots, e_\delta, y'_{\delta+1}, \ldots, y'_m)$ où $y'_{\delta+1}, \ldots, y'_m$ est une permutation à déterminer des colonnes $b'_{+1}, \ldots, b'_m$ de B'.

Dans la deuxième, nous déterminons successivement les couples $(k(\delta+1),p(\delta+1)), \ldots, (k(m-\delta'),p(m-\delta'))$. Dans ce but, nous construisons deux permutations $(\nu(\delta+1), \ldots, \nu(m-\delta'))$ et $(\mu(\delta+1), \ldots, \mu(m-\delta'))$ de la suite $(\delta+1, \ldots, m-\delta')$ pour lesquelles nous espérons que les pivots successifs qu'elles définissent par les relations $k(t) = \kappa(\nu(t))$, $p(t) = \pi(\mu(t))$, pour $t \in \{\delta+1, \ldots, m-\delta'\}$ minimisent le nombre d'opérations de mise à jour à effectuer. Une façon d'y parvenir consiste à itérer le processus suivant. Supposant avoir déjà déterminé les ensembles $\{\nu(i) : \delta+1 \leqslant i < t\}$ et $\{\mu(i) : \delta+1 \leqslant i < t\}$ ainsi que la matrice $Y^{(t-1)}$ et avoir $t \leqslant m-\delta$ (dans le cas contraire, les permutations sont entièrement définies), sélectionnons pour nombre $\mu(t)$ un indice de l'ensemble $i\mu = \{\delta+1, \ldots, m-\delta'\} \setminus \{\mu(\ell) : \delta+1 \leqslant \ell < t\}$ qui correspond à une ligne ℓ_i possédant parmi les éléments de $L(i) = \{\ell_{ij} : j \{\delta+1, \ldots, m\} \setminus \{\nu(\ell) : \delta+1 \leqslant \ell < t\}$ un maximum d'éléments nuls. Sélectionnons ensuite, $\mu(t)$ étant défini, pour élément $\nu(t)$ un indice correspondant à un élément 1 parmi $L(\mu(t))$ s'il en existe, sinon à n'importe quel élément

non nul de $L(\mu(t))$. Remarquons ici, que l'inversibilité de la base assure à chaque pas, pour tout $\ell \in i\mu$, l'existence dans $L(\ell)$ d'un élément non nul.

Enfin, dans la troisième, nous posons, pour tout $t \in \{m-\delta'+1, \ldots, m\}$, $k(t) = \kappa(t)$ et $p(t) = \pi(t)$. Dans ces conditions, à chaque itération $t \in \{m-\delta'+1, \ldots, m\}$, aucune mise à jour des colonnes de $Y^{(t)}$ d'indice $m-t+1$ à m n'est à faire.

V.5. Problèmes de postoptimisation, d'analyse de sensibilité et de paramétrage

Que ce soit dans le but d'étudier la sensibilité d'une solution optimale ou encore afin de considérer certaines données comme paramètres variables, il est intéressant d'analyser l'influence d'une variation des données numériques d'un PLS sur ses solutions optimales. Ceci implique trois types distincts de problèmes:

1) les problèmes de postoptimisation étudient la conséquence sur les solutions minimales d'un PLS d'une modification "discrète" de ses données,

2) les problèmes d'analyse de sensibilité permettent l'exploration du voisinage d'une solution minimale et la détermination de la variation d'une donnée (ou de plusieurs données simultanément) pour laquelle la base obtenue reste minimum,

3) les problèmes de paramétrage consistent à faire varier certaines données de façon continue.

Nous n'envisagerons dans ce paragraphe que quelques problèmes de postoptimisation, renvoyant le lecteur désireux d'en traiter davantage aux livres ([35]) et ([32]).

Dans ce qui suit, nous supposons que le PLS de départ possède une base réalisable minimum. Nous la désignons par B et réécrivons, en utilisant des notations connues, ce PLS sous la forme

$$\min z = \pi b + (c_N - \pi N)x_N$$

$$\text{s.c. } x_B + B^{-1}Nx_N = B^{-1}b$$

$$x_B \geq \underline{0},\ x_N \geq \underline{0} \ .$$

On a bien sûr $B^{-1}b \geq 0$ et $c_N - \pi N \geq 0$.

V.5.1. Modification du second membre des contraintes

Soit $b + \Delta b$ la nouvelle valeur du second membre des contraintes. Deux cas peuvent se présenter.

Dans le premier, le vecteur $\bar{x}_B = B^{-1}(b + \Delta b)$ est non négatif. Le nouveau PLS est alors déjà résolu puisque la solution $(x_B = \bar{x}_B, x_N = 0)$ est réalisable et minimum (pour ce nouveau problème).

Dans le second, certaines des composantes de $\bar{x}_B$ sont strictement négatives. Il s'ensuit que la solution $(x_B = \bar{x}_B, x_N = 0)$ n'est plus réalisable. Elle vérifie néanmoins les hypothèses de l'algorithme V.3.1.1.. La méthode la plus rapide pour résoudre le PLS modifié consiste dès lors à appliquer l'algorithme dual du simplexe V.3.1.1. (ou V.3.2.1.) à partir du dernier tableau du simplexe en changeant $B^{-1}b$ en $B^{-1}(b + \Delta b)$. On peut également utiliser l'algorithme primal du simplexe. Dans cette optique, si un grand nombre de composantes de $\bar{x}_B$ sont négatives, il est préférable de reprendre le problème dans son ensemble. Sinon, il est plus adroit d'opérer comme suit. Posons L l'ensemble $\{\ell : \bar{x}_\ell < 0\}$. Associons à chaque ligne $\ell \in L$ du tableau du simplexe une variable artificielle x^a_ℓ de coût arbitrairement grand et attribuons lui pour colonne des coéfficients dans le tableau du simplexe le vecteur $-P_\ell$. Puis remplaçons dans l'ensemble des variables de base les variables x_ℓ, $\ell \in L$, par les variables x^a_ℓ, de valeur $-\bar{x}_\ell$. Le nouvel ensemble ainsi créé constitue une base réalisable dont la base correspondante B' est obtenue à partir de B par multiplication des lignes correspondant à ces variables x_ℓ de base par -1. Il ne reste plus pour déterminer une solution minimum du PLS modifié qu'à appliquer l'algorithme primal du simplexe à partir de B'.

V.5.2. Modification des coéfficients de la fonction économique

Soit $c + \Delta c$ le nouveau vecteur des coéfficients de la fonction économique. Deux cas sont à prendre en considération.

D'une part, si le vecteur $d = (c_N - c_B B^{-1} N) + (\Delta c_N - \Delta c_B B^{-1} N)$ est non négatif, la solution $(x_B = B^{-1}b, x_N = 0)$ est encore minimum et la nouvelle valeur minimum de la fonction économique vaut $(c_B + \Delta c_B)x_B = \pi b + \Delta c_B x_B$.

D'autre part, si on a $d_j < 0$ pour au moins un indice j, la solution $(x_B = B^{-1}b, x_N = 0)$ est à nouveau une solution de base réalisable mais n'est plus minimum. Il suffit donc pour résoudre le nouveau PLS d'appliquer l'algorithme primal du simplexe à partir du dernier tableau du simplexe en y remplaçant $c_N - \pi N$ par d.

V.5.3. Modification de la matrice des coéfficients des contraintes par addition d'une nouvelle variable

Soit n le nombre de variables du PLS de départ. L'introduction d'une nouvelle variable $x_{n+1} \geq 0$ revient à l'addition d'une colonne P_{n+1} à la matrice A des contraintes ainsi qu'une nouvelle composante c_{n+1} au vecteur c.

Les conséquences sont faciles à évaluer. De fait, la solution $(x_B = b, x_N = 0, x_{n+1} = 0)$ est réalisable. Elle ne reste toutefois minimum que si l'on a $c_{n+1} - \pi P_{n+1} \geq 0$. Dans le cas contraire, il reste pour déterminer une solution minimum du nouveau PLS à appliquer l'algorithme primal du simplexe au dernier tableau du simplexe complété par la $(n+1)^{eme}$ colonne $((c_{n+1} - \pi P_{n+1}), (B^{-1}P_{n+1})')'$.

V.5.4. Modification de la matrice des coéfficients des contraintes par suppression d'une variable

Comme la suppression d'une variable hors base n'affecte ni la valeur minimum de la fonction économique, ni l'optimalité de la solution $(x_B = B^{-1}b, x_N = 0)$, envisageons le cas où la variable à supprimer est en base.

Soient $x_{\nu(k)}$ cette variable et k l'indice de la ligne du tableau du simplexe qui lui correspond. Soit en outre $P_{\nu(i)}$ la i^{eme} colonne de la base B.

Procédons comme suit. Supprimons la $\nu(k)^{eme}$ colonne du tableau du simplexe et associons à sa k^{eme} ligne une variable x^* de coût arbitrairement grand. Le PLS ainsi obtenu admet la base $(P_{\nu(1)}, \ldots, P_{\nu(k-1)}, e_k, P_{\nu(k+1)}, \ldots, P_{\nu(m)})$ (dont l'inverse est donnée par application de la proposition V.4.2.) et la solution de base réalisable $(x_{\nu(1)} = \bar{b}_1, \ldots, x_{\nu(k-1)} = \bar{b}_{k-1}, x^* = \bar{b}_k, x_{\nu(k+1)} = \bar{b}_{k+1}, \ldots, x_{\nu(m)} = \bar{b}_m)$. Il est donc aisé d'en déterminer une solution minimum et par suite de résoudre le problème modifié. De fait, la non annulation de x^* dans cette solution traduit l'incompatibilité de ses contraintes. De plus, dans le cas contraire, les valeurs des variables de cette solution constituent une solution minimum du nouveau PLS.

V.5.5. Modification de la matrice des coéfficients des contraintes par addition d'une contrainte

Soit m le nombre de contraintes du PLS de départ. Ajoutons une $(m+1)^{eme}$ contrainte. Elle peut se présenter sous les trois formes différentes :

$$\sum_{i=1}^{n} a_{m+1,i}\, x_i \leqslant b_{m+1} \qquad (1),$$

$$\sum_{i=1}^{n} a_{m+1,i}\, x_i \geqslant b_{m+1} \qquad (2),$$

$$\sum_{i=1}^{n} a_{m+1,i}\, x_i = b_{m+1} \qquad (3),$$

avec $b_{m+1} \geqslant 0$.

Quant à la première, la solution $(x_B = B^{-1}b, x_N = 0)$ la vérifie déjà si on a, en désignant par a_B le vecteur formé des composantes de $(a_{m+1,1}, \ldots, a_{m+1,n})$ correspondant aux variables dans la base B, $b_{m+1} - a_B B^{-1} b \geqslant 0$. Dans ce cas, le problème modifié est déjà résolu. Toutefois, afin de faciliter les manipulations ultérieures, il est utile d'introduire cette contrainte dans le tableau du simplexe. Ceci peut se faire très facilement en réécrivant (1) à l'aide d'une variable auxiliaire x^* de coût nul :

$$\sum_{i=1}^{n} a_{m+1,i}\, x_i + x^* = b_{m+1}\,.$$

La nouvelle base vaut alors

$$B' = \begin{pmatrix} B & 0 \\ a_B' & 1 \end{pmatrix}$$

- d'inverse

$$\begin{pmatrix} B^{-1} & 0 \\ -a_B'B^{-1} & 1 \end{pmatrix} \quad -$$

et la solution de base associée est $(x_B = B^{-1}b, x^* = b_{m+1} - a_B x_B, x_N = 0)$. Dans le cas contraire, associons au PLS modifié le PLS

$$\min z = \Sigma_{i=1}^{n} c_i x_i + 0x^* + Mx^{**}$$

$$\text{s.c.} \quad \Sigma_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, \ldots m$$

$$\Sigma_{i=1}^{n} a_{m+1,i} x_i + x^* - x^{**} = b_{m+1}$$

$$x_i \geqslant 0, \quad i = 1, \ldots, n$$

$$x^* \geqslant 0, \; x^{**} \geqslant 0 \quad .$$

Il admet la base

$$B' = \begin{pmatrix} B & 0 \\ a_B' & -1 \end{pmatrix}$$

- d'inverse

$$B'^{-1} = \begin{pmatrix} B^{-1} & 0 \\ a_B'B^{-1} & -1 \end{pmatrix} -$$

et la solution de base réalisable $(x_B = B^{-1}b, x^{**} = a_B B^{-1}b - b_{m+1}, x_N = 0, x^* = 0)$. Il est donc aisé, par application de l'algorithme primal du simplexe, d'en déterminer une solution minimum et, de là, si elle existe, une solution minimum du PLS modifié. De fait, d'une part, la non annulation de x^* traduit l'incompatibilité de la contrainte ajoutée avec les contraintes existantes. D'autre part, si

on a $x^* = 0$, les valeurs obtenues des variables $x_1, \ldots, x_n$ déterminent une solution minimum du PLS modifié.

Les cas où la contrainte ajoutée s'écrit sous la forme (2) ou (3) se résolvent de même.

V.5.6. Modification de la matrice des coéfficients des contraintes par suppression d'une contrainte

La suppression d'une contrainte résulte des problèmes précédents. De fait, pour ce faire, il suffit d'affecter au second membre de cette contrainte une valeur arbitrairement grande et d'ajouter à la ligne correspondante du tableau du simplexe une variable d'écart de coût nul.

V.6. Programmation linéaire en nombres entiers

Comme nous l'avons déjà signalé, certaines composantes de bon nombre de problèmes de flot doivent être entières. Malheureusement, la théorie développée dans les paragraphes précédents ne tient nullement compte de ce caractère. Il convient donc de la compléter afin qu'elle puisse prendre en considération des contraintes d'intégrité.

V.6.1. Quelques définitions et notations

Pour y parvenir, introduisons tout d'abord la notion de problème de programmation linéaire en nombres entiers et donnons une typologie de ces problèmes.

DEFINITION V.6.1.1. Un problème de programmation linéaire en nombres entiers, noté PLE, est un problème de la forme

$$\min z = c'x + d'y$$

$$\text{s.c. } Ax + Gy = b$$

$$x \geqslant \underline{0},\ y \geqslant \underline{0}$$

$$x \text{ entier}$$

où les vecteurs $c = (c_1, \ldots, c_n)'$ et $d = (d_1, \ldots, d_n)'$ ainsi que les matrices A et G (resp. de dimension mxn et nxp) représentent les données et où les vecteurs $x = (x_1, \ldots, x_n)'$ et $y = (y_1, \ldots, y_p)'$ représentent les inconnues. La fonction $z = c'x + d'y$ est appelée la fonction économique du PLE.

NOTATION V.6.1.2. Dans la suite, les notations

z, c, d, A, G, x, y

se réfèrent, sauf mention explicite du contraire, aux notations de la définition V.6.1.1..

Il existe quatre types de problèmes de programmation en nombres entiers.

DEFINITION V.6.1.3. Un problème de programmation linéaire est

a) totalement en nombres entiers, ce que nous désignons par PLTE, si on a $d = \underline{0}$ et si la matrice G est nulle;

b) partiellement en nombres entiers, ce que nous désignons par PLPE, ou mixte, ce que nous désignons par PLM, si la matrice G est non nulle;

c) totalement en variables bivalentes, ce que nous désignons par PLTB, si on a $d = \underline{0}$, si la matrice G est nulle et si les composantes de x doivent valoir 0 ou 1;

d) mixte bivalent, ce que nous désignons par PLMB, si la matrice G est non nulle et si les composantes de x doivent valoir 0 ou 1.

Seuls les deux premiers sont utiles à la résolution des problèmes de flot. Nous nous y limiterons donc dans ce qui suit.

Définissons à présent ce que nous entendons par solution admissible et solution admissible minimum d'un PLTE et d'un PLM.

DEFINITION V.6.1.4.

a) Une solution admissible d'un PLTE (resp. d'un PLM) est une solution $x \in \mathbb{R}^n$ (resp. $(x,y) \in \mathbb{R}^n \times \mathbb{R}^p$) qui vérifie les contraintes $Ax \geqslant b$ (resp. $Ax + Gy \geqslant b$) ainsi que les contraintes de non négativité et d'intégrité du PLTE (resp. PLM).

b) Une solution admissible minimum d'un PLTE (resp. d'un PLM) est une solution admissible donnant à la fonction économique une valeur minimum.

Adoptons encore la convention suivante.

CONVENTION V.6.1.5. Soit (P) le problème de programmation linéaire mixte

$$\min z = c'x + d'y$$

$$\text{s.c. } Ax + Gy = b$$

$$x \geqslant \underline{0},\ y \geqslant \underline{0}$$

$$x \text{ entier .}$$

Nous notons respectivement

$$\mathcal{Q}(P) \quad \text{et} \quad \mathcal{Q}(P_R)$$

les ensembles de solutions admissibles du PLM (P) et réalisables de la relaxation (P_R)

$$\min z = c'x + d'y$$

$$\text{s.c. } Ax + Gy = b$$

$$x \geqslant \underline{0},\ y \geqslant \underline{0}$$

de (P). En outre, nous désignons par

$$z(P) \quad \text{et} \quad z(P_R)$$

les valeurs minimales des fonctions économiques de (P) et (P_R).

Les propriétés suivantes sont évidentes.

PROPOSITION V.6.1.6. On a

$$\mathcal{Q}(P_R) = \phi \Longrightarrow \mathcal{Q}(P) = \phi \qquad (R1)$$

et

$$z(P) \geqslant z(P_R) \qquad (R_2).$$

En particulier, toute solution minimum de (P_R) admissible pour (P) est une solution minimum de (P). (R_3)

Démonstration. C'est immédiat. ///

V.6.2. Une méthode de résolution

Parmi les différentes méthodes de résolution d'un PLTE ou d'un PLM, nous avons retenu les méthodes "branch and bound" qui semblent avoir receuilli les suffrages des praticiens de la programmation linéaire en nombres entiers. Rappelons-en le canevas algorithmique. Il repose sur deux notions fondamentales : la séparation et le rejet.

(i) la séparation

Elle est basée sur les définitions suivantes.

DEFINITION V.6.2.1.

a) Le problème (P) est séparé en les sous-problèmes $(P_1), \ldots, (P_q)$ si les deux conditions suivantes sont satisfaites :

- (S_1) toute solution admissible de (P) est une solution admissible d'un et d'un seul des sous-problèmes $(P_1), \ldots, (P_q)$,
- (S_2) toute solution admissible de tout sous-problème (P_i) est une solution admissible de (P)

c'est-à-dire si l'ensemble $\{\mathcal{Q}(P_1), \ldots, \mathcal{Q}(P_q)\}$ constitue une partition de l'ensemble $\mathcal{Q}(P)$.

b) Un candidat à la séparation est un problème susceptible de subir une séparation.

c) Un intermédiaire de la séparation $\{(P_1), \ldots, (P_q)\}$ de (P) est un sous-problème $(P^*) \in \{(P_1), \ldots, (P_q)\}$ tel que

$$z^* = z(P^*) = \min_{i = 1, \ldots, q} z(P_i)$$

et pour lequel on connaît une solution admissible minimum.

Il est clair qu'un intermédiaire n'est pas considéré comme un candidat puisqu'on en connaît une solution minimum.

Ce qui précède suggère la méthode de résolution que nous avons adoptée. Elle consiste à effectuer des partitions de plus en plus fines du problème (P) à résoudre de façon à obtenir des problèmes dont la résolution est "plus aisée" et ce, jusqu'à ce qu'on puisse établir qu'une solution minimum d'un des sous-problèmes (P_i) coïncide avec une solution minimum de (P).

Précisons comment s'exécutent les séparations. Soit (CP) un candidat à séparer. Une façon de procéder, généralement utilisée, consiste à repérer une variable x_j soumise à une contrainte d'intégrité ainsi qu'à une contrainte du type $\alpha_j \leqslant x_j \leqslant \beta_j$ (avec éventuellement $\beta_j = +\infty$ ou $\alpha_j = 0$) et à créer, à partir de là, deux nouveaux sous-problèmes identiques à (CP) à cette différence près que l'on ajoute la contrainte $\alpha_j \leqslant x_j \leqslant \gamma_j$ pour l'un et la contrainte $\gamma_j + 1 \leqslant x_j \leqslant \beta_j$ pour l'autre, où γ_j est un entier situé dans $]\alpha_j, \beta_j[$. Pour conclure, il reste, bien entendu, à définir le mode de sélection de la variable de branchement x_j. Plusieurs modes ont été proposés (cfr. [1] , [4], [5] , [6], [7], [19]) parmi lesquels nous avons retenu :

1) choisir pour variable de branchement une variable x_j de l'ensemble

$$J = \{x_k : x_k = \gamma_k + f_k,\ \gamma_k \in N,\ 0 < f_k < 1\}$$

telle que

$$\min \{f_j, 1 - f_j\} = \max_{k \in J} \min \{f_k, 1 - f_k\};$$

2) choisir pour variable de branchement une variable x_j de J telle que

$$\min \{f_j, 1 - f_j\} = \min_{k \in J} \min \{f_k, 1 - f_k\};$$

3) choisir pour variable de branchement une variable x_j de J telle que

$$|c_j| = \min_{k : x_k \in J} |c_k| ;$$

4) choisir pour variable de branchement une variable x_j de J conformément à un ordre de priorité fourni par l'utilisateur.

(ii) le rejet

Il est clair, si l'on prend le temps de calcul en considération, que l'on a intérêt à éviter, quand c'est possible, la séparation de sous-problèmes.

Supposons qu'à un moment donné, dans le déroulement des séparations successives, z^* soit la valeur de la fonction économique de l'intermédiaire correspondant à la séparation obtenue à ce moment. Supposons en outre que le problème (CP) soit candidat à la séparation et que la relaxation (CP_R) de (CP) ait déjà été résolue. Trois cas sont susceptibles de se présenter.

Dans le premier, on a $\mathcal{Q}(CP_R) = \phi$. Dans ce cas, vu (R_1), on a $\mathcal{Q}(CP) = \phi$ et (CP) ne contient pas de solution admissible de (P). Il est donc inutile de séparer (CP) et on peut le rejeter.

Dans le deuxième, on a $z(CP) \geqslant z^*$. Dans ces conditions, il est impossible que $\mathcal{Q}(CP)$ contienne une solution admissible de (P) qui soit meilleure que la solution minimum de l'intermédiaire. On peut donc rejeter (CP) de la liste des candidats du moins si l'on ne désire qu'une solution minimum de (P). Malheureusement, la connaissance de $z(CP)$ suppose la résolution de (CP), ce que nous ne pouvons admettre (cela signifierait que nous disposions d'un algorithme autre que l'algorithme "branch and bound"; pourquoi alors ne pas résoudre (P) au moyen de cet algorithme?). Cependant, $z(CP_R)$ est connu et, par (R_2), on peut affirmer qu'on a $z(CP) \geqslant z(CP_R)$. Il s'ensuit que si l'on a $z(CP_R) \geqslant z^*$, le candidat (CP) peut être rejeté.

Enfin, dans le troisième, il existe une solution réalisable minimum de (CP_R) qui est aussi admissible pour (CP). Cette solution est d'après (R_3) une solution minimum de (CP) et par suite, vu (S_2), une solution admissible de (P). Il vient donc $z(CP) = z(CP_R) \geqslant z(P)$. Dès lors, si on a $z(CP) = z(CP_R) < z^*$, le sous-problème (CP) devient l'intermédiaire et peut être éliminé de la liste des candidats. De plus, si on a $z(CP) = z(CP_R) \geqslant z^*$, on peut rejeter (CP) de la liste des candidats.

En conclusion, le candidat (CP) peut être rejeté de la liste des candidats si l'un quelconque des trois critères de rejets suivants est satisfait :

(CR1) une analyse de (CP_R) montre qu'on a $Q(CP_R) = \phi$,

(CR2) une analyse de (CP_R) montre qu'on a $z(CP_R) \geqslant z^*$,

(CR3) une analyse de (CP_R) montre qu'une solution minimum de (CP_R) est aussi une solution admissible de (CP).

Nous disposons à présent des outils nécessaires à l'application de la méthode "branch and bound". Enonçons l'algorithme qui en découle.

ALGORITHME V.6.2.2.

(1). liste_des_candidats := {(P)}
$z^* := +\infty$

(2). Si liste_des_candidats = ϕ alors
si $z^* < +\infty$, arrêter les calculs : la solution minimum de l'intermédiaire est aussi solution minimum de (P)
sinon, arrêter les calculs : (P) ne possède pas de solution admissible
sinon aller en (3).

(3). Choisir un problème dans la liste liste_des_candidats et le retenir comme candidat à la séparation.
Aller en (4).

(4). Résoudre la relaxation (CP_R) de (CP).
Aller en (5).

(5). Si on a $Q(CP_R) = \phi$ alors
liste_des_candidats := liste_des_candidats $\setminus$ {(CP)}
aller en (2)
sinon aller en (6).

(6). Si on a $z(CP_R) \geqslant z^*$ alors
liste_des_candidats := liste_des_candidats $\setminus$ {(CP)}
aller en (2)
sinon aller en (7).

(7). Si une solution minimum de (CP_R) est admissible pour (CP) alors
liste_des_candidats := liste_des_candidats $\setminus$ {(CP)}
intermédiaire_actuel := (CP)
aller en (2)
sinon aller en (8).

(8). Séparer (CP) et le remplacer dans la liste liste_des_candidats par les sous-problèmes qui en résultent. Aller en (2).

L'étape *(3)* ne précise rien quant au choix du candidat dans la liste *liste_des_candidats*. Pour ce faire, nous suggérons deux méthodes. La première, "dernier arrivé-premier servi" (LIFO) sélectionne toujours le candidat qui a été ajouté en dernier lieu à la liste des candidats. Elle permet d'assurer facilement de la liste *liste_des_candidats* et de faciliter considérablement la réoptimisation de l'étape *(4)*. Par contre, elle ne permet pas de revenir directement à un candidat dont la séparation semblerait a priori plus prometteuse. La seconde offre un choix du candidat selon un ordre de priorité établi ou non à l'avance. En voici plusieurs réalisations:

1) l'utilisateur choisit lui-même le candidat,
2) ayant sélectionné un candidat dans la liste des candidats, le système rappelle également son compagnon (c'est-à-dire le candidat qui résulte de la même séparation) et ne retient pour exploitation que celui dans lequel la variable de branchement a été soumise à une contrainte de prédilection de l'utilisateur, par exemple, $x_k \leqslant v_k$,
3) ayant sélectionné un candidat dans la liste des candidats, le système l'exploite immédiatement puis rappelle et exploite son compagnon.

Toutefois, comme le montrent les résultats obtenus dans [4], [6], [7] et [20], il n'existe aucune stratégie universelle aussi bien en ce qui concerne le choix du candidat à séparer qu'en ce qui concerne le choix de la variable de branchement.

BIBLIOGRAPHIE

[1] R.D. ARMSTRONG, P. SINHA, Improved penalty calculations for mixed-integer branch and bound algorithms, Mathematical programming, 6 (1974), 212-223.

[2] C. BAUDOIN, B. MEYER, Méthodes de programmation, Eyrolles, Paris (1980).

[3] G. BAYER, Algorithm 423 Maxflow, Communications of the ACM, vol. 11, n° 2 (1968), 117-118.

[4] E.M.L. BEALE, Mathematical programming in practice, Pitman, Londres (1968).

[5] E.M.L. BEALE, The current algorithmic scope of mathematical programming systems, Mathematical programming study, 4 (1975), 1-11.

[6] M. BENICHOU, J.M. GAUTHIER, P. GIRODET, G. HENTGES, G.RIBIERE, O. VINCENT, Experiments in mixed-integer linear programming, Mathematical programming, 3 (1971), 76-94.

[7] M. BENICHOU, J.M. GAUTHIER, G. HENTGES, G. RIBIERE, The efficient solution of large-scale linear programming problems. Some algorithmic techniques and computational results, Mathematical programming, 13 (1977), 280-322.

[8] C. BERGE, Graphes et Hypergraphes, Dunod, Paris (1970).

[9] T.Y. CHEUNG, Computational comparison of eight methods for the maximum network flow problem, ACM Transaction of Mathematical Software, vol. 6, n° 1 (1980), 1- 16.

[10] G.B. DANTZIG, Linear programming and extensions, Princeton University Press, Princeton, N. J. (1963).

[11] E.A. DINIC, Algorithm for solution of a problem of Maximum Flow in a Network with power estimation, Soviet. Math. Dokl., vol. 11, n° 5 (1970), 1277-1280.

[12] J. EDMONDS, Paths, trees and flowers, Canad. J. Math. 17 (1965), 449-467.

[13] J. EDMONDS, R.M. KARP, Theoretical Improvements in Algorithmic efficiency for Network Flow Problems, J. ACM, vol. 19, n° 2 (1972), 248-264.

[14] J.R. EVANS, Maximum Flow in Probabilistic Graphs. The discrete case, Networks 6 (1976), 161-183.

[15] S. EVEN, R. TARJAN, Network Flow and Testing Graph Connectivity, SIAM Comput., vol. 4, n° 4 (1975), 507-518.

[16] J. FICHEFET, Cours de programmation linéaire, Cours de 2 ^{e} licence, Ed. Ronéotypée, Namur.

[17] C.O. FONG, M.R. RAO, Accelerated Labeling Algorithms for the Maximal Flow Problem with Applications to Transportation and Assignment Problems, Working paper, n° 7222, Graduate School of Management, University of Rochester, Rochester, N. Y. (1972).

[18] L.R. FORD, D.R. FULKERSON, Flows in Networks, Princeton University Press, Princeton (1962).

[19] S.I. GASS, An Illustrated Guide to Linear Programming, Mcgraw-Hill, New-York (1970).

[20] J.M. GAUTHIER, G. RIBIERE, Experiments in mixed-integer linear programming using pseudo-costs, Mathematical programming 12 (1977), 26-47.

[21] G. HADLEY, Linear Programming, Addison-Wesley, Reading, Mass. (1972).

[22] F. HARARY, Graph Theory, Addison-Wesley, Reading, Mass. (1969).

[23] T.C. HU, Integer Programming, Addison-Wesley, Reading, Mass. (1969).

[24] M. HORPS, B. ROY, Algèbre moderne et théorie des graphes, tome 2, fascicule 4, Dunod, Paris (1970).

[25] A. ITAI, Y. SHILOACH, Maximum Flow in Planar Networks, S.I.A.M. Comput., vol. 8, n° 2 (1979), 135-150.

[26] J.M. JACQUET, Analyse des performances d'algorithmes et application à la comparaison d'algorithmes de flot dans les réseaux de transport, Mémoire de licence, Ed. Ronéo (1984).

[27] E.L. JOHNSON, Networks and basic solutions, Oper. Res. 14 (1966), 619-623.

[28] A.V. KARZANOV, Determining the maximal flow in a network by the method of preflows, Soviet. Math. Dokl., vol. 15, n° 2 (1974), 434-437.

[29] B. KINARIWALA, A.G. RAO, Flow Switching Approach to the Maximum Flow Problem : I, Journal of the ACM, vol. 24, n° 4 (1977), 630-645.

[30] H.P. KUNZI, H.G. TZSCHACH, C.A. ZEHNDER, Numerical methods of mathematical optimization, Academic Press, New-York (1971).

[31] V.M. MALHOTRA, M.P. KUMAR, S.M. MAHESHWARI, An $\mathcal{O}(|V^3|)$ algorithm for finding Maximum Flows in Networks, Information Processing Letters, vol. 7, n° 6 (1978), 277-278.

[32] W. ORCHARD-HAYS, Advanced Linear Programming Computing Techniques, MacGraw-Hill, New-York (1968).

[33] O. ORE, The four-color problem, Academic Press, New-York (1967).

[34] A.G. RAO, An alternate approach to the maximum flow problem, Thèse de doctorat, Université d'Hawai, (1975).

[35] M. SIMMONARD, Programmation linéaire, Tomes 1 et 2, Dunod, Paris (1972).

[36] V. SRINIVASAN, G.L. THOMPSON, Accelerated Algorithms for Labeling and Relabeling of Trees with applications to Distribution Problems, Journal of the ACM, vol. 19, n° 4 (1972), 712-726.

[37] W.L. SULKINS, Certification of Algorithm 266, Communications of the ACM, vol. 15, n° 12 (1972), 1073.

[38] N. ZADEH, More pathological examples for Network Flow Problems, Mathematical programming 5 (1973), 217-224.

[39] N. ZADEH, Theoritical Efficiency of the Edmonds-Karp Algorithm for Computing Maximal Flows, Journal of the ACM, vol. 19, n° 1 (1972), 184-192.

TABLE DES MATIERES

CHAPITRE III : LA METHODE DE REDISTRIBUTION DE FLUX

CHAPITRE IV : LA METHODE DES PREFLOTS

CHAPITRE V : LA PROGRAMMATION LINEAIRE