MASTER EN SCIENCES INFORMATIQUES

Contribution à la conception des systèmes informatiques répartis

Detalle, Jean-Claude; Pichot, Michel

Award date:
1984

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique

Année académique 1983-1984

CONTRIBUTION A LA CONCEPTION DES SYSTEMES INFORMATIQUES REPARTIS

DETALLE Jean-Claude

PICHOT Michel

Mémoire présenté en vue de l'obtention du grade
de Licencié et Maître en Informatique.
Promoteur de mémoire : Monsieur le Professeur VAN BASTELAER.

Nous tenons à remercier vivement Monsieur le Professeur VAN BASTELAER de nous avoir guidé dans la réalisation de ce travail.

INTRODUCTION

Nous nous sommes fixés pour but essentiel d'étudier et d'implémenter le travail de S. Mahmoud et J.S. Riordon "Optimal allocation of resources in distributed information networks" (cfr [MAHMOUD 76]). Ces auteurs proposent un algorithme heuristique [1] de solution au problème de l'allocation de fichiers aux noeuds et de capacités aux lignes de communication d'un réseau informatique donné, dans le but de minimiser le coût global du système [2], tout en répondant aux besoins des utilisateurs et en satisfaisant diverses contraintes [3].

[MAHMOUD 76] n'est en fait qu'un travail parmi bien d'autres. De très nombreux auteurs (cfr chapitre I) ont étudié le problème de l'allocation de fichiers (cfr I.1 à I.4) ou encore celui de la conception du réseau de communication [4] (cfr I.5), indépendamment de tout autre problème de design.

---

(1) I.e. un algorithme itératif ne convergeant pas nécessairement vers la solution cherchée.

(2) Coût de stockage des fichiers et de location des lignes.

(3) Sur le délai moyen de propagation d'un message dans le réseau, ainsi que sur la disponibilité des fichiers (probabilité qu'une copie du fichier concerné soit accessible lorsque réclamée par un utilisateur).

(4) Conception de la topologie du réseau, allocation de capacités aux lignes de communication, ... etc.

Il nous a semblé intéressant de présenter un aperçu de leurs approches (1), afin de suggérer diverses alternatives ou améliorations aux choix (i) pris par [Mahmoud 76]. Les travaux traitant simultanément de ces deux problèmes sont nettement moins abondants (cfr I.6). Tel est cependant (partiellement) (2) le cas de [MAHMOUD 76], dont nous proposons une première introduction (3) en I.6.

Tout en n'étant qu'un travail parmi d'autres, [MAHMOUD 76] nous a cependant paru digne d'un intérêt plus particulier pour trois raisons essentielles :

(i) il traite simultanément des problèmes de l'allocation des fichiers et des capacités;

(ii) il propose une technique heuristique de solution, seule approche applicable (cfr chapitre I) à des problèmes de taille réaliste;

(iii) on peut facilement pallier à l'hypothèse de topologie fixe (cfr renvoi (2) ci-dessous) en construisant un programme qui considérerait différentes topologies (cfr I.5 et I.6), et qui, pour chacune d'elles, appellerait cette heuristique.

---

(1) Jeux d'hypothèses, modèles mathématiques sous-jacents, techniques de solution et tests de performance.

(2) [MAHMOUD 76] suppose que la topologie du réseau (emplacements des noeuds et des lignes) est connue et fixée; il se "contente" d'allouer les capacités aux lignes; cette lacune a motivé le développement du paragraphe I.5.

(3) Hypothèses des auteurs et critiques (énoncées dans la littérature), philosophie générale de leur heuristique, tests de performance proposés.

Le chapitre II étudie un programme basé sur l'algorithme donné par Mahmoud [MAHMOUD 76]. Les données, les contraintes et l'énoncé du problème sont tout d'abord explicités au paragraphe II.1. Ensuite, la mise en oeuvre du programme est détaillée au paragraphe II.2.

Les spécifications des procédures sont données en appendice tandis que le code du programme et les tests, sont donnés en annexe sur listings.

TABLE DES MATIERES

APPENDICES

CHAPITRE I

SYNTHESE DES EFFORTS DE RECHERCHE RELATIFS AU PROBLEME DE LA DISTRIBUTION D'INFORMATIONS SUR UN SYSTEME INFORMATIQUE REPARTI

Introduction générale

Le but de ce chapitre est de présenter une synthèse (partielle) des efforts de recherche relatifs au problème de la distribution d'informations (1) sur un système informatique réparti (2).

Ces efforts de recherche consistent essentiellement en l'application de techniques de la programmation mathématique classique ou de techniques heuristiques, à un problème (3) de design des systèmes informatiques communément appelé **"problème d'allocation de fichiers"** ("file allocation problem" ou encore **FAP**), que l'on pourrait énoncer de la façon suivante:

---

(1) Fichiers, fragments de BDs résultant de partitions horizontales et/ou verticales de celles-ci, ... etc.

(2) Nous étudierons également, avec un certain détail, le problème de la conception du réseau de communication (design de la topologie du réseau, allocation de capacités aux lignes de communication,...etc).

(3) Ou plutôt au modèle mathématique sous-jacent à ce problème.

considérant un réseau informatique et des fichiers et

étant donné une description de la demande de service de chaque utilisateur (noeud du réseau) à l'égard de chaque fichier [1] ainsi que des ressources disponibles pour satisfaire cette demande [2].

déterminer une allocation optimale (selon certains critères) des fichiers (avec d'éventuelles copies multiples d'un même fichier) aux noeuds du réseau qui respecte certaines contraintes [3] (par exemple de capacité de stockage aux noeuds, de disponibilité des fichiers, de temps de réponse, de parallélisme [4], de sécurité [5],... etc) et satisfait tous les accès.

---

(1) Par exemple sous la forme de taux de "queries" et d'"updates" émanant de chaque noeud du réseau pour chaque fichier.

(2) Par exemple la topologie du réseau, les capacités et les coûts des lignes et des noeuds ... etc.

(3) Ces contraintes, comme celles de délai et de disponibilité, empêchent souvent de décomposer le problème d'allocation de fichiers multiples en problèmes d'allocation de fichier unique. Nous reviendrons, au cours de ce chapitre, sur ces différentes contraintes, sur leur nature, leur pertinence ainsi que sur la façon avec laquelle différents modèles les ont incorporées.

(4) Une contrainte peut imposer, par exemple, que des copies de deux fichiers se trouvent sur des ordinateurs différents.

(5) Empêcher, par exemple, le stockage de telles copies en tels noeuds.

Le critère d'optimalité le plus souvent retenu est celui de la minimisation du coût (coût de stockage des fichiers, coût de communication ... etc) (1). Signalons d'emblée que le problème d'optimisation est habituellement un problème de programmation entière non linéaire, que les techniques optimales de solutions ne sont applicables généralement qu'à de petits exemples et que la "qualité" des solutions produites par les heuristiques (algorithmes itératifs ne convergeant pas nécessairement vers la solution cherchée, cfr [MULLER-MERBACH 81] et [BALL 81]) est très difficile à évaluer. Des mesures de performance du système et des seuils sur la capacité des "devices" peuvent être incluses (dans le modèle mathématique sous-jacent) comme contraintes mais habituellement, les délais de files d'attente ne sont pas pris en compte. Remarquons que les contraintes accroissent la complexité du problème.

Notre but est d'expliciter quelque peu la nature du problème de l'allocation de fichiers, d'en souligner la grande complexité et de présenter les avantages et les faiblesses des différents modèles et techniques de solution (relatifs à ce problème) que l'on peut rencontrer dans la littérature. Les modèles se distinguent par le (ou les) critère(s) d'optimalité (minimiser le coût de fonctionnement du système, le temps de réponse, maximiser le throughput ... etc) et le jeu d'hypothèses retenus (cfr [DOWDY 82]). Les techniques de solution (programmation mathématique, recherches exhaustives, heuristiques ... etc) sont critiquées sur la base des exemples numériques (souvent bien "maigres") fournis par les divers auteurs.

---

(1) La fonction de coût peut être plus ou moins complexe; le coût d'un "query", par exemple, peut être séparé en coût de la requête et coût du flux de données en retour.

Ce chapitre nous permet, également, de replacer dans son contexte le travail de S. Mahmoud et J.S. Riordon (cfr [MAHMOUD 76]), "optimal allocation of resources in distributed information networks" (cfr introduction générale). Nous nous contentons de présenter ici la nature du problème étudié par ces auteurs, les hypothèses sous-jacentes au modèle proposé, les techniques de solution suggérées et les tests de performance proposés, ainsi que diverses critiques énoncées, dans la littérature, au sujet de cette étude.

Le FAP est en relation avec bien d'autres problèmes de design tels, par exemple, celui de la décomposition et du traitement des "queries", des recherches en parallèle, du partitionnement et de la compression de l'information, de la synchronisation des "updates", ... etc; nous y ferons allusion (en I.6.) brièvement, en indiquant certaines pistes à suivre . Nous avons trouvé, par contre, intéressant de développer quelque peu le problème de la conception du réseau de communication (design de la topologie du réseau, allocation de capacités aux lignes, ... etc); nous l'étudions isolément du FAP (en I.5.), puis nous proposons divers efforts d'intégration de ces deux problèmes (en I.6.). Cela permet, entre autre chose, de pallier à une lacune de [MAHMOUD 76], qui étudie le FAP et le problème de l'allocation des capacités (simultanément), mais suppose que la topologie du réseau est fixe (cfr I.6.).

[WAH 84] signale qu'il est très difficile, dans le cas de BDs réelles, de résoudre ces problèmes comme des unités à part entière, aussi la pluspart des concepteurs les ont-ils décomposés en problèmes indépendants. Il fait également remarquer une certaine tendance, parmi les travaux récents, à l'intégration de ces problèmes avec celui du placement de fichiers.

La **première étude** relative au problème de la distribution de fichiers parmi les noeuds d'un réseau informatique, est sans doute celle de [CHU 69]. Nous présentons cette étude en introduction, afin de familiariser d'emblée le lecteur à la nature du problème que nous nous proposons d'étudier tout au long de ce chapitre.

Chu étudie le problème suivant : étant donné un certain nombre d'ordinateurs travaillant sur un ensemble commun de fichiers, les taux de requêtes de chaque ordinateur à l'égard de chaque fichier et les taux de mise à jour relatifs à chaque fichier (on suppose que chaque paire d'ordinateurs est capable de transmettre de l'information en mode bidirectionnel), comment peut-on allouer les fichiers (aux ordinateurs) de façon à minimiser les coûts de fonctionnement (coûts de stockage et de transmission) tout en respectant les contraintes suivantes :

(i) le nombre de copies de chaque fichier est supposé fixe;

(ii) le temps moyen d'accès à chaque fichier depuis chaque ordinateur est inférieur à une valeur donnée; (1)

(iii) la taille mémoire disponible à chaque ordinateur est limitée (2).

Le modèle de Chu a la forme d'un programme entier 0-1 non linéaire; l'auteur suggère de linéariser ce programme (il décrit une technique pour ce faire) et d'appliquer une méthode directe de programmation linéaire (par exemple la technique dite de "cutting" de GOMORY, cfr [GOMORY 63]).

---

(1) Le délai d'accès est un facteur important dans le FAP. En général, les délais individuels ne sont pas analysés : un délai moyen pour toutes les transactions est plutôt calculé.

(2) Les autres chercheurs n'ont généralement pas inclus cette contrainte dans leurs modèles, parce que les coûts de stockage sont très faibles et que des capacités de stockage quasi illimitées sont habituellement disponibles en chaque noeud (si ce n'est peut-être pour les systèmes basés sur des micro-ordinateurs).

La linéarisation implique un accroissement énorme du nombre de variables et de contraintes et conduit, de ce fait, à un programme extrêmement difficile du point de vue des calculs (cfr [CERI 82^{b}]), bien que [PRICE 78] montre qu'on peut résoudre des problèmes de programmation linéaire 0-1 de taille honnête.

Chu propose un modèle plus complet encore dans [CHU 73] . Il y ajoute des contraintes sur la disponibilité [(1)] des fichiers. Les noeuds et les lignes tombant en panne périodiquement, ces contraintes forcent la distribution d'un nombre suffisant de copies de fichiers.

Remarquons que les hypothèses sous-jacentes aux définitions du temps de réponse et de la disponibilité d'un fichier proposées dans [CHU 73] ne sont applicables qu'à un réseau "fully connected" (cfr [DOWDY 82]).

A la suite de cette première étude, un certain nombre d'auteurs se sont intéressés au problème (plus simple) de l'allocation d'un fichier unique ainsi qu'à la possibilité de décomposer le problème d'allocation de fichiers multiples en de tels sous-problèmes.

Chaque placement de fichier est considéré indépendamment du placement des autres fichiers, les délais de file d'attente devant être, dès lors,négligés. Remarquons d'emblée que cette technique ne peut conduire à une allocation globale (i.e. de tous les fichiers) optimale.

Il se peut, par exemple, que l'allocation de deux fichiers à un même noeud provoque une contention (file d'attente) intolérable en ce noeud. Il peut donc être préférable d'allouer ces fichiers en des noeuds distincts.

---

(1) La disponibilité d'un fichier peut être définie comme étant la probabilité qu'au moins une de ses copies soit accessible lorsqu'un accès à ce fichier est demandé; elle est liée au nombre de copies du fichier, à la fiabilité et à la connectivité du réseau ... etc.

Cette technique peut s'avérer utile lors de l'étude de caractéristiques telles l'interaction entre programmes et données, la recherche heuristique, la complexité d'allocation, les copies de fichiers etc; elle fournit de bonnes approximations. L'introduction de contraintes (par exemple de délai ou de disponibilité) empêche souvent la décomposition du problème d'allocation de fichiers multiples; dès lors, la technique de solution la plus fiable, dans ce cas, est encore l'énumération exhaustive (par exemple les algorithmes "branch-and-bound", cfr [CHEN 80]).

Nous présentons tout d'abord (en I.1) avec un certain détail la formulation et les techniques de solution du problème de l'allocation d'un fichier unique, proposées par CASEY (cfr [CASEY 72]). Ce modèle, simple dans sa formulation, a été repris et étudié par de très nombreux auteurs. ESWARAN a démontré que le problème de Casey - qu'il appelle "File Allocation problem" (FAP) et que nous rebaptiserons (avec [RAMAMOORTHY 83]) "Simple File Allocation problem" (SFAP) - est NP-complet. Une telle affirmation a conduit de nombreux auteurs (cfr I.2.) à développer des critères permettant de simplifier le SFAP, à considérer des réseaux particuliers ou encore à développer des techniques de solution alternatives à celles de CASEY.

Cette première approche du problème général de l'allocation de fichiers, par le biais du SFAP (cas particulier on ne saurait plus simple !), nous aura permis de préciser la nature du problème (et les divers compromis qu'il soulève), sa complexité, la façon dont on peut le modéliser et le résoudre.

Nous présentons ensuite (en I.3.) un aperçu détaillé des travaux de K.D. LEVIN et H.L. MORGAN. L'importance et l'influence de ces deux auteurs semblent incontestables. Ils suggèrent un cadre de recherche (qui est caractérisé par trois dimensions, donc $2^3 = 8$ jeux d'hypothèses distincts) relatif au problème de la minimisation du "coût de fonctionnement" d'une base de données distribuée partagée par une communauté d'utilisateurs interconnectés au travers d'un réseau informatique; ils proposent des modèles, des techniques de solution et des exemples numériques relatifs à chacun des jeux d'hypothèses.

Ce cadre de recherche englobe le SFAP et de nombreuses variations plus réalistes de celui-ci.

Nous présentons (en I.4.) divers travaux qui s'insèrent dans le cadre de recherche développé par Levin et Morgan. Nous insistons plus particulièrement sur les diverses techniques de solution utilisées et les tests de performance proposés par les auteurs (de ces travaux).

Nous proposons (en I.5.) quelques résultats relatifs à l'analyse de fiabilité et de délai dans les réseaux, ainsi qu'au problème de la conception des réseaux centralisés et distribués (conception de la topologie du réseau, allocation des capacités aux lignes,...etc.)

Nous présentons enfin (en I.6.) divers travaux (dont parmi eux [MAHMOUD 76]) relatifs à l'étude simultanée du FAP et du problème de la conception du réseau de communication (étudié en I.5). Nous signalons également l'existence d'autres problèmes de design liés au FAP, et proposons divers essais d'intégration de ces différents problèmes.

I.1. LE PROBLEME DE CASEY OU "SIMPLE FILE ALLOCATION PROBLEM" (SFAP)

(Référence principale : [CASEY 72])

1) Introduction

On considère un **réseau** informatique constitué de n noeuds (indicés de 1 à n) dont chacun a la possibilité de communiquer avec tout autre noeud via des lignes de communication (et d'éventuels noeuds intermédiaires). On distingue [1] deux types de **transactions** relatives à un **fichier** F (dont plusieurs copies peuvent être stockées sur le réseau) :

(i) les **queries** ("query traffic") : un "query" a pour origine un noeud du réseau et est communiqué à une seule copie du fichier, celle qui minimise le coût de communication (lecture d'information depuis une copie unique);

(ii) les **"updates"** ("update traffic") : on suppose qu'un message d'"update" est envoyé à chaque copie du fichier (écriture d'information sur toutes les copies).

Appelons **allocation du fichier F** (ou encore allocation de F) un sous-ensemble I de $\{1, 2, \ldots, n\}$ dont les éléments correspondent à des indices de noeuds où est stockée une copie de F (et notons I_k, une allocation de F à k éléments ($\# I_k = n$), $k = 0, 1, \ldots, n$).

---

(1) Casey est certainement le premier à mettre en évidence la différence entre "updates" et "queries".

Considérons une certaine période (de planification) pendant laquelle l'utilisation globale du fichier F est parfaitement connue (et exprimée sous la forme de taux de "query" et d'"update" émanant de chaque noeud du réseau).

Le problème consiste à trouver une allocation de F qui minimise la somme des coûts de stockage et de communication des "queries" et des "updates".

Remarquons d'emblée (cfr [CASEY 72]) que le coût de communication des "queries" diminue lorsqu'augmente le nombre de copies de F dans le réseau (c'est plus économique pour les utilisateurs proches d'une nouvelle copie et cela ne saurait être pire pour les autres !) et que, d'un autre côté, les coûts de stockage et de communication des "updates" croissent lorsque de nouvelles copies de F sont introduites (chaque copie doit être mise à jour !). On pourrait envisager les **cas limites** suivants :

(i) une copie à chaque noeud s'il n'y a pas de mise à jour et si les coûts de stockage sont faibles;

(ii) une seule copie en un noeud optimal s'il n'y a que des "updates".

Nous décrivons ci-après le modèle de Casey et diverses formulations de sa fonction objective (en 2); nous discutons ensuite de la complexité du SFAP et nous explicitons les techniques de solution proposées par Casey (en 3); nous procédons, enfin, à une étude critique du modèle et des techniques de solution (en 4).

2) Le modèle

2.1. Les données

σ_i = coût fixe de location d'une copie de F au noeud i ($ /mois);

d_{ij} (resp. d'_{ij}) = coût d'une unité de communication du noeud i au noeud j pour un "query" (resp. "update") ($/Mbits);

λ_i (resp. ψ_i) = volume de "query traffic" (resp. "update traffic") émanant du noeud i (Mbits/mois).

Remarque : il est possible que $d_{ij} \neq d'_{ij}$, les "updates" étant souvent accumulées et envoyées par un moyen et/ou à un moment favorables.

2.2. Fonction de coût

L'expression générale du coût total est la suivante :

$$C(I) = \sum_{j=1}^{n} \left(\sum_{i \in I} \psi_j \, d'_{jk} + \lambda_j \min_{k \in I} d_{jk} \right) + \sum_{k \in I} \sigma_k ,$$

où I est une allocation quelconque de F. Le problème d'allocation du fichier, appelé **"Simple File Allocation Problem"** (cfr introduction) consiste à choisir l'allocation de F (notée I_{opt}) qui minimise C (I). Remarquons que la fonction de coût peut être réécrite sous la forme suivante (qui est celle du "Plant location problem", cfr [EFROYMSON 66]):

$$C(I) = \sum_{k \in I} U_k + \sum_{j=1}^{n} G_j(I)$$

où $U_k = \sigma_k + \sum_{j=1}^{n} \psi_j \cdot d'_{jk}$

et $G_j(I) = \lambda_j \cdot \min_{k \in I} d_{jk}$

Remarque 1 : U_k représente le coût de stockage et de mise à jour d'une copie du fichier au noeud k.

Remarque 2 : le coût de communication est supposé linéaire avec la quantité d'informations envoyée; Casey signale que cela peut ne pas être le cas en pratique (approximation au 1er ordre d'une fonction de coût monotonément croissante complexe)

Remarque 3 : Casey ne suppose pas que le nombre de copies du fichier est fixe (cfr Chu p.5).

3) Techniques de solution

3.1. Complexité du SFAP

Le SFAP est un problème complexe (il y a $2^n - 1$ allocations de fichier possibles): [ESWARAN 74] a démontré qu'il est NP-complet. Rappelons qu'un **problème NP-complet** (NP pour "non polynomial") est un problème pour lequel on ne connaît aucun algorithme optimal dont le temps de calcul croisse polynomialement avec la taille (par exemple le nombre de sites dans un réseau) du problème (il croît en fait au moins exponentiellement avec cette taille). Cette propriété est donc fonction de l'état de l'art dans les algorithmes optimaux de design. La méthode utilisée pour prouver qu'un problème (P) est NP-complet consiste à exhiber un problème (Q) dont on sait qu'il est NP-complet et qu'il est réductible à (P).

La croissance exponentielle du calcul signifie que le problème devient impossible à résoudre même pour des réseaux de taille modérée. Casey indique (cfr [CASEY 72]) qu'il faut 2 secondes à un IBM 360/91 pour résoudre un SFAP de 19 noeuds; remarquons (avec [GRAPA 77]) que chaque noeud additionnel multiplie le temps de calcul par 2 et qu'il faut dès lors une heure pour résoudre un réseau de 30 noeuds, 48 jours pour résoudre un réseau de 40 noeuds et 136 ans pour résoudre un réseau de 50 noeuds ! (cfr [SICKLE 77]).

Eswaran démontre que le "set covering problem" (SCP) (cfr I.3.3.2. (iv)), problème connu comme étant NP-complet (cfr [KARP 72]), est réductible au SFAP; ainsi si on sait résoudre le SFAP dans un temps polynomial, on peut également résoudre le SCP et un très grand nombre de problèmes difficiles dans un temps polynomial, ce qui est très peu probable.

[SICKLE 77] traite de la complexité de calcul de plusieurs problèmes de conception relatifs aux réseaux d'ordinateurs. Les auteurs démontrent que **les problèmes suivants sont "NP-complets"** :

(1) allocation de copies multiples d'un fichier dans un réseau d'ordinateurs (le SFAP);

(2) allocation de capacités aux lignes de communication;

(3) emplacements de concentrateurs;

(4) conception de réseaux multipoints en arbre;

(5) conception de réseaux multipoints en boucle.

Il signale que l'on peut distinguer **deux approches dans le développement d'algorithmes** relatifs aux problèmes de conception à propos des réseaux :

(i) les **algorithmes optimaux** (programmation mathématique, recherches exhaustives ... etc), qui garantissent des solutions optimales (pour des problèmes simplifiés), et leur variante, les **algorithmes d'approximation,** qui peuvent fournir un écart à l'optimalité des solutions produites;

(ii) les **algorithmes heuristiques** (stratégie de recherche en un temps polynomial), qui ne garantissent pas l'optimalité mais sont utilisés à partir de modèles détaillés.

Les approches énumératives (par exemple "branch-and-bound", technique de l'hypercube ... cfr ci-dessous) sont optimales mais impraticables lorsque le problème est grand : bien qu'on puisse réduire le nombre des énumérations explicites (cfr Casey p. 16), la complexité du problème ne cesse néanmoins de croître exponentiellement.

Les heuristiques accélèrent le traitement en limitant, ou en éliminant même, le "backtracking" des techniques énumératives, au risque de réduire la qualité de la solution.

Les résultats énoncés ci-dessus fournissent un argument théorique en faveur de la seconde approche, argument renforcé par la constatation d'une tendance vers des réseaux dont le nombre de noeuds ne cesse de croître. En faveur de l'approche (i), il faut remarquer qu'une diminution du coût, ne fût-ce que d'un pourcent, peut s'avérer substantielle (étant donné l'ordre de grandeur de ces coûts) et donc justifier, pour certaines tailles de problèmes, l'emploi d'algorithmes optimaux. Il démontre, entre autre chose, que le "node cover problem" (problème NP-complet) peut être réduit au SFAP.

Les techniques optimales (cfr [EFROYMSON 66] par exemple) étant très coûteuses en temps calcul, des heuristiques ont été développées (cfr [FRAZER 67] par exemple) qui sacrifient l'optimalité à une réduction de la quantité de calcul.

Cependant, ces diverses techniques ne sont applicables qu'à des problèmes de petite taille. Face à ce problème, divers auteurs ont développé des critères permettant de réduire la taille du problème à priori et d'accélérer les procédures de recherche (cfr I.2.). Remarquons immédiatement que l'introduction de contraintes invalide ces critères.

Le SFAP peut être résolu par les **techniques de la programmation entière** (cfr [GEOFFRION 72] pour un résumé de l'état de l'art en matière de programmation entière). Casey (cfr ci-dessous) et Levin et Morgan (cfr ci-après en I.3.) ont utilisé une méthode d'énumération sur l'ensemble réduit des solutions possibles afin de trouver l'optimum (technique de l'hypercube).

L'approche de Casey est la suivante :

(i) développer une procédure de recherche optimale simplifiée par l'intermédiaire de propriétés mathématiques de C (I);

(ii) modifier cette procédure en une heuristique lorsqu'elle s'avère trop coûteuse.

3.2. Propriétés mathématiques de la fonction de coût

3.2.1. Limites supérieures à $\# I_{opt}$

Supposons que $d_{jk} = d'_{jk}$, pour tout $(j, k) \in \{1, 2, \ldots, n\}^2$

L'auteur démontre le **théorème** suivant :

> **S'il** existe $r \leqslant n$ tel que $\Psi_j / \curvearrowright_j \geqslant 1/(r-1)$, pour tout $j = 1, \ldots, n$, **alors** tout I_r est plus coûteux que $I_{1_{opt}}$

Dès lors, (**corollaire 1**) **si** $\Psi_j / \curvearrowright_j \geqslant 1/(r-1)$ $(r \in \mathbb{N})$, pour tout $j = 1_k, \ldots, n$, **alors** $\# I_{opt} \leqslant r$; (1)

(**corollaire 2**) **si** chaque utilisateur génère au moins 50 % de son trafic sous la forme "update", **alors** $\# I_{opt} = 1$.

Remarque : ces résultats sont le reflet du compromis qui existe entre les "updates" et les "queries", l'ajout d'une copie du fichier augmentant (resp. diminuant) le coût d'"update" (resp. de "query").

---

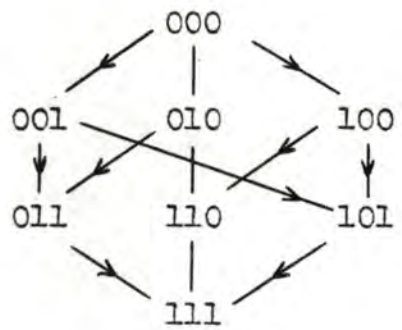
(1) Quand le quotient update/query augmente, l'avantage de considérer des copies redondantes du fichier diminue.

3.2.2. Etude du comportement de la fonction de coût lorsqu'on ajoute des copies du fichier

(i) notions d'hypercube et de graphe des coûts

Un hypercube est un graphe orienté, dont chaque sommet correspond à une allocation de F, arrangé en niveaux (les sommets du niveau k représentent tous les I_k, k = 0, 1, 2,..., n) et dont chaque arc correspond à l'ajout d'une copie unique du fichier au sommet ascendant. La figure ci-dessous est un hypercube qui représente l'ensemble des allocations de F possibles pour un réseau de trois noeuds.

—. ex. d'hypercube (n = 3).—



niv 0
niv 1
niv 2
niv 3

Le **graphe des coûts** est un hypercube auquel on associe C (I) à chaque sommet I (et $+\infty$ à I_o)

(ii) propriété du graphe des coûts

Afin de trouver I_{opt}, il suffit de suivre chaque chemin du graphe des coûts jusqu'à ce que le coût croisse, pas plus loin.

3.3. La technique "path-tracing"

L'auteur suggère de développer une **routine optimale** dite de **"path-tracing"** qui consiste, pour l'essentiel, à suivre tous les chemins du graphe des coûts (un à la fois ou tous en parallèle), un chemin étant abandonné lorsque le coût croît (cfr 3.2.2. (ii)).

Le corollaire 1 (cfr 3.2.1.) permet éventuellement de réduire la taille du graphe des coûts.

L'auteur suggère de **regrouper les noeuds voisins** en un seul noeud, d'appliquer la routine et ensuite de procéder à des calculs plus fins.

La **souplesse de cette technique** permet de l'appliquer à des problèmes sortant du cadre de la programmation linéaire (par exemple si l'on suppose que le coût d'un "update" est celui du sous-arbre le plus économique qui contient le noeud origine de l'"update" et les noeuds contenant une copie du fichier). (1)

L'auteur propose deux **exemples** (n = 5 et n = 19). Signalons que l'exemple à 5 noeuds sera très souvent repris dans de nombreux travaux ultérieurs.

L'auteur propose ensuite une **modification heuristique de la routine "path-tracing"**. Remarquons que, si l'optimum contient 40 noeuds, il faut évaluer au moins $2^{40} \simeq 10^{12}$ valeurs de la fonction de coût.

Notons A_k, l'ensemble des sommets de niveau k dont tous les sommets adjacents de niveau (k - 1) sont de coût supérieur, et supposons que la recherche au travers du graphe des coûts se fasse en parallèle. L'idée de l'heuristique est de sélectionner, pour l'étape (k + 1), un nombre limité d'éléments de A_k.

Casey propose également une autre méthode (qui a été programmée) qui peut être divisée en deux étapes. La première étape consiste à réaliser un "path-tracing" complet sur les quelques premiers niveaux du graphe des coûts pour ensuite ne suivre, dans une seconde étape, que les chemins "les plus prometteurs".

---

(1) Lors de la mise à jour des copies multiples d'un fichier, les "updates" peuvent être envoyés séquentiellement ou en parallèle.

L'auteur a testé, sur le "19 noeuds", l'algorithme heuristique avec la fonction de coût modifiée (cfr ci-dessus) : les résultats montrent la forte influence (sur le coût) de l'introduction de cette nouvelle fonction de coût.

4) Etude critique du modèle et des techniques de solution

L'**avantage** de ce modèle est sa **simplicité** : les techniques de programmation entière ou de recherche sur l'hypercube (cfr ci-dessus) sont directement applicables et de bonnes heuristiques de recherche peuvent être obtenues en "élaguant" le graphe des coûts de façon intelligente. Lorsque les coûts de communication des "queries" et des "updates" sont supposés être les mêmes, on peut déterminer une borne supérieure sur le nombre de copies du fichier dans l'allocation optimale (cfr [CASEY 72] et [COFFMAN 80]). Dans le cas de **l'allocation de fichiers multiples,** les techniques de solution développées pour le SFAP sont encore applicables si l'on suppose indépendants les accès aux divers fichiers (cfr I.4.3).

Les **désavantages** principaux de ce modèle résident dans ses hypothèses :

(1) il ne considère qu'un seul fichier à la fois;

(2) les délais de file d'attente sont ignorés;

(3) aucun niveau de performance n'est garanti (temps de réponse, disponibilité, fiabilité,..., etc);

(4) pas de contrainte de capacité;

(5) le routage est supposé statique;

(6) les coûts de communication sont supposés varier linéairement avec la quantité d'information envoyée;

(7) l'utilisation du fichier est statique;

(8) des bornes sur le nombre optimal de fichiers ne sont disponibles que dans le cas où les coûts de communication des "updates" et des "queries" sont égaux (cfr [DOWDY 82]).

I.2. ETUDES ULTERIEURES SUR LE SFAP

Suite à la démonstration, par ESWARAN, de la NP-complétude du SFAP (qui résulte de la nature exponentielle des algorithmes optimaux, cfr ci-dessus), plusieurs auteurs ont tenté de réduire ou de faire face à la complexité du problème en proposant :

(i) des critères permettant de réduire la complexité du SFAP;

(ii) de simplifier le SFAP en considérant des réseaux particuliers mais néanmoins utiles en pratique;

(iii) des techniques de solution alternatives à celles de Casey.

1) Critères permettant de réduire la complexité du SFAP

(i) **Efroymson et Ray** (cfr [EFROYMSON 66]) ont étudié le "Single Plant Location Problem" (SPLP), une classe de problèmes dont on peut démontrer qu'elle est isomorphe au SFAP (cfr [RAMAMOORTHY 83]); ainsi, les techniques de solution et les conditions diverses qui s'appliquent à l'une de ces classes s'appliquent également et directement à l'autre. Considérons un ensemble de m usines qui peuvent fournir n clients de biens d'un seul type. Supposons qu'il n'y ait aucune contrainte sur la quantité envoyée d'une quelconque usine. Le SPLP consiste à déterminer l'emplacement des usines ($\{ y_i : i = 1,\ldots,m \}$) et la partie (x_{ij}) de la demande (D_j) du client j ($j = 1,\ldots,n$) fournie par l'usine i ($i = 1,\ldots,m$), qui minimisent la fonction suivante:

$$\sum_{i,j} c_{ij} x_{ij} + \sum_{i} f_i y_i ,$$

sous les contraintes,

$$\sum_{i \in N_j} x_{ij} = 1 \qquad (j = 1,\ldots,n)$$

$$0 \leq \sum_{j \in P_i} x_{ij} \leq n_i\, y_i \leq 1 \quad (i = 1,\ldots,m)$$

$$y_i = 0 \text{ ou } 1 \qquad (i = 1,\ldots,m),$$

où $c_{ij} = t_{ij}.D_j$, t_{ij} = le coût unitaire de transport de l'usine i au client j, f_i (≥ 0) = le coût fixe associé à l'usine i, P_i = l'ensemble des indices des clients qui peuvent être fournis par l'usine i, N_j = l'ensemble des indices des usines qui peuvent fournir le client j et $n_i = |P_i|$.

Signalons les autres problèmes de recherche opérationnelle du type "Plant Location" définis par [RAMAMOORTHY 83] , qui consistent en des variations du SPLP.

Nous décrivons brièvement ci-dessous (cfr I.2.3.(i)) la technique de solution proposée par les auteurs. Contentons-nous de signaler, pour l'instant, qu'il s'agit d'un **"branch-and-bound"** ("b&b") (cfr [GEOFFRION 72], [LAWLER 66], [MITTEN 70],...etc) à propos duquel les auteurs suggèrent diverses **simplifications** permettant de réduire le nombre de branches pouvant émaner d'un noeud quelconque de l'arbre de recherche. Une première simplification détermine une borne inférieure sur la réduction de coût correspondant à l'ouverture d'une usine; si cette borne est positive, l'usine sera fixée "ouverte". Une autre simplification détermine une borne supérieure sur la réduction du coût correspondant à l'ouverture d'une usine. Si cette borne est négative, l'usine sera fixée "fermée". Afin de concrétiser quelque peu le discours, nous proposons ci-dessous une formulation précise de la première simplification :

si $\sum_{j \in P_i} \Delta_{ij} - f_i > 0$

Alors $y_i = 1$ pour toutes les branches émanant du noeud considéré
où $\Delta_{ij} = \min_{\substack{k \in K_1 \cup K_2 \\ k \in N_j}} \left\{ \max (C_{kj} - C_{ij}, 0) \right\}$, $i \in K_2$, $j \in P_i$

(ii) **Grapa et Belford** (cfr [GRAPA 77]) démontrent trois théorèmes, relatifs au SFAP, dont le but est de réduire la taille du problème en indiquant **à priori** que tels et tels noeuds doivent (resp. ne peuvent) être inclus dans toute (resp. aucune) allocation optimale. Ils précisent que la procédure de recherche développée par Casey (cfr [CASEY 72]) - bien que certains théorèmes démontrés par ce dernier permettent de l'accélérer - reste caractérisée par un temps de calcul qui croît généralement de façon exponentielle avec le nombre de noeuds. Nous nous contenterons ici d'énoncer, pour l'exemple, le théorème suivant.

Idée : si le coût nécessaire pour maintenir une copie locale du fichier est supérieur aux gains que l'on pourrait en attendre, alors il ne faut pas maintenir une telle copie.

Enoncé : **si** $U_i > \sum_{j=1}^{n} \lambda_j (\max_k d_{jk} - d_{ji})$
(où $i \in \{1, 2, \ldots, n\}$)

alors aucune allocation optimale de plus d'un noeud ne peut contenir le noeud i (pour les notations, cfr p. 10).

Les auteurs font remarquer que certains calculs nécessaires à l'évaluation des conditions de ces trois théorèmes peuvent être très lourds : il faudrait alors ne les exécuter que de façon sélective. En partant des idées intuitives sous-jacentes à leurs théorèmes, ils suggèrent l'utilisation de certaines quantités comme facteurs de recherche dans des heuristiques, ou encore afin d'identifier les situations dans lesquelles des théorèmes réclamant des calculs plus lourds peuvent s'avérer néanmoins utiles (par exemple U_i / λ_i est une bonne mesure de la probabilité que le noeud i doive se trouver dans une allocation optimale).

Ils reprennent les deux exemples de Casey (cfr [CASEY 72]), le "5 noeuds" et le "19 noeuds", dont la taille est réduite, à priori, respectivement, à 3 et à 13 noeuds.

Ils font remarquer que, si l'on ajoute des contraintes au problème, les théorèmes ne sont pas immédiatement applicables (il faut les adapter). Cependant, ils pensent qu'il faut essayer, dans tout problème d'allocation de fichier(s), de trouver le plus d'information possible sur l'optimum, à priori : cela peut rendre praticable un problème en réduisant sa taille de façon à la rendre acceptable.

(iii) **Kollias et Hatzopoulos** (cfr [KOLLIAS 81a]) présentant des critères plus forts que ceux introduits par les théorèmes de Grapa et Belford, montrent comment on peut les appliquer **à priori** et, dans le cas où la taille du problème n'est pas réduite de façon significative, comment ils peuvent être **incorporés à l'heuristique** de Casey (ou à une autre) afin d'en améliorer la performance.

Ils signalent que le théorème 1 de [GRAPA 77] est plus faible que la première simplification de [EFROYMSON 66] (cfr p. 20). Ils proposent un critère semblable au théorème 3 de [GRAPA 77] à la différence qu'il peut devenir partie intégrante de tout algorithme.

(iv) **Ramamoorthy et Wah** (cfr [RAMAMOORTHY 83]) montrent que le théorème 2 de [GRAPA 77] peut être combiné avec la simplification de [EFROYMSON 66] pour former une condition plus forte.

On dispose donc d'un ensemble de conditions pour le placement ou le "non-placement" à priori et/ou à chaque itération d'une copie du fichier en un noeud non alloué (cfr résumé de ces conditions dans [WAH 84] et [RAMAMOORTHY 83]). Ces conditions peuvent être intégrées aux techniques de solution optimales du SFAP afin d'en réduire la complexité; elles facilitent également l'élaboration d'heuristiques de bonne qualité.

2) Le SFAP sur des réseaux particuliers

RAMAMOORTHY 83 cite le travail de **Kijuno et al.** (cfr [KIJUNO 81]) dans lequel les auteurs démontrent que le problème du placement de fichiers multiples sur un réseau arborescent sans contrainte de capacité de stockage ni coût de mise à jour, peut être résolu en un temps polynomial.

Missikoff et Pagli (cfr [MISSIKOFF 80]) montrent que, dans le cas d'un réseau symétrique et homogène (1), le problème d'allocation de fichiers multiples n'est pas NP-complet. Ils présentent une méthode permettant d'obtenir une allocation optimale de complexité $O(n.m)$ (où m est le nombre de fichiers distincts). Dans un tel type de réseau, le coût de stockage d'un fichier est indépendant du noeud et le coût de communication entre deux noeuds est constant (ainsi par exemple l'ajout d'une copie d'un fichier en un noeud ne modifie pas le coût d'accès des autres noeuds à ce fichier).

3) Techniques de solution alternatives à celles de Casey

(i) Nous reprenons l'étude d'**Efroymson et Ray** (cfr [EFROYMSON 66]) relative au SPLP (cfr ci-dessus p 20).

Ils signalent que de nombreux auteurs ont développé des **heuristiques** qui fournissent de "bonnes solutions" au SPLP et sont très efficaces du point de vue temps CPU (cfr [KUEHN 63], [MANNE 64]). Le but des auteurs est de proposer une **méthode optimale de solution** au SPLP qui serait compétitive au niveau de la complexité de calcul avec ces heuristiques.

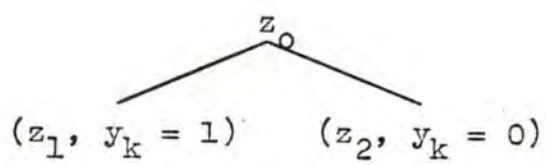
---

(1) Réseau dont les noeuds sont capables de performances semblables, peuvent être interconnectés dans n'importe quelle configuration et utilisent des lignes de communication à coût de transmission constant.

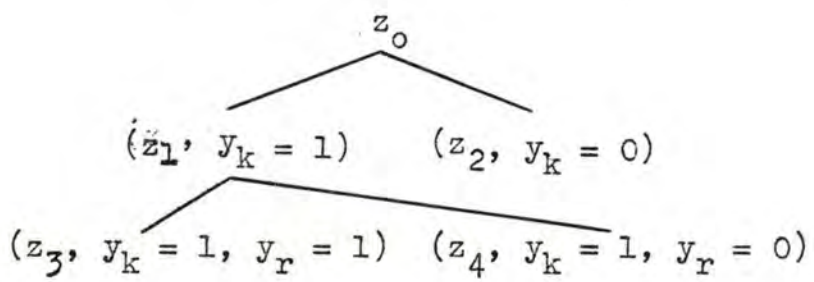
Ils utilisent pour ce faire la méthode "branch-and-bound" ("b&b") qui est une technique de programmation entière mixte, finie, à but général, qui a été développée par A.H. LAND et A.G. DOIG (cfr [LAND 60] et les références citées p. 2I).

L'idée de base est de résoudre une suite de programmes linéaires sans contrainte d'intégralité (CI) qui fournissent des limites inférieures (resp. supérieures) s'il s'agit d'une minimisation (resp. maximisation), sans cesse améliorées, à la valeur de la solution optimale du problème avec CI.

Illustrons le "b & b" sur le SPLP. On résoud le SPLP sans CI sur $\{y_i\}$; cela donne une solution z_o; si tous les y_i sont entiers, alors z_o est la solution optimale au SPLP, sinon on sélectionne un y_k fractionnaire quelconque et on résoud le SPLP sans CI, successivement avec y_k fixé à 1 puis à 0, ce qui donne les solutions z_1 puis z_2 ($z_1 \geq z_o$ et $z_2 \geq z_o$); il est clair que $\bar{z} = \min(z_1, z_2)$ est une nouvelle limite inférieure sur la valeur de la solution (du SPLP avec CI). On amorce ainsi la construction d'un arbre :



et l'on se "branche sur le noeud déterminé par $\bar{z}$"; on sélectionne un autre y_r fractionnaire quelconque (parmi les y_i correspondant à z_1) et l'on résoud le SPLP sans CI, successivement avec ($y_k = 1$, $y_r = 1$) puis ($y_k = 1$, $y_r = 0$), ce qui donne les solutions z_3 puis z_4.



On se branche sur $\bar{\bar{z}} = \min (z_3, z_4, \max (z_1, z_2))$, qui est une nouvelle limite inférieure améliorée. Il suffit de garder trace des noeuds terminaux. Le processus se termine quand un noeud est atteint où tous les y_i sont entiers et dont la valeur est inférieure ou égale à celle de tout autre noeud terminal. Quand un noeud est infaisable, aucune branche ne peut en émaner.

La grande **difficulté** du "b & b" réside au niveau du calcul et du stockage des noeuds; il faut formuler le problème de telle façon que le programme linéaire sans CI puisse être résolu efficacement.

Les auteurs signalent deux caractéristiques qui permettent de réduire la quantité de stockage ainsi que le temps de calcul (entre autre via la connaissance d'une "bonne solution"). Ils ont testé plusieurs cas où n = 200 et m = 50, le temps de calcul moyen étant de 10 minutes (sur IBM 7.094).

WAH applique l'algorithme "branch-and-bound" ("b & b") avec la stratégie "best-first search" (cfr [WAH 84]) à l'exemple de 5 noeuds de Casey (cfr [CASEY 72]). Il utilise en fait l'algorithme de [EFROYMSON 66] dont il simplifie les calculs par l'intermédiaire des critères décrits ci-dessus. Il explique de façon intuitive le principe de la technique "b & b" (énumération exhaustive). Il insiste sur le fait que les méthodes énumératives optimales demeurent impraticables lorsque le problème est grand, quels que soient les moyens employés pour réduire le nombre des énumérations explicites.

(ii) **Chandy et Hewes** (cfr [CHANDY 76]) considèrent un modèle quelque peu différent de celui de Casey (cfr p. 10) : ils supposent que les coûts de mise à jour dépendent seulement d'où est alloué le fichier, et sont indépendants du nombre et de l'emplacement des autres copies :

$$\min \sum_{j=1}^{n} (U_j x_j + \sum_{k \neq j} \lambda_j d_{jk} y_{jk})$$

sous les contraintes : $y_{jk} \leq x_j \qquad (j, k = 1, \ldots, n \text{ et } j \neq k)$

$$\sum_{j \neq k} y_{jk} + x_k = 1 \qquad (k = 1, \ldots, n)$$

$$y_{ij},\ x_j = 0 \text{ ou } 1 \qquad (CI)$$

où $x_j = 1$ si une copie du fichier est stockée au noeud j (= 0 sinon)

$y_{jk} = 1$ si un "query" de j accède à une copie en k (= 0 sinon)

Notons (P) ce problème et (P') $\equiv$ (P) où on ignore les contraintes d'intégralités (CI). La fonction objective et toutes les contraintes de (P') sont linéaires; la structure spéciale de (P') permet de le résoudre efficacement (place en mémoire et calcul) par les techniques classiques de la programmation linéaire. Toute solution de (P') sert de borne inférieure (INF) à la solution de (P). Les auteurs proposent également une heuristique appelée **"m-distance heuristic"** qui fournit une borne supérieure (SUP) à la solution de (P). Cette heuristique entière retenue consiste essentiellement en une technique itérative; initialement, une copie du fichier est allouée au hasard en un noeud; toutes les allocations de fichier "à une distance de m" sont évaluées (i.e. on ajoute et/ou on soustrait m copies au plus par rapport à l'allocation précédente); l'allocation la moins coûteuse est sélectionnée et le processus continue jusqu'à ce qu'aucune allocation "à distance m" moins chère n'existe. Une distance de 2 semble être excellente ! Les auteurs suggèrent de résoudre (P) par un **"branch-and-bound"** ("b & b") où sont éliminés les noeuds tels que (INF $\geq$ SUP). Ils font remarquer que l'heuristique peut servir à fournir une bonne **base initiale** (au sens de la programmation linéaire) à (P'). Sur l'ensemble des cas résolus, 95 % des solutions de (P') satisfont les CI ! Ainsi le "b & b" se termine-t-il en une étape dans 95 % des cas . [SPINETO 76] signale le même phénomène sur des problèmes d'optimisation similaires ! Il semble donc que certains problèmes combinatoires discrets puissent être approchés par des problèmes continus simples avec d'excellents résultats. Si l'on peut se contenter d'une solution présentant une déviation δ par rapport à l'optimum, les auteurs suggèrent de procéder de la façon suivante :

a) obtenir une solution faisable par l'heuristique;
b) calculer une limite supérieure sur δ via (P');
c) utiliser le "b & b" si la précision n'est pas satisfaisante.

Les auteurs présentent les δ positifs (resp. négatifs) des solutions heuristiques (resp. à (P')) relatifs à 172 réseaux de 3 à 11 noeuds.

(iii) **Kollias et Hatzopoulos** (cfr [KOLLIAS 81[a]] et p 23) montrent comment les critères qu'ils présentent peuvent être incorporés à l'heuristique (en deux étapes) de Casey (cfr p.16). Ces critères permettent d'éviter un "path-tracing" complet lors de la première étape (sur l'exemple à 19 noeuds, Casey vérifie 171 noeuds du niveau 2 de l'hypercube, alors qu'il suffit d'en vérifier 113 en utilisant le critère 3). Les auteurs suggèrent diverses règles (fondées sur les critères) permettant de déterminer les chemins à suivre lors de la seconde étape.

(iv) **Ramamoorthy et Wah** (cfr [RAMAMOORTHY 83]) proposent une "**greedy heuristique**" (cfr [FISHER 80] p. 722) fondée sur les critères développés ci-dessus en I.2.1. Les auteurs proposent le schéma suivant. A tout instant, les n noeuds du réseau peuvent être partitionnés en trois ensembles K_0, K_1 et K_2, où K_0 (resp. K_1) représente l'ensemble des noeuds sans copie (resp. avec copie) du fichier, et K_2, l'ensemble des noeuds non encore alloués. A tout instant $K_0 \cup K_1 \cup K_2 = \{1, 2, \ldots, n\}$ et initialement $K_0 = K_1 = \emptyset$ et $K_2 = \{1, 2, \ldots, n\}$. Initialement, on applique le théorème 3 de [GRAPA 77] . Lors d'une itération quelconque, on sélectionne un noeud de K_2 et on décide de le mettre dans K_0 ou dans K_1. On constate donc qu'il y a $2 . |K_2|$ extensions possibles de l'allocation partielle précédente. On calcule une **valeur représentative** pour chacune d'elles (cette valeur doit estimer le coût minimum d'une allocation contenant l'extension considérée).

Sur base de ces valeurs, un **critère** sélectionne un noeud de K_2 et décide de le mettre dans K_0 ou dans K_1. On applique ensuite les conditions de pré-allocation décrites par [EFROYMSON 66] et [GRAPA 77] et on répète le processus jusqu'à ce que $K_2 = \emptyset$. On constate que cet algorithme ne prévoit aucun "backtracking" sur une allocation partielle antérieure, d'où l'importance du critère de sélection et du calcul de la valeur représentative retenus. Ils proposent deux critères et deux calculs différents, ce qui donne quatre versions distinctes de l'heuristique. Ils testent ces quatre versions sur des exemples dont on connaît les solutions optimales, et les comparent également à l'heuristique de type **"add-drop"** développée dans [KEUHN 63] (cfr également [MAHMOUD 76]). Le principe de ce type d'algorithme est le suivant. On trouve tout d'abord une allocation du fichier faisable (i.e. satisfaisant les éventuelles contraintes). On essaie ensuite d'améliorer le coût total du système par des additions et des suppressions successives de copies du fichier. Quand une solution faisable de coût inférieur est trouvée, elle est adoptée comme nouvelle solution de départ et le processus continue. Un optimum local est éventuellement atteint pour lequel aucune addition ni suppression ne peut réduire le coût du système. Le processus entier peut être répété avec différentes solutions faisables de départ et plusieurs optima locaux peuvent ainsi être obtenus, la solution finale correspondant au minimum de ceux-ci. Les auteurs comparent les temps de calcul et les déviations par rapport aux solutions optimales de ces cinq heuristiques (dont la complexité est en $O(n^4)$). Ils suggèrent de les combiner et soulignent les extensions possibles de cette technique de solution aux problèmes d'allocation de fichiers plus généraux ainsi qu'à d'autres problèmes NP-complets.

Les auteurs citent encore d'autres techniques de solution développées pour le SPLP et que l'on peut utiliser pour résoudre le SFAP.

Parmi celles-ci, nous soulignons encore :

(I) l'algorithme "steepest ascent" (cfr [FISHER 80]);

(2) la méthode du sous-gradient (idem);

(3) l'algorithme de recherche directe ou d'énumération implicite développé dans [SPIELBERG 69].

Nous reviendrons plus en détail (en I.4.I)) sur l'article de Fisher et Hochbaum ([FISHER 80]).

I.3. UNE PRESENTATION DES TRAVAUX DE LEVIN ET MORGAN RELATIFS AU PROBLEME DE L'ORGANISATION DES BASES DE DONNEES DISTRIBUEES SUR LES RESEAUX D'ORDINATEURS

(Références principales : [LEVIN 75] , [MORGAN 77] , [LEVIN 78] et [LEVIN 82])

1) Introduction

Les auteurs constatent l'apparition d'un nombre croissant de **systèmes d'ordinateurs** distribués géographiquement et connectés par des lignes de communication à haute capacité, dont l'exemple le mieux connu est, sans doute, le réseau ARPANET (cfr [KLEINROCK 76]). Ces réseaux permettent le **partage de ressources** spécialisées et coûteuses (hardware, software, bases de données... etc) et facilitent la **collaboration** entre chercheurs séparés géographiquement et étudiant le même problème ou, plus généralement, entre organisations indépendantes ayant mêmes structures et pouvant être motivées par un tel partage.

Le **problème étudié** est celui de la minimisation du "coût de fonctionnement" d'une base de données distribuée partagée par une communauté d'utilisateurs interconnectés au travers d'un réseau informatique (dont la topologie est donnée, i.e. est fixe), et en particulier, celui du placement de fichiers de données, et de programmes qui utilisent ces fichiers, dans un tel réseau. Ils proposent un **cadre de recherche** (incluant le SFAP comme cas particulier) relatif à ce problème, dans lequel ils situent leurs propres travaux (cfr [LEVIN 75]). Nous donnons ci-dessous un premier aperçu des dimensions de ce cadre de recherche.

Morgan et Levin sont les premiers à prendre en compte le fait que les demandes d'accès aux fichiers de données sont générées par des programmes (**dépendance entre fichiers et programmes**).

Telle transaction exécutera d'abord tel fichier programme avant de traiter tel fichier de données : dans leurs modèles, les "queries" et les "updates" sont traités par des programmes avant d'être envoyés vers les bases de données correspondantes. De plus, les fichiers programmes ne peuvent pas nécessairement être exécutés en n'importe quel noeud du réseau (problèmes de compatibilité). Morgan et Levin examinent simultanément les problèmes de l'allocation des fichiers de données et des programmes (dans un réseau tel ARPANET) : la principale contribution du modèle qu'ils proposent est de considérer et d'incorporer les fichiers de programmes dans l'allocation des fichiers de données.

La nature dynamique des accès aux données est aussi incluse dans leurs études. Dans ce cas, les fichiers ont la possibilité d'émigrer au cours du temps au travers du réseau afin de s'adapter aux changements des demandes d'accès.

Ils examinent également la situation, extrêmement courante en pratique, dans laquelle les taux de demandes d'accès ne sont pas connus à l'avance et doivent donc être estimés.

Les auteurs ont adopté le **point de vue de l'utilisateur.** Ils supposent être sous la juridiction d'un gestionnaire de réseau fournissant des services (calculs, communications, mémorisations... etc) et imposant un système de prix. Ils considèrent les ressources du réseau comme ayant une configuration et un coût prédéfinis. On pourrait opposer, à cette approche, celle de [CERI 82^{b}], par exemple, qui adopte un point de vue d'optimisation globale, c'est-à-dire d'optimisation de la performance du système distribué considéré comme un tout. Un exemple caractéristique où ces points de vue divergent est celui des contraintes et des coûts de stockage. Pour le premier point de vue, la tâche de fournir suffisamment d'espace mémoire aux utilisateurs individuels devrait être de la juridiction du gestionnaire du réseau; il est donc inconcevable d'imposer des contraintes de stockage à n'importe quel utilisateur.

Pour le second point de vue, le stockage est considéré comme une contrainte et non comme un coût : les coûts de stockage sont en fait des coûts d'acquisition et de maintenance de mémoires, l'ajout d'une copie de fichier à un noeud ne coûtant rien si la capacité de stockage de ce noeud n'est pas dépassée. Les auteurs supposent également que diverses spécifications du réseau sont disponibles (ou le seront dans les réseaux futurs) comme par exemple le délai maximum sur une ligne de communication entre tel et tel noeud.

Nous présentons, ci-dessous, le cadre de recherche ainsi que les modèles et techniques de solution proposés par les auteurs.

2) Un cadre de recherche

Considérons ([LEVIN 75] et [MORGAN 77]) un réseau et une base de données constituée de fichiers et de programmes. Chaque noeud du réseau demande les services de fichiers et de programmes par l'intermédiaire de transactions. Les auteurs distinguent deux types de transactions : les "queries" et les "updates". Une transaction est d'abord transmise au programme qui lui est associé et de ce programme, un "query" (lecture d'information depuis une copie unique) est envoyé à la copie du fichier (concerné) la plus proche, alors qu'un "update" (écriture d'information sur toutes les copies) est envoyé à chaque copie du fichier (cfr figures 1^a et 1^b ci-dessous).

On suppose qu'un quelconque programme peut demander accès à n'importe quel fichier et que les demandes d'accès pour un fichier sont indépendantes des demandes d'accès pour les autres fichiers.

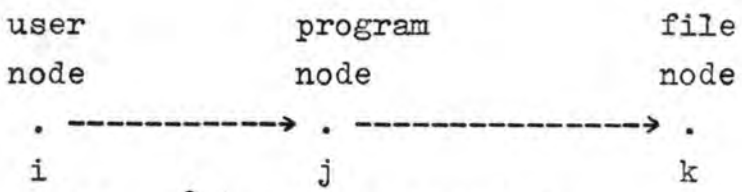


fig. 1^a. query processing

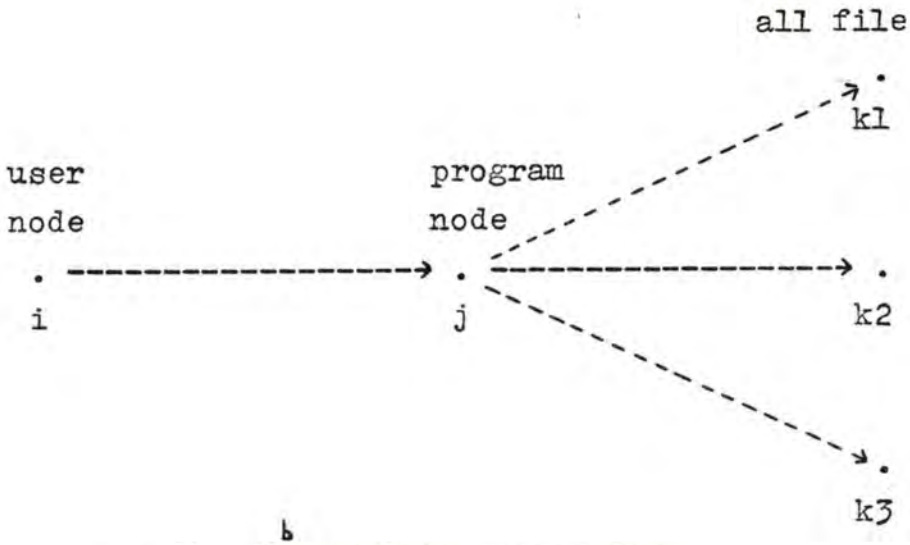


fig. 1 . update processing

Nous pouvons, dès à présent, préciser quelque peu la nature du problème étudié : il s'agit de déterminer la distribution des fichiers et des programmes sur le réseau qui minimise les coûts de fonctionnement de la BDD (coûts de stockage et coûts de communication).

Les auteurs distinguent **trois dimensions** dans le problème étudié ([LEVIN 75]):

(i) le niveau de partage (partage des données ou partage des données **et** des programmes);

(ii) le "comportement des types d'accès" (statique ou dynamique);

(iii) le niveau d'information sur le "comportement des types d'accès" (information complète ou partielle).

En **"partage des données"**, chaque noeud dispose d'une copie de chaque programme, les fichiers de données étant partagés (i.e. on peut y accéder à partir de noeuds éloignés, la requête étant traitée au noeud qui l'a générée). En **"partage des données et des programmes"**, tant les fichiers de données que les fichiers de programmes sont accessibles à partir de noeuds éloignés, l'allocation optimale des premiers dépendant évidemment de l'allocation des seconds (via la distribution des requêtes); on parle alors de "dépendance programmes-données".

On peut traduire cette notion de dépendance de la façon suivante : l'accès à un fichier f_i dépend du programme p_i, c'est-à-dire que pr $[f_i/p_i] \neq$ pr $[f_i/p_j]$ où p_i et p_j sont des programmes différents. Un "**comportement** des types d'accès" **dynamique** réfère à une situation où les types d'accès (i.e. les taux de demandes d'accès émanant de chaque noeud pour chaque fichier) varient au cours du temps (par opposition au comportement statique). On dispose d'une **information complète** sur le comportement des types d'accès si ceux-ci sont parfaitement connus à l'avance (dans le cas contraire, on parle d'information partielle).

Nous pouvons représenter les divers jeux d'hypothèses de la façon suivante :

		partage des données		partage des données et des programmes	
		statique	dynamique	statique	dynamique
information	complète	1	2	5	6
	partielle	3	4	7	8
		comportement des types d'accès		comportement des types d'accès	

fig. 2. Jeux d'hypothèses

Les auteurs rangent les travaux de Chu ([CHU 69]), de Casey ([CASEY 72]), de Whitney ([WHITNEY 70]), de Ramamoorthy et de Wah ([RAMAMOORTHY 79^a]) dans le jeu d'hypothèses 1; ils présentent les résultats de leurs propres recherches à propos des jeux 5([MORGAN 77]), 6([LEVIN 78]), 7 et 8([LEVIN 82]) (en faisant remarquer que les modèles développés à ce niveau sont applicables aux jeux 1 à 4, en relâchant simplement l'hypothèse de dépendance).

Remarque : la distinction entre les fichiers de données et les fichiers de programmes s'impose dans un réseau hétérogène, ces derniers étant bien moins aisément transportables d'un site à un autre (problèmes de la conversion initiale et de la maintenance continue) : il est donc sans doute plus intéressant, dans un tel réseau, de partager les programmes plutôt que de les dupliquer en chaque noeud (on introduit alors des dépendances entre programmes et données !) ([LEVIN 75]).

3) Etude du jeu d'hypothèses n° 5 ([LEVIN 75] et [MORGAN 77])

Nous donnons une description détaillée du modèle du jeu 5 (cfr 3.1. ci-après) afin de permettre une présentation plus précise, plus fine et plus rapide de divers modèles proches de ce dernier ou le complétant (cfr I.4).

Nous présentons ensuite les techniques de solution proposées par les auteurs (cfr 3.2. ci-dessous) : la décomposition du problème en plusieurs problèmes d'allocation d'un fichier unique (cfr 3.2. (i)), la notion d'hypercube (cfr 3.2.(ii)), l'introduction de contraintes supplémentaires (cfr 3.2.(iii)), la procédure de solution (cfr 3.2.(iv)) et divers exemples et résultats (cfr 3.2.(v)).

3.1. Modèle du jeu 5

Considérons un réseau de N noeuds et une BD de F fichiers (i.e. fichiers de données) et P programmes. Chaque noeud du réseau demande les services de programmes et de fichiers via deux types de transactions, les "queries" et les "updates" (cfr 2) ci-dessus). Nous décrivons successivement les paramètres, les variables, la fonction objective et les contraintes du modèle.

(i) Les paramètres

Remarque : les index i, j, k, p, f sont ceux, respectivement, des noeuds qui émettent des demandes d'accès, des noeuds où sont stockés les programmes, des noeuds où sont stockés les fichiers, des programmes, des fichiers.

d_{ipf} (resp. d'_{ipf}) = "query traffic" (resp. "update traffic") du noeud i vers le fichier f via le programme p;

Remarque : modélisation simpliste des transactions (cfr [CERI 83])

C_{ij} (resp. C'_{ij}) = coût de communication par unité de "query" (resp. d'"update") du noeud i vers le noeud j;

σ_{kf} = coût de stockage du fichier f au noeud k;

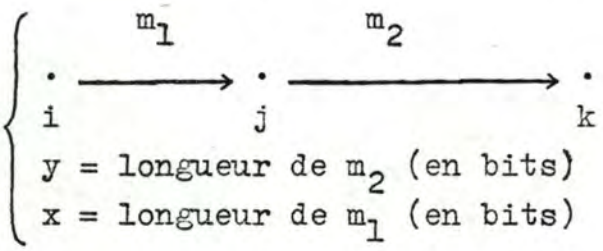
σ'_{jp} = coût de stockage du programme p au noeud j;

J_p = ensemble des noeuds auxquels le programme p peut être exécuté;

α (resp. β) = facteur d'expansion pour un "query message" (resp. un "update message")

p f (cfr fig 1[a])

où $\alpha = y/x$ avec (idem avec β)



(ii) Les variables

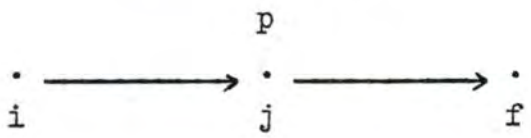
Ces variables déterminent les emplacements des fichiers et des programmes ainsi que la discipline de routage.

y_{kf} = 1 si une copie de f est stockée en k (= 0 sinon);

y'_{jp} = 1 si une copie de p est stockée en j (= 0 sinon);

x_{jkf} = la proportion des transactions de j vers f qui sont routées vers le noeud k

x^f_{ijp} = la proportion des transactions de i vers f via le programme p qui sont routées vers le noeud j



Remarque 1 : ces variables déterminent les proportions de routage, la route particulière prise par telle transaction pouvant être sélectionnée au moment de son exécution, en fonction de caractéristiques dynamiques ignorées du modèle.

Remarque 2 : l'indice f permet de différencier les deux situations suivantes :

$x^f_{ij_1p}$ et $x^{f'}_{ij_2p}$

Remarque 3 : on s'attend à ce que :

$$\begin{cases} x_{jkf} > 0 \Rightarrow y_{kf} = 1 \\ x^f_{ijp} > 0 \Rightarrow (j \in J_p \text{ et } y'_{jp} = 1) \end{cases}$$

Notations : $\rho_{jf} = \sum_{i,p} \lambda_{ipf} \cdot x^f_{ijp}$

("query traffic" vers f traité en j);

$\Psi_{jf} = \sum_{i,p} \lambda'_{ipf} \cdot x^f_{ijp}$

("updating traffic" vers f traité en j).

(iii) La fonction objective

L'objectif du modèle est de minimiser

C = coûts de communication + coûts de stockage

où (coûts de communication) = $C_1 + C_2 + C_3 + C_4$

(coûts de stockage) = $C_5 + C_6$ et où :

$$C_1 = \sum_{i,j,p,f} \lambda_{ipf} \cdot x^f_{ijp} \cdot c_{ij}$$

(coût de communication des "queries" depuis les noeuds origines vers les programmes);

$$C_2 = \sum_{i,j,p,f} \lambda'_{ipf} \cdot x^f_{ijp} \cdot c'_{ij}$$

(coût de communication des "updates" depuis les noeuds origines vers les programmes);

$$C_3 = \sum_{j,k,f} \rho_{jf} \cdot x_{jkf} \cdot \alpha \cdot c_{jk}$$

(coût de communication des "queries", des programmes vers les fichiers);

$$C_4 = \sum_{j,k,f} \gamma_{jf} \cdot y_{kf} \cdot \beta \cdot c'_{jk}$$

(coût de communication des "updates", des programmes vers les fichiers);

$C_5 = \sum_{f,k} \sigma_{kf} \cdot y_{kf}$ (coût de stockage des fichiers);

$C_6 = \sum_{j,p} \sigma'_{jp} \cdot y'_{jp}$ (coût de stockage des programmes).

Remarque : à l'optimum, on peut démontrer que x_{jkf}, $x^f_{ijp} \in \{0,1\}$, pour tout i,j,k,f,p.

(iv) Les contraintes

- Il faut au moins une copie de chaque fichier et de chaque programme :

$$\sum_j y'_{jp} \geq 1 \qquad \forall\, p = 1,\ldots, P; \qquad (C_1)$$

$$\sum_k y_{kf} \geq 1 \qquad \forall\, f = 1,\ldots, F. \qquad (C_2)$$

- Toute transaction vers un quelconque fichier via un quelconque programme et depuis un noeud quelconque, doit avoir une route définie :

$$\sum_j x^f_{ijp} \geqslant 1 \qquad \forall i = 1,\ldots, N; \quad \forall p = 1,\ldots, P; \quad \forall f = 1,\ldots, F; \qquad (C_3)$$

$$\sum_k x_{jkf} \geqslant 1 \qquad \forall j = 1,\ldots, N; \quad \forall f = 1,\ldots, F. \qquad (C_4)$$

- Il faut que les fichiers et programmes appropriés soient bien stockés en accord avec les routes définies :

$$\sum_i x^f_{ijp} \leqslant N \cdot y'_{jp} \qquad \forall j = 1,\ldots, N; \quad \forall p = 1,\ldots, P; \quad \forall f = 1,\ldots, F; \qquad (C_5)$$

$$\sum_j x_{jkf} \leqslant N \cdot y_{kf} \qquad \forall k = 1,\ldots, N; \quad \forall f = 1,\ldots, F. \qquad (C_6)$$

- Il faut que tout programme ne soit stocké qu'en un noeud où il peut être exécuté :

$$y'_{jp} = 0 \qquad \forall j \notin J_p, \quad p = 1,\ldots, P \qquad (C_7)$$

- Les variables du modèle sont binaires (cfr remarque en 3.1. (ii) ci-dessus) (C_8)

3.2. Techniques de solution ([MORGAN 77])

(i) Décomposition du problème

Le problème, tel qu'il est formulé en 3.1. ci-dessus, appartient à une classe de problèmes de programmation 0-1 non linéaire; une linéarisation brutale de ce problème le rendrait tout bonnement intraitable.

Le modèle suppose que l'allocation de tel fichier de données est indépendante de l'allocation de tel autre fichier de données. Il ne contient aucune contrainte de délai (cfr 3.2. (iii)) ni aucune contrainte de stockage (cfr 1) ci-dessus).

Les emplacements des programmes n'étant pas fixés, on ne peut déterminer le "query traffic" des programmes sur les BDs pour chaque noeud.

Levin a montré (cfr [LEVIN 74]), qu'en supposant les coûts de stockage des programmes négligeables par rapport aux coûts de stockage des fichiers de données (dans le monde des grandes BDs (par exemple de 10^8 à 10^{12} bits), les tailles des données sont nettement supérieures à celles de la plupart des programmes), le problème de minimisation multi-fichiers peut être décomposé en plusieurs problèmes de minimisation portant chacun sur un seul fichier. Il suffit de stocker une copie de chaque fichier programme en chaque noeud où il peut être exécuté. Il s'agit alors de résoudre le problème suivant (cfr [LEVIN 74]) :

$$\min C = \sum_{f=1}^{F} \min_{K_f \in Y_f} C(K_f)$$

où $K_f \in [0,1]^N$ avec $(K_f)_k = \begin{cases} 1 \iff y_{kf} = 1 \\ 0 \end{cases}$

Y_f = l'ensemble des K_f satisfaisant les contraintes (cfr 3.1. (iv))

$C(K_f)$ = le coût associé à l'allocation K_f du fichier f
$= Q(K_f) + U(K_f) + S(K_f)$,

où $Q(K_f) = \sum_{p \in Q} \sum_{i} \lambda_{ipf} \cdot \min_{j \in Q_p} (C_{ij} + \alpha \min_{k \in K_f} C_{jk})$

(coûts de communication des "queries")

$U(K_f) = \sum_{p \in U} \sum_{i} \lambda'_{ipf} \cdot \min_{j \; U_p} (C'_{ij} + \beta \sum_{k \in K_f} C'_{jk})$

(coûts de communication des "updates")

$S(K_f) = \sum_{k \in K_f} \sigma'_{kf}$

(coûts de stockage)

avec Q (resp. U) = l'ensemble des programmes "query" (resp. "update")

Q_p (resp. U_p) = l'ensemble des noeuds auxquels le programme "query" (resp. "update") p peut être exécuté.

Le problème relatif à un seul fichier (l'indice f étant partout enlevé) consiste donc à minimiser (sur K) la fonction de coût suivante :

$$C(K) = Q(K) + U(K) + S(K)$$

Les auteurs se proposent d'étudier le **comportement de la fonction C (K)** lorsqu'on ajoute des copies du fichier dans le réseau. On peut tout d'abord remarquer que, dans ce cas, les fonctions U (K) et S (K) croissent alors que Q (K) ne croît certainement pas.

Désignons par L (m) (ou encore L_n (m), $n \in \mathbb{N}$) l'ensemble des indices des noeuds associés à une allocation arbitraire de m copies du fichier et notons F (K) = Q (K) + S (K); on peut trouver dans [LEVIN 74] la démonstration du résultat suivant :

Si l'ajout à L_1 (m - 1) d'une copie du fichier au noeud k ($k \notin L_1$ (m - 1)) est tel que $F[L_2(m)] \geqslant F[L_1(m-1)]$, **où** $L_2(m) = L_1(m-1) \cup \{k\}$, **alors** le minimum global de C [L (.)] ne peut être atteint par une allocation L (.) telle que L (.) $\supset L_2$ (m)

(ii) La notion d'hypercube

Ils utilisent, comme Casey (cfr [CASEY 72]), la technique de l'hypercube afin de réaliser une énumération sur un ensemble réduit de solutions possibles dans le but de trouver l'optimum. Cette technique est identique à l'algorithme optimal d'Alcouffe et Muratet (cfr [ALCOUFFE 76]), il s'agit d'une énumération implicite avec conditions pour cesser la recherche.

La notion d'hypercube (introduite en I.1.3.), aide à visualiser l'ensemble des allocations (de copies) du fichier possibles et à illustrer le résultat énoncé ci-dessus.

Ce **résultat s'interprète de la façon suivante : soient** deux sommets x et y de l'hypercube; **si** y est un suivant de x tel que $F(x) \leqslant F(y)$, **alors** le minimum global de C ne sera jamais atteint ni en y ni en aucun de ses descendants.

(iii) Introduction de contraintes supplémentaires

Il semble approprié d'inclure diverses classes de contraintes dans ce problème. Ces contraintes, qui garantissent certains niveaux de performance, restreindront la taille de l'hypercube et rendront ainsi la procédure de recherche plus efficace.

1. Contraintes politiques et de sécurité

Cela consiste à définir pour chaque fichier un ensemble de "noeuds permis" (i.e. où il peut être stocké). Si seulement k noeuds sont permis, il suffit simplement de considérer l'hypercube à k dimensions correspondant (la taille du problème est réduite de $2^N - 2^k$ combinaisons).

2. Contraintes de temps d'accès

Les auteurs supposent que les grandeurs t_{ij} = délai maximum sur la ligne joignant le noeud i au noeud j font partie des spécifications du réseau (par exemple, délai inférieur ou égal à 0,2 sec. sur les lignes 50^K bps d'ARPANET). On note T, le délai maximum permis par l'utilisateur pour l'accès au fichier. Une **route** (i,j,k), du noeud i vers une copie du fichier stockée en k via le programme p stocké au noeud j, est dite **faisable** si $t_{ij} + t_{jk} \leqslant T$. Une **allocation** du fichier est **infaisable** s'il existe au moins un noeud duquel il n'existe aucune route faisable vers aucune copie du fichier. Il suffit alors de supprimer les allocations infaisables des allocations possibles. Une allocation L (.) est un **"cover"** si pour tout $i = 1, 2, \ldots, N$, il existe k L (.) et j Q_p tels que $t_{ij} + t_{jk} \leqslant T$. Une allocation L (.) est un **"prime cover"** si L (.) est un "cover" tel que la suppression d'un noeud quelconque le réduit à une allocation qui n'est pas un "cover".

Il s'agit donc de trouver tous les "prime covers" et d'exclure de la recherche de l'allocation optimale tous les sous-ensembles de ceux-ci. Remarquons que cette approche de la notion de délai d'accès adopte le point de vue de l'utilisateur et préserve la décomposition du "problème multi-fichiers" en "problèmes uni-fichier" (à la différence de l'approche de [CHU 69] et [MAHMOUD 76]).

(iv) Procédure de solution

La procédure proposée est constituée essentiellement d'une procédure de recherche basée sur le schéma énumératif pour la programmation 0-1 présenté dans [ELLWEIN 74] (où l'auteur démontre que cette procédure est exhaustive et finie) combinée avec le résultat énoncé en 3.2. (ii) ci-dessus qui permet de couper la partie inférieure de l'hypercube alors que les contraintes de temps d'accès en coupent la partie supérieure. Le schéma d'Ellwein est une méthode d'énumération implicite qui réclame peu de quantité de stockage et permet simultanément une flexibilité de branchement pareille à celle de l'approche "branch-and-bound".
Les étapes suivantes devraient être exécutées :

étape 1 : exclure tous les noeuds qui violent les contraintes politiques et de sécurité; aller à l'étape 2;

étape 2 : déterminer s'il existe au moins une solution faisable (via l'allocation 12... N); si oui aller à l'étape 3, sinon stop;

étape 3 : formuler un "set covering problem"(1) et trouver tous les "prime covers" y associés; aller à l'étape 4;

---

(1) Il s'agit d'un problème de sélection (i.e. un ensemble d'objets discrets est donné duquel un sous-ensemble doit être choisi) pour lequel on ne dispose d'aucun algorithme convergent efficace (cfr [BALAS 76]). Chaque objet représente un ensemble (différent) de propriétés discrètes et réclame des coûts différents; il s'agit de trouver un sous-ensemble d'objets dont la somme des coûts est minimale et qui couvre chaque propriété au moins une fois.

étape 4 : invoquer une procédure de recherche basée sur le schéma énumératif d'Ellwein et sur le lemme énoncé ci-dessus; prendre les "prime covers" comme points de départ de la recherche (branchements vers le bas).

(v) Exemples et résultats

Les auteurs introduisent une dépendance programme-fichier dans l'exemple du réseau à 5 noeuds de [CASEY 72]; ils font remarquer que le coût de la solution optimale relative au modèle de Casey (indépendance programmes-fichiers) est 2,8 fois supérieur au coût de la solution optimale relative à leur propre modèle (introduction d'une dépendance en supposant que les programmes ne peuvent être exécutés en tout noeud, ce qui est le cas dans les réseaux hétérogènes réels).

Ils ont également testé l'exemple du réseau ARPA à 19 noeuds décrit dans [KLEINROCK 69] ; ils précisent avoir trouvé la solution optimale en 4,75 sec. sur un DEC 10. Remarquons que les auteurs testent leur technique de solution sur des données de Casey (cfr [CASEY 72]) alors que leur modèle est plus général. Fisher et Hochbaum (cfr [FISHER 80]) soulignent l'absence de tests à grande échelle et pensent qu'il est peu probable que le simple "branch-and-bound" développé par Morgan et Levin soit capable de traiter de tels exemples. Signalons, pour l'anecdote, que les programmes de ces derniers ne semblent pas dépourvus d'erreurs ! (cfr [FISHER 80] p. 733).

4) Etude du jeu d'hypothèses 6 ([LEVIN 75] et [LEVIN 78])

4.1. Introduction

On suppose maintenant que les taux de demandes d'accès (toujours parfaitement connus à l'avance) peuvent varier au cours du temps. Il est dès lors important que la structure physique de la BD s'adapte à ces changements.

Remarquons que cette capacité d'adaptation a déjà été étudiée dans le cadre des BDs centralisées (cfr références dans [LEVIN 82])

Les auteurs considèrent un planning de T périodes; ils supposent que les taux de demandes d'accès (aux fichiers) sont constants au cours d'une période et varient d'une période à l'autre (ce qui nécessitera des migrations de fichiers entre périodes); ils parlent alors de "problème dynamique à T périodes" et en proposent une formulation de programmation dynamique (en 4.1.) et une technique "branch-and-bound" ("b & b") de solution (en 4.2.), dont l'efficacité relative tient à certaines caractéristiques du problème qui permettent de réduire l'espace des états du "b & b". La procédure "b & b" utilisée est la même que celle suggérée en 3.2. (iv), les bornes générées dans le cas statique étant ici resserrées grâce à l'inclusion de coûts de réallocation entre périodes.

4.2. Modèle du jeu 6

En vertu des résultats énoncés en 3.2. (i) ci-dessus, nous savons que le problème d'allocation à multiples fichiers peut être décomposé en problèmes individuels. Nous présentons ci-dessous le modèle d'allocation dynamique d'un fichier unique proposé par les auteurs.

Supposons un planning de T périodes discrètes indexées par $t = 1, \ldots, T$.

Soient $y_{kt} = 1$ si une copie du fichier est stockée au noeud k pendant la période t (= 0 sinon);

λ_{ipt} = (resp. λ'_{ipt}) = le "query (resp. update) traffic" du noeud i via le programme p pendant la période t;

$K_t = \{ k : y_{kt} = 1 \}$, une allocation arbitraire du fichier pendant la période t;

$K_T = [K_1, K_2, \ldots, K_T]$, un arrangement arbitraire d'allocations du fichier pendant les périodes 1 à T;

$T_t\ (K_{t-1}, K_t)$ = les coûts de migration de K_{t-1} à K_t;

b = le nombre d'unités de message nécessaires à la transmission du fichier sur une ligne de communication;

k (t) = l'index des noeuds à la période t;

$C\ (K_t) = Q\ (K_t) + U\ (K_t) + S\ (K_t)$

(cfr 3.1. où l'indice f est remplacé par t et l'indice k par k (t)).

Nous supposons que la migration de fichiers est réalisée de la façon la plus économique, ainsi :

$$T_t\ (K_{t-1}\ K_t) = \sum_{k\ (t) \in K_t} L_m \cdot \min_{k\ (t-1) \in K_{t-1}} C_{k\ (t-1)k(t}$$

La fonction de coût multi-période est donnée par :

$$G\ (\mathbb{K}_T) = \sum_{t=1}^{T} \left[C\ (K_t) + T_t\ (K_{t-1}, K_t)\right],$$

en supposant, implicitement, que l'allocation actuelle K_o est donnée. Le problème consiste à trouver $\mathbb{K}_T^*$ tel que :

$$G\ (\mathbb{K}_T^*) = \min_{\mathbb{K}_T} G\ (\mathbb{K}_T).$$

4.3. Etude du problème et techniques de solution

Levin et Morgan se penchent tout d'abord sur le cas T = 1. Ils démontrent un résultat similaire à celui énoncé dans MORGAN 77 pour le cas statique (cfr 3.2. (ii)). La même procédure de recherche que celle qui est appliquée au cas statique (cfr 3.2. (iv)) peut être utilisée dans ce cas et s'avère être au moins aussi efficace (les bornes sont plus resserrées que dans le cas statique).

Ils étudient ensuite le cas général à T périodes. Ils proposent une formulation de programmation dynamique (PD). Ils rappelent (cfr [LODISH 70]) les limitations en temps de calcul de la PD : croissance linéaire avec le nombre d'étapes (les périodes) et exponentielle avec le nombre d'états (les allocations).

Ils adoptent dès lors une stratégie "b & b" et considèrent le problème comme un programme dynamique avec des limites spéciales utilisées afin de réduire l'espace des états. Morin et Marsten (cfr [MORIN 76]) ont déjà employé une telle stratégie. Ils explicitent avec précision l'équation fonctionnelle de la programmation dynamique pour coûts additifs ainsi que les divers problèmes (de mémorisations et de calculs) posés par la solution récursive de l'équation fonctionnelle. Ils montrent comment les méthodes "b & b" peuvent être utilisées pour réduire les exigences de stockage et de calcul des programmes dynamiques discrets. Levin et Morgan énoncent un résultat [illegible] d'une condition qui, lorsqu'elle est satisfaite par un état E_t (de l'étape t), permet d'éliminer de l'espace des états à l'étape t tous les états descendants de E_t.

Levin et Morgan proposent un algorithme, basé sur la formulation récursive (PD) du problème, qui part de t = T et se termine (en T étapes) avec t = 1. A chaque étape t, il s'agit de calculer B_t, un ensemble dont le complémentaire est l'ensemble de toutes les allocations du fichier exclues du processus de minimisation à l'étape t par le résultat démontré dans le cas statique (cfr 3.2. (ii)), si t = T, ou par R (via B_{t+1}), si $T < t \leqslant 1$.

Ils reprennent (uniquement !) l'exemple à 5 noeuds de Casey (cfr [CASEY 72]) et comparent les résultats obtenus à partir du modèle statique et du modèle dynamique (à l'avantage de ce dernier).

5) Etude des jeux d'hypothèses 7 et 8 ([LEVIN 82])

Dans les applications pratiques, il est rare que les taux de demandes d'accès soient parfaitement connus à l'avance; il est dès lors nécessaire d'estimer ces taux et d'incorporer ces estimations dans le design de la BDD. Levin détaille, dans [LEVIN 82], les diverses fonctions d'un DDBMS adaptatif en accord avec les directives proposées dans [HAMMER 77]. Levin adopte les **modèles d'optimisation** décrits ci-dessus (cfr 3 et 4); les taux de demandes d'accès futurs sont estimés par un **module estimateur** à partir de statistiques collectées par un **module récolteur d'informations.** Les taux λ_{ipt} et λ'_{ipt} (cfr 4 ci-dessus) sont des variables aléatoires; il en est de même de $G(K_t) = C(K_t) + T_t(K_{t-1}, K_t)$.

Levin propose l'approche suivante (pour le cas $\Gamma = I$): calculer des estimations de ces taux ($\lambda\hat{}_{ip}(t)$) via une procédure (cf. [LEVIN 82]), définissant le module estimateur, qui calcule $\lambda\hat{}_{ip}(0)$, ..., $\lambda\hat{}_{ip}(t+I) = \lambda\hat{}_{ip}(t) + e(t)$,... où $e(t)$ est un facteur d'ajustement, et trouver Π_t^* tel que $E[G(\Pi_t^*)] = \min_{\Pi_t}\{ E[G(\Pi_t)]\}$. Nous ne parlerons pas ici des propriétés statistiques de ces estimations.

I.4. TRAVAUX S'INSERANT DANS LE CADRE DE RECHERCHE DEVELOPPE PAR LEVIN ET MORGAN

Nous poursuivons l'étude du FAP, considéré indépendamment de tout autre problème de conception, en présentant divers travaux qui s'insèrent dans le cadre de recherche proposé par Levin et Morgan (cfr I.3. ci-avant).

1) **Fisher et Hochbaum** (cfr [FISHER 80]) se proposent de corriger un tant soit peu deux défauts majeurs (selon eux) des travaux antérieurs relatifs au FAP, à savoir :

(i) le peu d'attention porté au **développement d'algorithmes** et

(ii) l'insuffisance des **tests** ainsi que leur limitation à de très petits problèmes.

En fait, ces travaux se contentent essentiellement de **décrire des modèles** appropriés au problème (le plus souvent sous la forme de programmes linéaires entiers ou mixtes). Fisher et Hochbaum donnent comme exemple [MORGAN 77] (cfr jeu d'hypothèses 5 page 35) - dont ils reprennent plus ou moins le modèle - qui propose un algorithme "branch-and-bound" dont il semble peu probable qu'il puisse résoudre des problèmes à grande échelle (Levin et Morgan se contentent de le tester sur un réseau de 19 noeuds [1]). Ils proposent toute une stratégie de solution (heuristiques et techniques de programmation mathématique diverses intégrées à un "b & b", test permettant de réduire la taille du problème à priori, ...etc) ainsi qu'une impressionante batterie de tests (50 réseaux de 15 à 58 noeuds) qu'il nous a semblé intéressant de détailler.

---

(1) Une version d'ARPANET (cfr [KLEINROCK 69]). Les auteurs signalent que c'est le seul exemple non trivial testé qu'ils aient trouvé dans la littérature. Ils font remarquer également que la solution proposée par Levin et Morgan n'est pas optimale (ils supposent qu'il y a une erreur dans le programme de ces derniers!).

Leur modèle prend la forme d'un programme entier mixte. Ils décrivent deux heuristiques dont le but est d'engendrer des solutions faisables (1): une "greedy heuristic" et une "interchange heuristic". La **"greedy heuristic"** construit une solution en ajoutant des copies successives du fichier (là où cela produit la plus grande réduction de la fonction objective) jusqu'à ce qu'aucune copie additionnelle ne parvienne plus à améliorer la solution. L'**"interchange heuristic"** essaye d'améliorer une solution de départ par étapes successives dont chacune consiste à enlever une copie et à en ajouter une autre. Elle se termine lorsqu'aucun de ces "interchange" ne parvient plus à améliorer la solution (2). Ils utilisent la méthode dite de **"relaxation lagrangienne"** (cfr [GEOFFRION 74] p.ex.) afin d'obtenir des limites inférieures (3) au coût de la solution optimale (4). Ils décrivent deux méthodes alternatives dont le but est de fournir des solutions optimales ou sous-optimales au problème lagrangien dual de celui étudié précédemment: la **méthode du sous-gradient** et une variante de la **méthode "steepest ascent"**. Le principe général de ces méthodes est le suivant. Supposons que u et v soient les deux paramètres du problème lagrangien. Les méthodes commencent en (u^0,v^0) et génèrent une suite (u^k,v^k) (qui devrait converger vers la solution optimale) par la règle: $(u^{k+1},v^{k+1}) = (u^k,v^k) + f(d^k,e^k)$, où (d^k,e^k) est appelé vecteur direction à partir de (u^k,v^k). Les méthodes varient dans la façon de calculer ces vecteurs directions (compromis quantité de calculs-optimalité). Les auteurs suggèrent d'appliquer tout d'abord les heuristiques.

---

(1) Et donc des limites supérieures au coût de la solution optimale.

(2) En fait, cette heuristique n'a pas été utilisée, la qualité des solutions faisables générées par la "greedy heuristic" étant satisfaisante.

(3) Par ailleurs très simples à calculer.

(4) On peut aussi obtenir une solution faisable au problème.

Si l'on découvre une limite inférieure et une solution faisable ayant même valeur de coût, le problème est résolu; sinon la recherche peut être complétée par un **algorithme "branch-and-bound"** utilisant ces heuristiques comme procédures (1). Les auteurs démontrent une condition (dont la vérification ne nécessite guère de calculs!) permettant de réduire la taille du problème à priori (i.e. avant toute exécution du "b & b").

Les auteurs ont testé diverses méthodes de génération de limites inférieures (2) ainsi que l'algorithme global de solution, sur 58 problèmes de 15 à 58 noeuds. La performance générale de ce dernier semble très satisfaisante (pour des exemples de cet ordre de grandeur). Les auteurs soulignent les excellents résultats de la"greedy heuristic" (très rapide et très souvent optimale (3)). Ils proposent enfin diverses courbes décrivant l'effet, sur la valeur optimale, du nombre de copies du fichier, du pourcentage d'"updates" et du nombre de programmes de mise à jour (diverses exécutions sur des variations paramétriques de l'ARPA 19 noeuds).

2) **Hatzopoulos et Kollias** (cfr [KOLLIAS 80]) proposent essentiellement une extension des résultats de [GRAPA 77] et de [KOLLIAS 81] (cfr (ii) et (iii) pp.22 et 23) au **SFAP** (cfr p.9) **avec utilisation dynamique** (cfr jeu d'hypothèses 6 p. 35). Rappelons que le même problème a déjà été étudié dans [LEVIN 78] où les auteurs développent un algorithme qui limite la croissance de l'espace des états en utilisant des caractéristiques spéciales du problème. Hatzopoulos et Kollias présentent un modèle qui est une extension du modèle de Casey (cfr p. 10)(4).

---

(1) Les auteurs ont testé diverses variations (sur les heuristiques employées) de l'algorithme "b & b" de base.

(2) Compromis entre la qualité de ces limites, le temps CPU et l'espace de stokage requis

(3) Remarquons cependant qu'on ne dispose pas, à son sujet, de limite sur l'erreur produite.

(4) Ce modèle est caractérisé par $(2^n-1)^p$ solutions possibles (où n = nombre de noeuds du réseau et p = nombre de périodes considérées).

Ils démontrent quatre **théorèmes**, relatifs à ce modèle, qui permettent de réduire l'espace des solutions et fournissent de bonnes limites utiles à toute méthode de recherche algorithmique sur celui-ci (cfr p.ex. [LEVIN 78], p. 44). Le premier (resp. troisième) fournit une condition nécessaire d'allocation (resp. de non allocation) d'une copie de fichier en tel noeud à telle période. Le deuxième théorème permet de réduire l'ensemble des noeuds qui peuvent être servis, à une période donnée, par un noeud donné. Ces trois théorèmes peuvent être utilisés à chaque étape de toute procédure résolvant le problème considéré. Le quatrième théorème ne peut être utilisé qu'à priori (i.e. initialement à la procédure de solution) et permet d'interdire à tel noeud de stocker une copie du fichier pendant telle période (généralisation du troisième théorème de [GRAPA 77]). En se basant sur les résultats de l'exemple proposé (1), les auteurs suggèrent de mettre à profit l'existence de types d'utilisation non uniformes afin de sélectionner de meilleures politiques d'allocation de fichier.

3) **Kollias et Hatzopoulos** (cfr [KOLLIAS 81b]) étudient le problème de l'**allocation simultanée de s fichiers distincts** (p.ex. une partition de BD en s éléments) sur un réseau informatique, en imposant à tout noeud de stocker au plus k fichiers (où k est un paramètre contrôlé par le concepteur, $k \leq s$). L'application de s STAP séparés eut conduit à un réseau mal équilibré (noeuds surchargés, contraintes de place de stockage violées...etc.). Les auteurs distinguent le cas redondant du cas non redondant (i.e. pas de copies multiples d'un même fichier) et proposent des modèles semblables à celui, pour ces deux cas, de [CASEY 72] (cfr p.10). Le modèle du cas non redondant peut être seulement résolu jusqu'à 30 fichiers en appliquant, par exemple, l'algorithme développé par [BYRNE 68] qui résoud les programmes linéaires 0-1 par énumération implicite. Ils proposent un **théorème** (généralisation d'un théorème démontré par Casey) fournissant

---

(1) Un unique exemple de 5 noeuds seulement!

une condition nécessaire de non redondance de l'allocation optimale. Ils décrivent une **heuristique** (dont le nombre maximum d'itérations est égal au nombre de noeuds du réseau) pour résoudre le cas redondant. Elle est basée sur un théorème donnant une condition nécessaire pour ne pas allouer une copie d'un tel fichier à tel noeud, et sur la résolution du problème où on a relâché la contrainte sur le nombre maximum (k) de fichiers que l'on peut stocker en un noeud. Ils se contentent du seul exemple (légèrement modifié) à 5 noeuds de Casey (cfr [CASEY 72]) pour illustrer leur heuristique. Ils soulignent que la redondance permet d'améliorer le temps de réponse et la fiabilité, et de diminuer les coûts de communication. Ils suggèrent de faire varier k et même s (déterminer ce que devrait être la partition d'une BD donnée).

4) **Ramamoorthy et Wah** (cfr [RAMAMOORTHY 79[a]]) décrivent une heuristique, relative au FAP dynamique (jeu d'hypothèses 6 p.35), qui utilise certains des principes des algorithmes "add-drop" (cfr p.29). Les résultats des tests proposés semblent peu concluants. Les auteurs signalent que des algorithmes polynomiaux peuvent exister dans certains cas spéciaux du FAP (cfr [STONE 77] p.ex.) dont l'utilité pratique semble, par ailleurs, très limitée.

Ils présentent brièvement trois classes d'heuristiques habituellement utilisées :

(i) "hierarchical designs";
(ii) "clustering algorithms";
(iii) "add-drop algorithms".

Les heuristiques sont généralement interactives : une solution faisable est générée, que l'on décide ou non d'améliorer via un algorithme de décision (qui précise comment l'améliorer). Elles présentent les désavantages suivants :

(i) elles fournissent habituellement un optimum local;
(ii) leurs comportements "dans les plus mauvais cas" ("worst case behavior") sont très difficiles à déterminer.

Les hypothèses suivantes sont utilisées dans le développement du modèle retenu (programme entier linéarisé) :

(H_1) les accès aux fichiers sont indépendants (optimalisation pour chaque fichier indépendamment)

(H_2) toutes les contraintes sur le système peuvent être représentées sous la forme de coûts;

(H_3) le comportement des accès à un fichier pour une période est estimé au début de cette période et, à cet instant, on ne peut estimer les comportements pour les périodes suivantes (optimalisation pour chaque période indépendamment).

Le principe de l'algorithme de solution est le suivant. Notons K_0 (resp. K_1, K_2), l'ensemble des noeuds sans copie (resp. avec copie, non assignés). Initialement, $K_0 = \emptyset$, $K_1 = \emptyset$ et $K_2 = \{1,2,\ldots,n\}$ (où n = nombre de noeuds du réseau). L'algorithme procède en n étapes : à chaque étape, un indice est enlevé de K_2 et ajouté à K_0 ou à K_1. A une étape donnée, le choix de cet indice est réalisé de la façon suivante. A chaque élément i de K_2, on associe deux valeurs :

v_i = solution du programme entier sans contrainte d'intégralité sur le problème relatif à K_0, $K_1 \cup \{i\}$, $K \setminus \{i\}$.

$\bar{v}_i$ = idem (problème $K_0 \cup \{i\}$, K_1, $K_2 \setminus \{i\}$)

L'indice i_0 retenu est celui de valeur minimale (il est ajouté à K_0 si la valeur minimale est $\bar{v}_{i_0}$, et à K_1 s'il s'agit de v_{i_0}).

La complexité de la solution du programme entier sans contrainte d'intégralité est $O(n^2)$ (cfr [EFROYMSON 66]); la complexité de l'algorithme global est donc $O(n^4)$. Les auteurs présentent peu de résultats. Ils proposent des solutions (aux exemples à 5 et 19 noeuds de Casey, cfr [CASEY 72]) dont une présente une déviation de plus de 42 % par rapport à l'optimum.

5) **Akoka** (cfr [AKOKA 80]) développe un modèle qui est essentiellement celui de [MORGAN 77] (cfr I.3.). Les différences principales entre son approche et celle de Levin et Morgan sont les suivantes:

(i) il prend en compte le flux d'information en retour, c'est-à-dire le flux d'information qu'un utilisateur reçoit comme

réponse à ses "queries"[1];

(ii) il considère une dépendance, entre programmes et BDs, plus forte que la dépendance de Morgan et Levin : il suppose, en effet, que n'importe quel programme ne peut travailler sur n'importe quelle BD;

(iii) il met en évidence et corrige une incohérence du modèle de Levin et Morgan (relative à une contrainte de routage).

La procédure de solution est un "b&b" modifié (appelé "bounded b&b", cfr [ABADIE 69^a]) qui résoud un ensemble de programmes (en nombre moindre qu'un "b&b" classique!) non linéaires continus (relaxation de contraintes d'intégralité) par la méthode du gradient réduit généralisé (cfr [ABADIE 69^b]). Cette procédure est plus souple que celle de Levin et Morgan (leur technique de décomposition interdisant l'introduction de certaines contraintes additionnelles).

Les tests de performance (sur l'exemple à cinq noeuds de Casey !) proposés mettent en évidence les influences non négligeables (sur l'optimum) de l'introduction du "flux d'information en retour" et des coûts de stockage (importants pour les BDs réelles!). L'auteur estime qu'un problème de 600 variables ne devrait pas exiger plus d'une minute de temps CPU. Remarquons qu'un problème à 5 noeuds, 4 fichiers et 3 programmes, réclame déjà plus de 600 variables. Comme toute technique optimale relative au FAP, leur "b&b" n'est applicable qu'à des problèmes de taille réduite.

---

(1) Via l'estimation de quotients δ = taille réponse/taille "query". L'auteur montre, sur un exemple, l'influence importante de cette composante des flux d'information.

I.5. LE PROBLEME DE LA CONCEPTION D'UN RESEAU INFORMATIQUE

1) Introduction

Nous nous proposons d'étudier à présent le **problème de la conception d'un réseau informatique** (noté CCNDP [1]) indépendamment de tout autre problème de design [2][3].

Grossièrement, on pourrait diviser le CCNDP en deux **sous-problèmes :**

(i) le **problème de la conception de la topologie du réseau** (i.e. déterminer le graphe des lignes de communication) que l'on notera NTDP [4];

et (ii) le **problème de l'allocation des capacités** [5] **aux lignes de communication** (noté CCAP [6]).

Remarquons que le NTDP et le CCAP sont en fait intimement liés et qu'ils doivent être considérés simultanément avec d'autres problèmes tel celui, par exemple, de la détermination de la procédure de routage.

Il existe en fait diverses formulations du CCNDP; elles correspondent à différents choix de mesures de performance, de variables de conception et de contraintes. Tout comme nous l'avons fait dans le

---

(1) "Computer Communication Network Design Problem".
(2) En particulier, indépendamment du problème de l'allocation de fichiers (FAP).
(3) Nous introduisons, ci-dessous, de façon intuitive, divers concepts qui seront en fait abondamment explicités et précisés par la suite.
(4) "Network Topology Design Problem".
(5) Par exemple, 2400 bps (bits par seconde) ou 4800 bps...etc.
(6) "Channel Capacities Allocation Problem".

cas du FAP (cfr p.1), nous proposons ci-dessous **une formulation** (parmi d'autres) **du CCNDP** (sans préciser pour l'instant les divers concepts y apparaissant) afin de concrétiser d'emblée le discours (cfr [KLEINROCK 80] et [TANENBAUM 81]) :

étant donné l'emplacement des noeuds (1) du réseau, les options et coûts des capacités des lignes ainsi que le trafic (2)

minimiser le coût total de communication (3)

sur la topologie, les capacités des lignes et le routage (4) et

sous des contraintes de délai (5), de fiabilité (6) et de trafic (i.e. il faut satisfaire le trafic)

Nous aimerions, dès à présent, souligner l'extrême **complexité du CCNDP** dont les **raisons majeures** (7) sont les suivantes :

(i) le nombre de topologies possibles croît exponentiellement avec le nombre de noeuds du réseau : dans le cas d'un réseau de N noeuds, il y a N(N-1)/2 lignes potentielles et donc $2^{N(N-1)/2}$ topologies à prendre en considération! (8) [FRANK 71a] fait remarquer que le réseau

---

(1) Contentons-nous, pour l'instant, de ce terme général (il pourrait s'agir de terminaux, d'ordinateurs, de concentrateurs, de commutateurs...
(2) Sous la forme, par exemple, du nombre moyen de messages qui doivent être envoyés, par seconde, de tel noeud à tel noeud.
(3) Essentiellement le coût de l'allocation des capacités aux lignes de communication. Ce coût peut être une fonction arbitraire de divers paramètres tels la longueur de la ligne, la capacité allouée, le taux d'erreur, les modems...etc.
(4) Variables de design.
(5) Exiger, par exemple, que le délai moyen de transmission d'un message soit inférieur à une valeur donnée (cfr 2)).
(6) Notion liée essentiellement à la connectivité du réseau (cfr 1)).
(7) Nous y reviendrons.
(8) Pour un petit réseau de 10 noeuds, il y a 3.10^{13} topologies possibles; il faudrait mille ans pour les examiner toutes à raison d'une ms par topologie (ce qui est optimiste!) (cfr [TANENBAUM 81]).

optimal (global) est habituellement impossible à trouver même si on se limite à des topologies de type arbre).

(ii) en pratique, les capacités des lignes de communication sont choisies parmi un ensemble fini d'options disponibles; en conséquence, on se trouve face à un problème de conception discret que les méthodes existantes de programmation entière ne peuvent généralement résoudre, si ce n'est dans le cas de tailles peu réalistes;

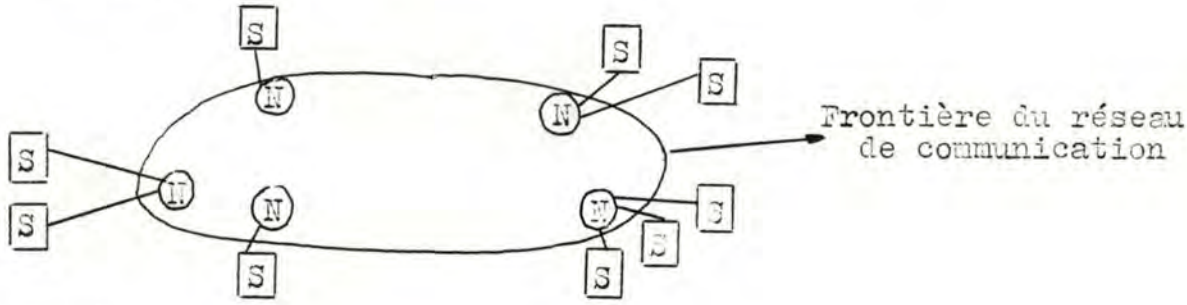
(iii) les contraintes doivent être satisfaites;

(iv) les expressions du délai sont non linéaires (cfr [KLEINROCK 72] et [KLEINROCK 76]);

(v) les mesures de fiabilité sont souvent non linéaires (cfr [FRANK 72[a]])

Chandy et Sickle (cfr [SICKLE 77]) ont démontré que divers problèmes de design (extrêmement simplifiés) relatifs aux réseaux sont NP-complets (cfr p.13). Ils préconisent dès lors le développement d'algorithmes de solution heuristiques tout en soulignant l'importance (économique) des techniques optimales dans le cas de réseaux de taille modérée (cfr pp.13 et 14)

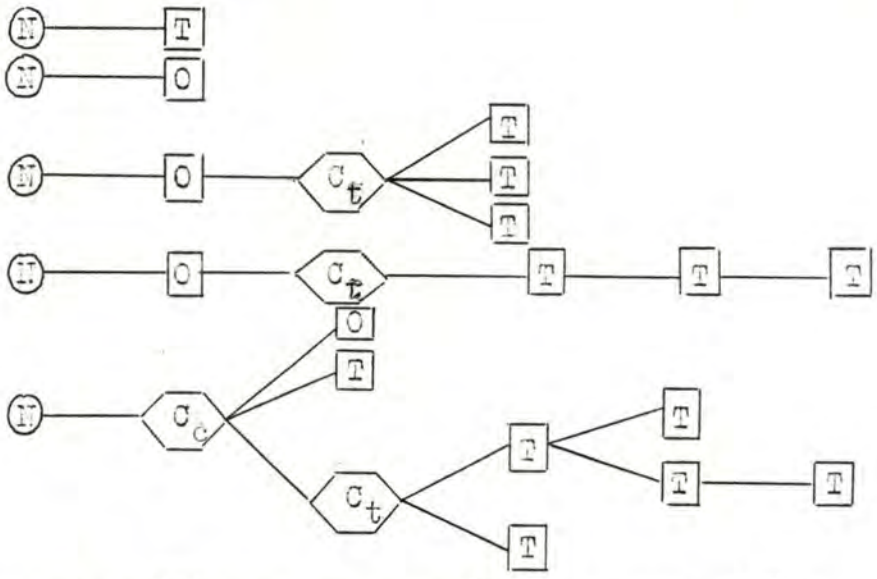
La structure d'un **réseau informatique** peut être schématisée de la façon suivante (cfr [KLEINROCK 76], [WONG 78[b]], [CORNAFION 81] et [TANENBAUM 81]) :



(N) = noeud
[S] = station

_ . Réseau informatique . _

On y distingue l'aspect "communication" (réseau de communication) de l'aspect "application" (les stations). Un **réseau de communication** est constitué d'un ensemble de **noeuds** (1) (généralement des ordinateurs spécialisés) reliés entre eux par des voies simples (i.e. ne comportant pas d'organe de mémorisation) encore appelées **lignes**; chaque noeud peut communiquer avec n'importe quel autre, soit directement, soit par l'intermédiaire d'autres noeuds. Le réseau de communication sert à relier entre eux des systèmes informatiques (ordinateurs, terminaux, systèmes spécialisés,...etc) appelés **stations**; chaque station est connectée à un (ou occasionnellement plusieurs) noeuds; un utilisateur ou un programme d'application attaché à une station est dit **abonné** à cette station; les abonnés se transmettent de l'information codée sous une forme convenue (on parle alors de **message** (2)), le travail du réseau de communication consistant à transporter ces messages entre stations (de façon fiable et rapide). La **nature d'une station** peut être diverse, par exemple:



où (N) = noeud, [O] = ordinateur, [T] = terminal, $\langle C_t \rangle$ = contrôleur et $\langle C_c \rangle$ = concentrateur.

_ . Stations . _

---

(1) (Appelés aussi commutateurs) qui assurent des fonctions de routage, de contrôle de flux et d'erreur, de file d'attente, d'insertion et de suppression de messages...etc.

(2) Commandes, demandes de renseignements, transmissions de fichiers,...

Le choix du chemin suivi par un message est généralement réglé par une procédure de routage. Une **procédure de routage** des messages est une règle de décision qui détermine le noeud suivant que le message visitera sur son chemin au travers du réseau de communication. **L'algorithme de routage** opère la mise en oeuvre de la procédure de routage. Les paramètres impliqués dans cet algorithme peuvent être, par exemple : l'origine et la destination du message, la disponibilité de certaines lignes ou de certains noeuds ...etc. En fait, les politiques de routage sont adaptatives ou déterministes selon qu'elles tiennent compte ou non des caractéristiques dynamiques du réseau de communication.(1). La méthode de **routage fixe** est un exemple de politique déterministe. Les algorithmes de routage fixe spécifient statiquement un chemin unique entre tout couple (noeud source, noeud destination) (chaque noeud dispose d'une table de routage qui, pour chaque destination, indique le prochain noeud à atteindre). On peut rendre le routage fixe plus fiable en définissant un ou plusieurs chemins de secours utilisés en cas de défaillance du chemin normal. Quand plus d'un chemin est associé à un couple (noeud source, noeud destination), on parle de **routage alterné** : le routage alterné peut être déterministe ou aléatoire (on parle alors de **routage aléatoire**) si tel est le cas du choix du chemin.

A ce modèle de réseau informatique correspond une **stratégie hiérarchique de design** (2) qui consiste à distinguer les deux **sous-problèmes** suivants (cfr p.ex. [ICCC 78] et [KAMENBAUM 81])

(i) "the **backbone network design** problem" c'est-à-dire le problème du design du réseau de communication,

---

(1) observations sur les flux de trafic, les défaillances de noeuds, de lignes ...etc.

(2) Qui permet de réduire quelque peu la complexité du CCNDP.

et (ii) "the **local access network design** problem", c'est-à-dire le problème du design d'une station (ou encore, le problème du design des réseaux centralisés (1)).

Remarquons d'emblée que la topologie d'une station est un arbre, ce qui permet diverses simplifications (dans un arbre il n'y a, par exemple, qu'un seul chemin entre deux noeuds donnés, ce qui élimine déjà le problème du routage (2)).

Remarquons également que le NTDP et le CCAP se retrouvent dans l'un et dans l'autre de ces deux sous-problèmes (3). Il faut souligner que le "backbone network design" et le"local access network design" (ou encore "centralized network design") ne sont que des aspects du problème général de la conception des réseaux informatiques (cfr p.ex. [ICCC 78] ou encore [KLEINROCK 76] (4)).

Notre but dans ce paragraphe, comme d'ailleurs dans l'ensemble de ce chapitre, est d'indiquer quelques pistes à suivre, tout en donnant une première idée sur la nature des divers problèmes traités, sur leur complexité et sur quelques techniques de solution développées à leur sujet. Aussi nous contenterons-nous d'étudier sommairement

---

(1) Les **réseaux centralisés** (appelés aussi "multi-drop networks") consistent en un ensemble de sites éloignés connectés à un site central par lequel est "routé" le trafic échangé entre deux sites quelconques. La plupart des systèmes "time-sharing", de réservation...etc sont de ce type (cfr [FRANK 71]).

(2) Il faut souligner cependant qu'un arbre peut ne pas être la topologie la plus économique, même dans le cas d'un réseau centralisé (cfr contre-exemple dans [FRANK 72a]).

(3) Avec d'autres problèmes, comme par exemple

(i) la détermination de la procédure de routage, dans le cas du "backbone design",

et (ii) le problème de remplacement des concentrateurs (par ailleurs NP-complet , cfr [SICKLE 77]), dans le cas du"local access design".

(4) Problèmes de contrôle de flux, de protocoles, de routage...etc.

les notions d'analyse de fiabilité (en 2)) et de délai (en 3)) ainsi que les modèles, techniques de solution et résultats d'expériences relatifs au problème de la conception des réseaux centralisés (en 4)) et des "backbone networks" (en 5)).

Nous tenions à détailler quelque peu la problématique du CMDP afin de présenter dans son contexte le MTDP, un aspect que l'on ne retrouve pas dans le travail de Mahmoud et Riordon, travail que nous avons choisi d'étudier (plus en détail) et d'implémenter (cfr chapitre II

2) Analyse de la fiabilité des réseaux

Nous présentons, dans ce paragraphe, quelques résultats relatifs à l'**analyse** de la fiabilité des réseaux, l'aspect design du problème[1] étant, pour l'heure, ignoré

Nous proposons (en (i)) une brève introduction intuitive à la notion de fiabilité; nous décrivons ensuite (en (ii)) un cadre général pour le problème de l'analyse de fiabilité, ainsi que trois travaux spécifiques relatifs à ce problème (en (iii)); nous soulignons enfin (en (iv)) l'extrême complexité du problème et signalons (en (v)) l'utilité de la simulation face à cette complexité.

(i) Introduction intuitive

Intuitivement, on pourrait définir la fiabilité d'un réseau comme étant sa capacité de permettre à toute paire de noeuds[2] (en état de fonctionnement) de communiquer en dépit des éventuelles pannes de lignes et de noeuds qui pourraient survenir et dégrader, plus ou

---

(1) Le problème de design consiste à identifier les emplacements de lignes tels que la contrainte de fiabilité (imposée par le concepteur) soit satisfaite. Nous parlerons de cet aspect en I.5.4), I.5.5) et I.6.
(2) Nous considérons ici qu'un réseau est constitué de "noeuds" reliés par des"lignes".

moins fortement, la performance du système (flux, délai,...etc) (1). L'idée générale de la fiabilité est qu'un réseau fiable devrait rester connexe même si certains noeuds ou lignes tombent en panne (2).

Par **exemple**, afin d'atteindre un niveau raisonnable de fiabilité, la topologie d'**ARPANET** est conçue de telle façon qu'il existe au moins deux chemins indépendants (i.e. physiquement séparés) entre chaque paire de noeuds (critère de fiabilité); ainsi, deux noeuds et/ou lignes doivent tomber en panne avant que le réseau ne soit disconnecté (i.e. avant que deux noeuds ne puissent plus communiquer entre eux).

(ii) Cadre général pour le problème de l'analyse de fiabilité

(Référence principale : [BALL 80])

A un réseau correspond un **graphe** constitué d'**arcs** et de **noeuds** (3) (abstraction évidente), ce graphe (que nous continuerons à appeler réseau, par abus de langage) pouvant être **orienté** ou non en fonction du mode d'utilisation des lignes de communication du réseau qu'il représente (4).

---

(1) Remarquons d'emblée que le principal problème de fiabilité dans le domaine des bases de données distribuées (BDD) est de maintenir la cohérence ("consistency") de la BDD en dépit des pannes de noeuds et de lignes.

(2) On peut définir la fiabilité d'une ligne (ou d'un noeud) comme étant sa probabilité de bon fonctionnement.

(3) On parle de "composants" du réseau, pour désigner, indistinctement, les noeuds et les arcs.

(4) On distingue les modes "simplex" (la ligne ne peut transmettre de l'information que dans un seul sens), "half-duplex" (elle peut transmettre dans les deux sens alternativement) et "full-duplex" (elle peut transmettre simultanément dans les deux sens). Une ligne "simplex" est modélisée par un arc orienté, tandis qu'une ligne "half-duplex" ou "full-duplex" est modélisée par un arc non orienté.

L'analyse de fiabilité étudie plus particulièrement les **réseaux stochastiques,** réseaux dont les caractéristiques sont les suivantes :

(i) chaque arc et chaque noeud peut être dans un des deux états de fonctionnement ou de panne;

(ii) les états des composants du réseau sont des événements aléatoires indépendants.

Un réseau stochastique peut être représenté par un couple (T,Γ) où T est l'ensemble (non vide) de ses composants (arcs et noeuds) et $\Gamma = \{ F \subset T$: si tout élément de F tombe en panne et tout élément de T-F fonctionne, alors le réseau fonctionne $\}$. On note $pr[\Gamma, \{p_i\}]$, la probabilité que le système (T, Γ) fonctionne, où p_i est la probabilité de panne du composant i, pour tout $i \in T$.

Le **problème de l'analyse de la fiabilité d'un réseau** (stochastique) est le suivant (cfr [BALL 80]) :

étant donné les probabilités de panne des noeuds et des arcs du réseau,

calculer une mesure de fiabilité du réseau.

Il existe de nombreux **algorithmes d'analyse de fiabilité** (i.e. qui calculent des mesures de fiabilité de réseaux); ils se distinguent essentiellement par

(i) la mesure de fiabilité calculée (cfr la liste établie ci-après) et (ii) la forme de leurs "outputs" (cfr les classes de problèmes décrites ci-dessous).

[BALL 80] décrit trois **classes de problèmes d'analyse de fiabilité** (1) :

---

(1) La plupart des algorithmes d'analyse de fiabilité peuvent calculer des mesures sous chacune de ces trois formes.

(C_1) Inputs: $(T,\Gamma), \varepsilon, \beta, \{p_i\}_{i \in T}$, où $0 < \varepsilon < 1 < \beta$ et $0 \leq p_i \leq 1$, $\forall i \in T$

Output: r, où $pr[\gamma, \{p_i\}] \; \varepsilon < r < pr\,[\gamma, \{p_i\}]\, \beta$;

(C_2) Inputs: (T,Γ) et $\{(d_i,e_i)\}_{i \in T}$ où $d_i, e_i \in \mathbb{N}$, $e_i \neq 0$ et $d_i \leq e_i$, $\forall i \in T$,

Outputs: d et e, où $d,e \in \mathbb{N}$, $e \neq 0$, $d \leq e$ et $pr\,[\gamma, \{d_i/e_i\}] = d/e$;

(C_3) Input: (T,Γ)

Output: $s_j \in \mathbb{N}$ $(j = 0,1,\ldots,|T|)$ où $pr[\gamma,p] = \sum_{j=0}^{|T|} s_j (1-p)^j p^{|T|-j}$

(C_1) est peut-être la formule la plus naturelle lorsque les p_i sont estimés statistiquement. (C_2) a l'avantage de fournir une réponse précise. (C_3) est moins général ($p_i = p$, $\forall i \in T$), mais permet de développer une analyse de sensibilité très puissante (ayant les s_j, faire varier p).

Il existe de très nombreuses mesures de fiabilité (cfr $pr[\gamma, \{p_i\}]$ ci-dessus); nous donnons, ci-dessous, une liste (cfr [BALL 80] et [FRANK 72^b]) de celles qui sont le plus fréquemment rencontrées en pratique. Notons $[i \sim j]$ (resp. $[i \to j]$) l'événement aléatoire qui consiste en l'existence d'un chemin en état de fonctionnement entre les noeuds i et j (i et $j \in N$, l'ensemble des noeuds), le réseau étant non orienté (resp orienté). On peut dès lors définir les **mesures de fiabilité** suivantes :

$\begin{cases} (M_1) \\ (M_1') \end{cases}$ $pr\,[s \overset{\sim}{\underset{\to}{}} t]$ où $(s,t) \in N^2$ et $s \neq t$

$\begin{cases} (M_2) \\ (M_2') \end{cases}$ $pr\,[i \overset{\sim}{\underset{\to}{}} j \;,\; \forall (i,j) \in N^2 \;\text{ t.q. }\; i \neq j]$

$\begin{cases} (M_3) \\ (M_3') \end{cases}$ $pr\,[i \overset{\sim}{\underset{\to}{}} j, \; \forall (i,j) \in N^2$ t.q. $i \neq j$ et i,j en fonctionnement$]$

(M'_4) pr $[s \to i, \forall i \in M$ t.q. $i \neq s]$ où s $\in$ M

(M'_5) pr $[s \to i, \forall i \in M$ t.q. $i \neq s$ et i en fonctionnement$]$

(iii) Exemples de travaux relatifs au problème de l'analyse de fiabilité

(cfr appendice 1)

(iv) Complexité du problème de l'analyse de fiabilité

Les résultats énoncés en (iii) ci-avant mettent déjà en évidence l'extrême complexité du problème de l'analyse de fiabilité.

Plus rigoureusement, [BALL 80] a démontré que les problèmes suivants (cfr notations en (ii) ci-dessus) sont NP-complets (cfr p.12) :

(i) dans les réseaux avec pannes des noeuds **et** d'arcs :
(C1) avec (M2), (M'2), (M'3) et (M'5), et (C2) avec (M1), (M3), (M'1), (M'2), (M'3) et (M'5);

(ii) dans les réseaux avec pannes de noeuds uniquement :
(C3) avec (M1) et (M'1);

(iii) dans les réseaux avec pannes d'arcs uniquement :
(C3) avec (M1), (M'1), (M'2) et (M'3).

Remarquons cependant qu'il existe des algorithmes pour des problèmes d'analyse de fiabilité relatifs à des réseaux structurés dont le temps d'exécution est polynomial; c'est le cas, par exemple, des réseaux en arbre à propos desquels ces calculs peuvent être réalisés analytiquement (cfr [KERSHENBAUM 73]).

(v) analyse de fiabilité et simulation

[FRANK 72^a] résume un certain nombre d'approches relatives au problème de l'analyse de fiabilité d'un réseau.

Les auteurs pensent que les approches purement analytiques, dans le cas de grands réseaux distribués, sont impraticables (au niveau des calculs) et suggèrent d'utiliser, dans de tels cas, une combinaison d'analyse et de simulation afin d'obtenir des estimations de paramètres tels la probabilité de disconnection du réseau ou encore le pourcentage moyen de paires de noeuds qui ne peuvent communiquer. Ils décrivent trois approches de solution au problème suivant (cfr références dans [FRANK 72^a]) :

étant donné un réseau de n noeuds et m lignes,

et en supposant que la probabilité de panne de chaque composant du réseau soit égale à p,

estimer h(p), la probabilité que le réseau soit disconnecté et n(p), le nombre moyen de paires de noeuds incapables de communiquer, pour P valeurs de p.

L'idée de la première approche (simulation Monte-Carlo) est la suivante (cfr également [TANENBAUM 81] où un programme de simulation est proposé) :

(i) génération d'un nombre aléatoire pour chaque composant;

(ii) si le nombre est inférieur à p, le composant correspondant est enlevé du réseau;

(iii) on évalue la connectivité ou le nombre de paires de noeuds disconnectées du réseau résultant

(iv) ce calcul est répété de façon à obtenir une estimation suffisamment précise (de h(p) ou n(p));

(v) la procédure entière est répétée pour les P valeurs de p concernées.

[TANENBAUM 81] fait remarquer également que dans le cas de grands réseaux [1] irréguliers le seul recours est la **simulation.** L'auteur propose un programme qui calcule le pourcentage de noeuds incapables de communiquer, pour diverses valeurs de p (la probabilité de panne d'une ligne quelconque), ainsi que la probabilité que le réseau soit disconnecté. Il explique comment la simulation peut aider l'opérateur d'un réseau existant qui désire ajouter de nouvelles lignes afin de réduire la probabilité de disconnection du réseau tout en satisfaisant au mieux au trafic (inutile d'ajouter des lignes inutilisées).

3) Analyse de délai

L'analyse de délai que nous présentons dans ce paragraphe concerne les réseaux à commutation de messages (RCM) [2].

Reprenons la terminologie introduite en I.5.1, lors de la description de la structure de base d'un réseau informatique, et supposons connue la technique de commutation de données (messages ou paquets) (cfr par exemple [CORNAFION 81]).

---

(1) Remarquons que certaines études semblent indiquer (cfr [FRANK 72a]) que la fiabilité est peut-être la contrainte de design dominante dans le cas des grands réseaux.

(2) Puisque les paquets peuvent être traités comme des petits messages, les résultats de cette analyse s'appliqueront également aux réseaux à commutation par paquets.

Les deux mesures de performance suivantes sont très importantes dans le cas d'un RCM [1] :

(i) le délai de bout-en-bout,
et (ii) le "throughput du réseau".

Le délai de bout-en-bout (d'un message), que nous appellerons plus simplement **"délai (d'un message)"**, est le temps qui s'est écoulé depuis l'arrivée du message à son noeud source (auquel est attachée la station émettrice du message) jusqu'à ce qu'il atteigne avec succès son noeud destination (auquel est attachée la station réceptrice du message). **Le throughput du réseau** est le nombre moyen de messages fournis par le réseau (de communication) par unité de temps. Lors du design du réseau de communication, on voudrait que le délai soit suffisamment petit afin qu'il ne devienne pas une composante significative du temps de réponse des utilisateurs éloignés. [2]

Kleinrock a développé un modèle de réseau ouvert de files d'attente pour les RCM (cfr (i) ci-après) et en a déduit une expression du délai moyen (pris sur tous les messages délivrés par le réseau) [3] (cfr [KLEINROCK 72]). Il a ensuite utilisé cette expression de façon intensive pour l'analyse de performance et le design des réseaux [4] (cfr [KLEINROCK 76]).

---

(1) Les taux d'utilisation des lignes sont également très importants, car ils fournissent des informations à propos de goulots d'étranglement potentiels du réseau.

(2) Par exemple, un objectif de design d'ARPANET est que le délai moyen soit inférieur ou égal à 0,2 s. (cfr [FRANK 70], par exemple).

(3) Sans distinction sur la **classe** du message, c'est-à-dire le couple (noeud source, noeud destination).

(4) "Backbone design" (cfr 5) ci-après.

Nous présentons (en ii) ci-dessous) quelques éléments de l'analyse de délai qu'il a déduite [1] de ce modèle. [2]

(i) Le modèle de Kleinrock [3]

(cfr appendice 2)

(ii) Analyse de délai de Kleinrock (cfr [KLEINROCK 76])

Il existe diverses **composantes dans le délai d'un message** (cfr définition ci-dessus).

(i) le temps de service, ou encore, temps de transfert des données [4];

(ii) le temps d'attente (phénomène de file d'attente);

(iii) le temps de traitement aux noeuds (supposé constant et noté K) [5];

---

(1) Nous proposons également certains résultats déduits par Wong (cfr [WONG 78^a]).

(2) Cette analyse sera reprise par [MAHMOUD 76]!

(3) Ce modèle est en fait un cas particulier du modèle général de réseau de files d'attente étudié par [BASKETT 75].

(4) Longueur du message/capacité de la ligne.

(5) Chaque fois qu'un message entre dans un noeud, des instructions sont exécutées, afin de réaliser les fonctions de commutation ("buffering", sélection de la ligne sortante... etc).

(iv) le délai de propagation (noté P_i, pour la ligne i) [1].

Les composantes (iii) et (iv) sont supposées négligeables. ($K = 0$ et $P_i = 0$ pour $i = 1, \ldots, M$). Notons T_i le temps moyen que passe un message dans le système i, où le système i est défini comme étant la ligne i (un serveur) plus la file d'attente de messages devant cette ligne (T_i contient donc les deux composantes (i) et (ii) relatives à la ligne i), pour tout $i = 1, \ldots, M$. Notons T, le délai moyen d'un message. On peut démontrer que (cfr appendice 2) :

$$T = \sum_{i=1}^{M} \frac{\lambda_i}{\gamma} T_i$$

Cette équation est parfaitement générale. Le problème d'analyse se réduit ainsi au calcul des T_i.

Il peut être intéressant d'introduire la notion de longueur de chemin. Notons n_{jk}, la longueur de π_{jk} (i.e. le nombre de lignes rencontrées par un message sur π_{jk}). On s'intéressera plus particulièrement à la **longueur moyenne d'un chemin** ($\bar{n}$) définie par :

$$\bar{n} = \sum_{j=1}^{N} \sum_{k=1}^{N} \frac{\gamma_{jk}}{\gamma} n_{jk}$$

Il est évident que $\lambda = \sum_{j=1}^{N} \sum_{k=1}^{N} \gamma_{jk} n_{jk}$; dès lors, $\bar{n} = \frac{\lambda}{\gamma}$ (résultat général). On en déduit que $T = \bar{n} \sum_{i=1} \frac{\lambda_i T_i}{\lambda}$

---

(1) Ce délai dépend de la longueur physique de la ligne.

Il s'agit à présent d'**évaluer** T_i (i = 1,..., M). Grâce à l'"Independence Assumption", nous déduisons que : (1)

$$T_i = \frac{1}{\mu C_i - \gamma_i}$$

pour i = 1,..., M. Kleinrock souligne les différences essentielles qui existent entre les réseaux à commutation de données réels (par exemple ARPANET) et le modèle qu'il en propose (cfr [KLEINROCK 70]). Dans une étude de validation de modèle (cfr [KLEINROCK 74], [KLEINROCK 76]), il étend son modèle de base afin d'inclure diverses caractéristiques telles que le temps de traitement aux noeuds, le délai de propagation ... etc qui sont pertinentes dans le cas d'ARPANET. Il donne une expression plus précise de T (2), qui tient compte de K , de P_i et des différences, dans la contribution au délai, entre les messages de commande et les messages de données purs (cfr [KLEINROCK 70]) de prendre en compte des distributions de longueurs de message plus générales (que la distribution exponentielle) (3) en utilisant l'équation de Pollaczek-Khuichin (files M/G/1). Kleinrock étudie également (cfr [KLEINROCK 76]) le comportement de T en fonction de γ (le throughput).

Wong met l'accent sur la notion de classe de message (cfr [WONG 78a]).

---

(1) Résultat élémentaire des files M/M/1.

(2) Dont la précision, dans le cas d'ARPANET, a été vérifiée par simulation.

(3) Qui peut être ou ne pas être bonne pour une application donnée.

Il déduit du modèle de Kleinrock, l'expression de la distribution (et donc de la moyenne et de la variance) du délai des messages pour chaque classe ainsi que pour des groupes de classes de messages (1). Il propose donc une caractérisation plus fine (que celle de Kleinrock) du délai des messages dans un RCM.

Signalons aussi (avec [WONG 78^b]) (2) le travail de Labetoulle et Pujol (cfr [LABETOULLE 76]), qui considèrent un modèle de réseau ouvert avec longueur de message déterministe, et en déduisent une expression approchée du délai moyen. Ils proposent une validation de leur résultat par simulation.

Signalons enfin l'expression du délai moyen d'un message dans un réseau hiérarchique (3) (cfr 5) ci-après), proposée par Kleinrock et Kamoun (cfr [KLEINROCK 80]).

---

(1) Tant pour un routage fixe qu'aléatoire.

(2) Ce travail décrit les résultats de recherche dans le domaine de la modélisation des réseaux informatiques par des réseaux de files d'attente en insistant plus particulièrement sur l'analyse de délai, le "buffer management", le contrôle de flux et la modélisation du "réseau-utilisateur" (les stations). (cfr références in [WONG 78^b]).

(3) En termes des délais moyens dans les sous-réseaux le composant.

4) Conception des réseaux centralisés

Nous nous proposons, à présent, d'étudier le problème du design d'une "station" (cfr 1) ci-dessus), ce qu'on a appelé le "Local Access Network Design", ou encore le design des réseaux centralisés. Le problème consiste essentiellement à connecter des terminaux éloignés à un centre informatique unique. Le réseau en étoile et le "minimum spanning tree" (cfr ci-après) sont des solutions extrêmes. Une solution intermédiaire consiste à relier les terminaux au centre via un **"réseau multipoint"** (i.e. un arbre constitué de liaisons multipoints). Nous étudions ce problème en (i) ci-dessous (remarquons que dans ce premier problème, le centre peut fort bien être un concentrateur, un commutateur ou encore un centre de traitement de données !). Nous étudions ensuite (en (ii)) le problème de l'emplacement de **concentrateurs** (utiliser un point de concentration peut être une alternative intéressante au réseau multipoint). Nous proposons enfin (en (iii)) quelques résultats relatifs au problème plus spécifique de l'**allocation de capacités** (CCAP) aux lignes d'un réseau centralisé.

(i) Conception des réseaux multipoint ("multidrop network").

Le problème de la connection de sites éloignés (par exemple des terminaux) à un centre unique (par exemple un ordinateur, un concentrateur, un commutateur ... etc), par l'intermédiaire de liaisons multipoints, de façon à former une topologie en arbre (1), a été (et est) beaucoup étudié dans la littérature ("multidrop networks").

---

(1) Remarquons que la topologie la plus économique, même pour un réseau centralisé, peut ne pas être un arbre ([FRANK 72^a] en donne un contre-exemple extrêmement simple); [YAGED 71] montre que tel est dependant le cas lorsque les coûts des lignes sont des fonctions convexes (donc continues) des capacités. Disons que la structure d'arbre semble raisonnable dans ce genre de problème (qu'elle simplifie grandement en éliminant le problème du routage !).

On pourrait concevoir deux topologies extrêmes, l'étoile (peu de congestion de trafic, très coûteuse, très fiable [1]) et le "minimum spanning tree" (MST) [2] (congestion de trafic, la moins coûteuse, très peu fiable). Le problème de design consiste à construire un réseau (à liaisons multipoints en arbre)minimisant le coût des lignes et tel que le délai moyen (dû à la congestion des lignes) et la fiabilité soient acceptables.

De nombreux auteurs ont étudié le **"CMST problem"** ("Constrained Minimum Spanning Tree problem"), qui est une simplification du problème général de design décrit ci-dessus, et dont nous proposons la formulation qu'en a donnée [CHANDY 72] :

étant donné 1) n-1 terminaux (le terminal i se trouvant à l'emplacement i, i = 2, 3,..., n);

2) un centre (à l'emplacement 1);

3) une matrice de coût C, (n . n) et symétrique, dont l'élément (i, j) est le coût d'une ligne directe entre les emplacements i et j;

---

(1) Une mesure de fiabilité intéressante, dans le cas des réseaux en arbre, pourrait être le nombre de sites éloignés incapables de communiquer avec le centre par suite d'une panne d'une seule ligne. Une contrainte de fiabilité significative consisterait alors à imposer une limite supérieure à ce nombre. En général, plus un réseau est fiable, plus il coûte cher ! (cfr [CHANDY 72]).

(2) Etant donné n points et une matrice de coût (n . n) symétrique, dont l'élément (i,j) est le coût de connection des points i et j, un MST est un arbre qui connecte tous les points au coût minimum.

4) le taux de trafic a_i généré par le terminal i ($i = 2, 3, \ldots, n$);

5) la capacité unique B pour les lignes du réseau;

et 6) le nombre maximum de terminaux qui peuvent être incapables de communiquer avec le centre par suite de la panne d'une ligne (contrainte de fiabilité);

déterminer l'arbre de coût minimum connectant les terminaux au centre, satisfaisant la contrainte de fiabilité et tel que le trafic total sur une ligne quelconque n'excède pas B.

Remarquons que les délais de file d'attente sont traités implicitement par l'intermédiaire du calcul de B (1).

Le CMST est un problème relativement simplifié (par rapport au problème général). C'est évident. Néanmoins, [SICKLE 77] a démontré que le CMST sans contrainte de fiabilité (donc encore plus simple !) est NP-complet (2) (cfr définition p.12).

Cette constatation n'a pas empêché divers auteurs de développer des **algorithmes optimaux** fondés essentiellement sur les méthodes "branch-and-bound" (3).

---

(1) Le délai moyen de file d'attente sur une ligne est en fait une fonction complexe de la distribution de la longueur des messages, de la stratégie de polling, du nombre de terminaux connectés à la ligne (multipoint) ... etc.

(2) [SICKLE 77] prétend que les algorithmes optimaux relatifs au CMST ne peuvent résoudre des problèmes de plus de 60 terminaux (estimation valable en 1977 !).

(3) Cfr [CHANDY 72], [CHANDY 73], [ELIAS 74], [GAVISH 80], [KERSHENBAUM 83], ... etc.

Les buts poursuivis par ces auteurs sont les suivants :

(i) trouver des techniques optimales plus performantes [1] (temps CPU, place mémoire) afin de pouvoir traiter des problèmes de tailles de plus en plus grandes (donc de plus en plus réalistes);

(ii) vérifier l'efficacité des heuristiques (dont nous parlerons ci-après) à partir des solutions optimales (lié à (i));

(iii) déduire (des études liées au développement des techniques optimales) des résultats sur lesquels de nouvelles heuristiques pourraient être développées.

Signalons l'algorithme optimal de [CHANDY 72], un "b & b" utilisant le coût du MST (sans contrainte) [2] comme limite inférieure sur le coût d'une solution faisable du CMST (et dans le but de guider la recherche). Les auteurs utilisent également la propriété suivante (qu'ils démontrent) : si les terminaux $i_1, \ldots, i_r$ sont directement connectés au centre dans le MST, alors il existe une solution optimale au CMST dans laquelle les terminaux $i_1, \ldots, i_r$ sont directement connectés au centre (cfr [CHANDY 72] pour la description de leur algorithme sur un exemple). [KERSHENBAUM 83] développe l'étude préliminaire de [CHANDY 72] en s'attachant aux divers objectifs énoncés ci-dessus. Signalons encore l'approche de [MATSUI 78].

---

(1) Via des techniques d'accélération en améliorant les bornes (utilisées dans le "b & b") afin de réduire le nombre de sous-problèmes à examiner ... etc.

(2) Le MST sans contrainte peut être déterminé très rapidement (cfr [DIJKSTRA 59]). [TANENBAUM 81] décrit également les algorithmes de Prim et de Kruskal (pour trouver le MST).

L'auteur présente une procédure dont le but est de trouver la vraie solution optimale d'un réseau centralisé en utilisant les techniques de la programmation entière. Cependant le problème est contraint à prendre une forme hiérarchique. En particulier, la plupart des noeuds sont déclarés "esclaves", quelques-uns "maîtres" et un unique noeud est appelé "centre". Les esclaves doivent être connectés aux maîtres et les maîtres doivent être connectés directement au centre (il ne peut y avoir ainsi de connection esclave-esclave ou maître-maître). Cette contrainte sur le design du réseau est très lourde et permet de réduire grandement la quantité de calculs nécessaires pour obtenir la solution. Cette quantité semble raisonnable dans le cas de petits réseaux, mais l'auteur ne donne aucune indication sur sa croissance lorsqu'augmente le nombre d'esclaves et/ou de maîtres.

De nombreuses **heuristiques** (relatives au CMST) efficaces (en temps CPU et place mémoire) et produisant des résultats assez bons en apparence, ont été développées (et utilisées) (1). D'une manière générale, les techniques itératives (2) (relatives au design des réseaux informatiques) utilisent les **méthodes d'échange** (de noeuds ou de lignes) de façon heuristique (cfr [FRANK 72^a]). Les méthodes d'échange sont des procédures de recherche qui optimisent la structure du réseau en procédant par modifications de petites parties de celle-ci (transformations topologiques locales).

---

(1) Cfr [ESAU 66], [MARTIN 67], [DOLL 69], [SHARMA 70], [FRANK 71], [CHOU 73], [YOSHIMURA 78], [KERSHENBAUM 80]... etc.

(2) [FRANK 72^a] signale aussi la possibilité d'un design interactif (le concepteur, par l'intermédiaire d'un terminal interactif, est capable de tester rapidement des alternatives de design qu'il propose lui-même) (cfr références in [FRANK 72^a]).

On commence avec un réseau satisfaisant toutes les contraintes [1]. On applique ensuite des méthodes d'échange à ce réseau initial jusqu'à trouver un nouveau réseau satisfaisant toutes les contraintes et de coût inférieur. Le processus est répété jusqu'à ce qu'on ne parvienne plus à baisser le coût. Le réseau résultant est alors appelé "optimum local". Le processus peut être répété à partir d'un nouveau réseau initial, ou en utilisant d'autres méthodes d'échange, ou encore, en utilisant les mêmes méthodes d'échange, mais dans un ordre différent. La qualité de la méthode dépend fortement de celle des transformations topologiques locales choisies. Nous nous proposons de décrire (cfr appendice 3) brièvement le principe de quelques-unes de ces heuristiques. Nous avons choisi :

(i) la procédure d'Esau-Williams (cfr [ESAU 66]) [2],

(ii) l'heuristique de Frank, Frisch, Chou et Van Slyke (cfr [FRANK 71[a]]) [3],

(iii) la méthode d'approximation de Vogel (VAM)

et (iv) l'heuristique de Yosnimura (cfr [YOSNIMURA 78]).

---

(1) Ce réseau initial est sélectionné "à la main" ou généré par un programme de design différent.

(2) Méthode appelée "Multipoint Path Creation" (cfr [FRANK 72[a]]).

(3) Méthode appelée "Circuit Deletions" (cfr [FRANK 72[a]]).

On peut encore citer une technique dite "Simple Path Exchange" (cfr [SHARMA 70]) décrite dans FRANK 72[a] , l'algorithme de Martin (cfr [MARTIN 67]), l'heuristique de Chou et Kershenbaum (cfr [CHOU 73]), ainsi que les algorithmes de Kruskal et de Prim (cfr [TANENBAUM 81]).

(ii) Le problème de l'emplacement des concentrateurs

Les concentrateurs sont des appareils qui reçoivent des messages en provenance de plusieurs lignes (input) et les envoient sur une seule ligne (output) à haute vitesse qui peut être reliée directement au centre (de traitement de données). On utilise les concentrateurs lorsqu'il s'agit de relier plusieurs sites éloignés à un centre unique. Toute site éloigné est un candidat concentrateur et il peut y avoir un nombre arbitraire de concentrateurs dans le réseau. Le trafic total entrant dans un concentrateur ne peut excéder sa capacité, sa capacité étant déterminée à partir de considérations de files d'attente. Dans le cas le plus simple, les lignes input du concentrateur forment un réseau en étoile (cfr [SICKLE 77]).

[TANENBAUM 81] propose une présentation claire du problème. Il distingue deux sous-problèmes :

(1) le problème du choix de concentrateur;

(2) le problème de l'emplacement des concentrateurs, proprement dit.

Ces deux problèmes sont extrêmement complexes. Le premier consiste à relier n sites [(1)] à m concentrateurs (les emplacements des sites et des concentrateurs étant donnés) de façon à minimiser le coût global de connection, chaque site devant être relié à un et un seul concentrateur et chaque concentrateur pouvant gérer un nombre maximum (donné) de sites (contraintes).

---

(1) Un site peut être constitué d'un simple terminal, d'un gros ordinateur, de plusieurs terminaux ... etc.

Il y a 2^{mn} allocations possibles ! Dans le second problème, on suppose qu'il y a m emplacements potentiels pour les concentrateurs (le nombre de concentrateurs devient donc une variable du problème) (1).

[SICKLE 77] a démontré que le problème de l'emplacement des concentrateurs est NP-complet.

[TANENBAUM 81] décrit une heuristique pour le premier problème et renvoit à [KERSHENBAUM 75] pour une technique optimale. Le second problème est en fait un problème de "Plant Location" (cfr (i) p. 20 et (i) p. 24) bien connu. Nous renvoyons le lecteur aux références citées pp. 20 et 24. [TANENBAUM 81] décrit deux heuristiques (celles de [KUEHN 63] et [FELDMAN 66]). Citons encore les heuristiques de [BAHL 72], [WOO 73], [TANG 74] et [MAC GREGOR 77].

[CHOU 78] présente une approche au problème de l'emplacement optimal des "installations permettant d'accéder au réseau" (terminaux, concentrateurs,.... etc). Les auteurs prétendent que leur procédure peut être appliquée tant au "local access design" qu'au "backbone network design". Cependant, la plus grande partie de leur étude concerne le "local access design". Ils estiment que la décomposition du réseau en régions constituées de voisins proches connectés entre eux, est une approche heuristique efficace. Leur approche semble plus efficace que les algorithmes de Mac Gregor-Shen et Chou-Kershenbaum (cfr [MAC GREGOR 77] et [CHOU 73]).

[DE BACKER 78] présente une approche qui combine diverses procédures sous-optimales bien connues afin de réaliser un design intégré incluant les variables (de design) suivantes : la topologie, les capacités des lignes, le nombre et l'emplacement des concentrateurs.

---

(1) On introduit une variable binaire supplémentaire indiquant si tel concentrateur est ou n'est pas utilisé (on associe un coût fixe à chaque concentrateur).

Ces variables sont optimisées de façon à minimiser le coût tout en fournissant un temps de réponse raisonnable. L'auteur utilise les quatre algorithmes suivants :

(i) Esau-Williams,
(ii) Vam,
(iii) Kruskal,
et (iv) Prim.

afin de générer une solution initiale qu'il modifie dans le but de l'améliorer.

(iii) Le problème de l'allocation des capacités

Nous avons déjà signalé l'extrême complexité du **CCAP** (cfr I.5.1). Si l'objectif poursuivi est de minimiser le coût, tout en maintenant le délai moyen inférieur ou égal à une valeur donnée, alors il existe une procédure optimale de solution à ce problème dans le cas des réseaux centralisés dont la topologie est un arbre (cfr [FRANK 71^{a}]). Les auteurs suggèrent deux variations à leur algorithme de base : le transformer en algorithme sous-optimal dans le cas de topologies quelconques, ou encore, accepter des trafics variant au cours du temps. Il apparaît que cet algorithme, bien programmé, peut résoudre des réseaux de plusieurs milliers de noeuds en quelques minutes CPU. Il est clair que le routage, dans le cas d'un arbre, n'est plus une variable de design, ce qui explique en grande partie ce résultat. Nous renvoyons à [FRANK 71^{a}] pour une description particulièrement claire de cette procédure. [FRANK 72^{a}] suggère alors d'utiliser cette procédure optimale en conjonction avec les techniques itératives citées et/ou décrites ci-dessus.

Si l'objectif poursuivi est de minimiser le délai moyen à coût fixe, des résultats analytiques sont disponibles lorsque les capacités des lignes sont des variables continues (cfr [KLEINROCK 76], [MEISTER 71] et [KOMATSU 78], nous en reparlerons en I.5.5.) ci-après). Nous avons déjà fait remarquer que le trafic sur les lignes d'un arbre est connu à priori. Par contre, dans le cas des réseaux distribués, les capacités et les flux des lignes interagissent, puisqu'un bon algorithme de routage aura tendance à diriger le flux de façon à éviter les parties saturées du réseau. Ainsi, l'allocation des capacités aux lignes (CCAP) et le routage doivent être considérés simultanément, ce qui explique l'absence de résultats optimaux pratiques dans le cas des réseaux distribués.

[SICKLE 77] souligne l'extrême complexité du CCAP en démontrant que le problème qui consiste à allouer des capacités aux lignes d'un réseau, afin de minimiser le temps de réponse moyen, tout en respectant une contrainte budgétaire, est NP-complet.

5) Conception des "backbone networks"

Nous présentons, dans ce paragraphe, quelques éléments de solution au problème, extrêmement complexe, de la conception des "backbone networks". (on parle aussi de réseaux maillés ou distribués). On peut préférer un réseau maillé à un réseau arborescent, pour les raisons suivantes :

(i) éviter des saturations de noeuds (dans le cas de trafics élevés entre noeuds);

(ii) satisfaire des exigences de fiabilité;
... etc.

Dans un arbre, il n'y a qu'un chemin entre deux noeuds quelconques, ainsi le trafic sur les lignes est connu à priori. Dans le cas du design distribué, les capacités et les flux des lignes interagissent, puisqu'un bon algorithme de routage aura tendance à envoyer le flux de façon à éviter les parties saturées du réseau. Ainsi, l'allocation des capacités aux lignes et le routage du réseau devraient-ils être considérés **simultanément.** A cause de cette difficulté, il n'existe aucun résultat optimal pratique pour les réseaux distribués (il existe d'assez "bonnes" heuristiques).

De nombreux efforts ont été fournis pour résoudre le problème du **design d'ARPANET,** et certains algorithmes sous-optimaux ont été proposés. (1)

Nous décrivons, dans l'appendice 4 , l'approche suivie par [FRANK 70] à propos du design d'ARPANET. Cela nous permet d'introduire la **méthode classiquement utilisée pour le CCNDP des réseaux distribués,** méthode utilisée également dans le cas des réseaux centralisés, et dont les éléments clés (cfr [TANENBAUM 81]) sont les suivants :

(i) choix de topologies initiales faisables (i.e. satisfaisant certaines contraintes);

(ii) génération de réseaux légèrement modifiés (au moyen de "transformations topologiques locales") à partir d'un réseau donné (via des heuristiques dites "de perturbation");

---

(1) Cfr par exemple [FRANK 70], [FRATTA 73], [MEISTER 71], [CANTOR 72], [GERLA 73],..., etc.

(iii) application d'algorithmes sous-optimaux (que l'on peut exécuter de nombreuses fois, parce que peu exigeants en temps CPU) relatifs, par exemple, au routage (allocation de flux aux lignes) et à l'allocation des capacités aux lignes.

L'essentiel de ce paragraphe consistera en une introduction sommaire (en (i) ci-après) aux divers résultats (relatifs au "Backbone Design") présentés dans [KLEINROCK 76], ainsi qu'en une courte étude (en (ii) ci-dessous) du design des grands réseaux (cfr [KLEINROCK 80]). Nous aimerions, en préliminaire aux travaux de Kleinrock, présenter l'approche de solution (quelque peu étoffée) proposée par [TANENBAUM 81] et insister encore sur le problème du routage (cfr appendice 5).

(i) Un résumé des efforts de recherche relatifs au "backbone design"

(Référence principale : [KLEINROCK 76]).

Nous avons décrit (en 3) ci-dessus) l'**analyse** de Kleinrock relative au "backbone design" (modèle et analyse de délai). En fait, le problème du réseau de communication est également un problème de **design.** Kleinrock retient un critère d'optimisation, T (cfr I.5.3), une contrainte de coût (1) et trois variables de design ($\{C_i\}$, $\{d_i\}$ (2) et la topologie).

(1) Notons D, le coût total du réseau de communication :

$$D = \sum_{i=1}^{M} d_i (C_i),$$

où $d_i (C_i)$, une fonction arbitraire de la capacité et de la ligne, désigne le "coût de construction" de la ligne i avec la capacité C_i. Nous renvoyons le lecteur à [KLEINROCK 70], où l'auteur étudie et compare (entre elles et avec les données disponibles en pratique) trois fonctions d_i (dont nous parlerons plus loin).

(2) Il ne décrit pas la procédure de routage permettant d'atteindre ces valeurs (tâche très difficile en général).

Il définit **quatre problèmes d'optimisation** (consistant tous à minimiser T) (cfr figure "problèmes d'optimisation étudiés par Kleinrock").

nom (1)	donnée(s)	variable(s)	contrainte
CAP	$\{\lambda_i\}$ top (2)	$\{C_i\}$	$D = \sum_{i=1}^{M} d_i(C_i)$
FAP	$\{C_i\}$ top	$\{\lambda_i\}$	———
CFAP	topologie	$\{C_i\}$, $\{\lambda_i\}$	$D = \sum_{i=1}^{M} d_i(C_i)$
TCFAP	———	$\{C_i\}$, $\{\lambda_i\}$, top.	$D = \sum_{i=1}^{m} d_i(C_i)$

-. problèmes d'optimisation étudiés par Kleinrock .-

On peut définir également les formes duales de ces problèmes (minimiser D sous la contrainte $T \leqslant T_{max}$, où T_{max} est une valeur donnée).

Une des difficultés de ces problèmes réside dans le fait que les capacités, en pratique, sont choisies hors d'un ensemble fini d'options possibles. Ainsi, dans le cas du **CAP**, et pour des fonctions de coût discrètes, sommes-nous réduits à des solutions heuristiques ou encore à des procédures énumératives extrêmement complexes.
Le CAP à capacités discrètes est un problème de programmation dynamique (cfr [CANTOR 74]) résolu de manière efficace et optimale dans certains cas particuliers (cfr [FRANK 71[a]], où la topologie du réseau est supposée être un arbre).

---

(1) **C pour capacity**, F pour flow, T pour topology, A pour assignment et P pour problem.

(2) i.e. topologie.

Le CAP à capacités continues est résolu pour certaines fonctions de coût [1] : ces solutions continues permettent de guider de façon efficace une sélection parmi l'ensemble des capacités discrètes disponibles.

[KOMATSU 78] étudie le problème de l'allocation optimale des capacités (CAP). [KLEINROCK 72] a étudié le problème pour $\bar{T} = \sum_i \frac{\lambda_i}{\gamma} \bar{T}_i$ (cfr notations de 3) ci-dessus); [MEISTER 71] a généralisé le problème en minimisant

$$\left(\sum_i \frac{\lambda_i}{\gamma} T_i^{\alpha} \right)^{1/\alpha}$$

Les auteurs observent que la formulation de Kleinrock est équivalente à la minimisation du nombre total moyen de messages dans le réseau (en effet $N_i \simeq \lambda_i T_i$, où N_i désigne le nombre moyen de messages utilisant ou attendant la ligne i). Les auteurs minimisent

$$\left[\sum_i N_i \right]^{1/\alpha}$$

et proposent de nombreux résultats établissant des relations entre les solutions optimales à leur problème et à celui de [MEISTER 71].

Le **FAP** est un problème relevant de la théorie des flux ("network flow theory", cfr [FRANK 71b]).

---

(1) Solutions analytiques pour $d_i(C_i) = d_i C_i$ et $d_i(C_i) = d_i \cdot \log \alpha C_i$; solutions d'une équation non linéaire, dont il existe des techniques itératives de solution très efficaces, dans le cas $d_i(C_i) = d_i C_i^{\alpha}$ ($\alpha \in [0,1]$), plus proche de la réalité.

Sa simplicité réside dans le fait, d'une part, que la contrainte de capacité ($f_i/\mu < C_i$) est incluse comme fonction de pénalité dans l'expression de T (cfr I.5.3) et peut, dès lors, être ignorée, et, d'autre part, que tout minimum local est un minimum global (convexité de T). L'auteur propose (cfr [KLEINROCK 76] p.343) un algorithme optimal (1) et relativement efficace, basé sur la notion de "chemin le plus court" (2) (cfr [FLOYD 62] ou encore [TANENBAUM 81] p.41). Le routage qui résulte de cet algorithme est généralement déterministe alterné (cfr I.5.1). Il propose également une méthode sous-optimale produisant une procédure de routage fixe et réclamant moins de calculs. Cette méthode - qui, par ailleurs, conduit souvent à des résultats extrêmement bons - est efficace pour les grands réseaux dont les γ_{jk} ne diffèrent guère l'un de l'autre.

Kleinrock décrit une prodédure qui fournit un optimum local au **CFAP** pour chaque vecteur de flux initial (et faisable) qu'on lui propose (combinaison du CAP et du FAP). (3)

---

(1) "Flow Deviation (FD) Algorithm" (cfr [FRATTA 73]). [CANTOR 72] propose une méthode exacte de solution au FAP différente de la méthode de déviation de flux. La méthode FD est assez semblable à la méthode du gradient pour les fonctions de variables continues; on pourrait l'appliquer à une topologie initiale de connectivité supérieure à celle exigée (par exemple à un réseau "fully connected"). Durant l'optimisation, on peut alors supprimer des lignes dont le flux s'annule.

(2) La complexité d'un algorithme déterminant le plus court chemin entre deux noeuds est de l'ordre de N^2 à N^3, il peut donc être exécuté plusieurs fois dans la course à l'optimisation.

(3) Des lignes peuvent être éliminées lors de l'exécution du CFA.

Le **TCFAP** est en fait le vrai problème de design des réseaux. L'auteur l'étudie sous sa forme duale; il en propose une solution heuristique (1) - appelée "concave branch élimination method" ou CBE (cfr [GERLA 73]) - ayant la structure suivante :

répéter pour diverses topologies initiales
répéter pour divers vecteurs de flux initiaux

| CFA
| discrétiser les capacités
| FA .

(ii) La conception des grands réseaux

Les procédures heuristiques **existantes (relatives** au CCNDP) sont assez efficaces pour des réseaux de taille modérée (jusqu'à 75 noeuds, par exemple).
Citons, par exemple (cfr références dans [KLEINROCK 80]) :

(i) la méthode "branch exchange" (cfr [FRANK 72[a]]);
(ii) la méthode dite de "cut saturation" (idem);
(iii) la méthode dite "concave branch elimination" (cfr [KLEINROCK 76] et [GERLA 73]).

---

(1) Dont l'écart à l'optimum devrait être de l'ordre de 5 à 10 %. En faveur de l'approche heuristique, Kleinrock fait remarquer qu'on peut rarement prédire les γ_{jk} avec beaucoup de précision.

La complexité de ces heuristiques est de l'ordre de N^3 à N^6, où N désigne le nombre de noeuds du réseau (cfr [FRANK 72^a]). Ces heuristiques ne sont donc pas applicables aux futurs grands réseaux de plusieurs centaines de noeuds (temps CPU et place de stockage énormes).

Il s'agit dès lors de trouver de nouvelles procédures de design permettant de traiter le cas des grands réseaux.

[FRANK 72^a] suggère de partitionner le réseau en régions. Les raisons de ce regroupement ("clustering") des noeuds en régions sont les suivantes :

(i) réduire la complexité des calculs du problème de design topologique;

(ii) déterminer des régions dont les lignes sont à basses, moyennes et hautes capacités (de façon hiérarchique);

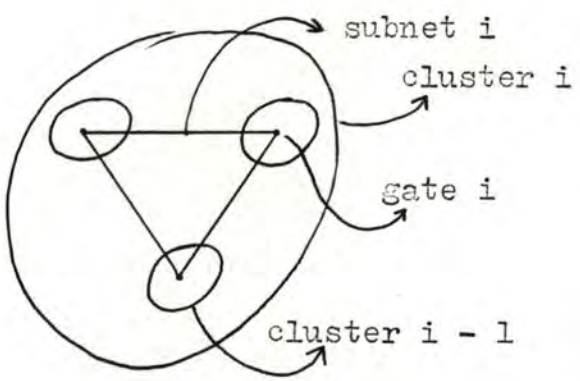
(iii) trouver les emplacements des points de concentration; ... etc.

Un problème important consiste alors à déterminer la taille et le nombre de régions. Il faut nécessairement définir une "mesure de proximité" (ce problème est épineux, car il faut tenir compte du coût des noeuds, des lignes ... etc, du trafic, du délai, de la fiabilité, du routage ... etc). [FRANK 72^a] suggère de déterminer la taille des régions de façon à minimiser les exigences de calculs de la procédure de design. Kleinrock et Kamoun reprennent cette idée (cfr [KLEINROCK 80]) et proposent une technique dite de "clustering hiérarchique" que nous explicitons ci-dessous.

L'idée générale du "clustering hiérarchique" est la suivante. Considérons le cas d'un design hiérarchique à 2 niveaux. Les noeuds sont regroupés en "clusters" et un noeud spécial (d'échange entre "clusters"), ou "gate", est sélectionné dans chaque "cluster".

Les sous-réseaux des différents "clusters" sont conçus séparément, tandis qu'un super-réseau de "gates" est conçu de façon à connecter ensemble tous les "clusters". (1) Le but des auteurs est de proposer une technique permettant de déterminer une structure de "clustering" (à m niveaux), à utiliser lors du design, qui minimise le coût des calculs de ce design. Cette décomposition devrait conduire au design de sous-réseaux plus petits pour lesquels on peut utiliser les statégies de design existantes.

L'idée principale du design hiérarchique est d'imposer une structure décomposable au problème de design (donc une indépendance entre variables de design) qui réduit fortement l'ensemble des solutions faisables sans cependant éliminer la solution optimale. De telles décompositions seront réalisées au moyen du "clustering" hiérarchique à m niveaux (MHC) de l'ensemble des noeuds (cfr figure ci-dessous).



noeud = cluster 0
= gate 1

-. clustering hiérarchique .-

---

(1) Le trafic entre deux noeuds quelconques suit la voie hiérarchique naturelle.

Des techniques spécifiques de "clustering", basées sur certaines mesures de proximité, permettent de regrouper les noeuds en "clusters". Le choix des "gates" (via des règles de sélection) dépend fortement de la contrainte de fiabilité. La topologie sous-jacente à un MHC, et qu'il faut concevoir, est notée MHT. Il s'agit donc de déterminer le MHC qui minimise le coût des calculs relatifs au design du MHT correspondant. (1)

Les auteurs présentent des résultats optimaux relatifs au nombre de clusters, de superclusters ... etc, ainsi qu'au nombre de niveaux hiérarchiques. Les auteurs proposent également une expression du délai moyen d'un message dans un réseau hiérarchique.

---

(1) On suppose que le coût des calculs relatifs au design d'un sous-réseau de niveau k reliant n "gates" de niveau k est égal à n^{α_k} (cfr[FRANK 72^a]) et que le coût total des calculs est égal à la somme des coûts relatifs aux différents sous-réseaux des différents niveaux.

I.6. ETUDE SIMULTANEE DES PROBLEMES DE L'ALLOCATION DES FICHIERS ET DE LA CONCEPTION DU RESEAU DE COMMUNICATION

Nous avons étudié séparément deux problèmes de design distincts : le FAP (cfr p.1), de I.1. à I.4., et le CCNDP (cfr p.57) en I.5. Nous étudions à présent le problème de l'intégration du FAP au CCNDP. Nous proposons un aperçu de divers travaux (dont fait partie [MAHMOUD 76]) relatifs à ce problème.

Au préalable, nous voudrions signaler rapidement quelques travaux ayant trait à divers problèmes de design liés au FAP, mais dont l'étude dépasserait le cadre de ce chapitre.

[GOSH 76] et [SRINIVASAN 80] étudient le problème de l'exploitation, lors de la distribution d'une BD sur un réseau informatique, des possibilités de recherches en parallèle qu'offre le réseau lui-même, afin de réduire le temps de réponse aux "queries" (cfr app. 6).

[RAMAMOORTHY 79^{a}] présente (cfr appendice 7) différents problèmes relatifs à la gestion des données dans un système à BDD (cfr également [ROTHNIE 77], [RAMAMOORTHY 79^{b}] et [WAH 84]). Parmi ces problèmes, nous aimerions épingler celui du "query processing" (traitement d'un "query", accédant à des données stockées à différents endroits du réseau, qui consiste à arranger les transmissions et les traitements locaux de données en tenant compte des délais de communication entre sites, ainsi que des possibilités de traitement en parallèle) (cfr [HEVNER 79]).

Nous signalons enfin (cfr appendice 8) deux approches au problème général de la conception des systèmes informatiques distribués. Nous suggérons au lecteur de leur porter un intérêt tout particulier.

FAP et CCNDP

1. Introduction

Certains auteurs ont étudié le FAP et le CCNDP simultanément. Sans prétendre aucunement à l'exhaustivité, nous avons choisi de présenter brièvement les travaux suivants : [CASEY 73], [MAHMOUD 76], [IRANI 82] et [CHEN 80].

Nous disposons en fait de tous les éléments pour concevoir des techniques de solution relatives à ce problème.

On pourrait, par exemple, se limiter à des réseaux arborescents, et combiner la procédure d'ESAU-WILLIAMS (cfr I.5.4) avec la solution au SFAP proposée par Casey (cfr I.1); c'est ce que propose [CASEY 73].

Nous avons passé en revue, dans les paragraphes précédents, diverses techniques de solution, heuristiques ou optimales. Nous avons pu constater l'intérêt porté aux heuristiques de type "add-drop" (cfr p. 29 , par exemple) ainsi qu'à la méthode (optimale) "branch-and-bound". Ces techniques sont reprises, respectivement, par [MAHMOUD 76] et [CHEN 80]. Les paragraphes précédents nous suggèrent également de développer des heuristiques afin d'obtenir des bornes supérieures sur le coût de la solution optimale, de négliger certaines contraintes (par exemple d'intégralité) afin d'obtenir des bornes inférieures sur ce même coût, d'utiliser divers critères (qui se basent sur le modèle retenu) permettant de réduire la complexité du problème, ... etc. Ainsi, chacun des travaux que nous présentons ci-dessous ne représente-t-il qu'une approche parmi tant d'autres, existantes ou potentielles. Tel est le cas, par exemple, du travail de Mahmoud et Riordon (cfr [MAHMOUD 76]) que nous avons choisi d'étudier (plus en détail) et d'implémenter (cfr chapitre II). Ces auteurs supposent que la topologie est fixée et se proposent d'allouer de façon optimale (ou quasi-optimale) les fichiers (FAP) et les capacités (CCAP, cfr I.5.). On pourrait fort bien imaginer de combiner leur procédure avec un algorithme générant des topologies (cfr I.5). On pourrait encore (sans doute), par exemple, inclure leur procédure dans l'algorithme général proposé par [IRANI 82] ... etc.

Ce que les paragraphes précédents nous ont essentiellement appris, c'est l'extrême complexité du FAP (cfr I.1.) et du CCMDP (et même de quasi tous les sous-problèmes constituant le CCMDP, cfr I.5). En outre, la complexité de l'interdépendance entre ces deux problèmes semble encore plus évidente. En conséquence, il s'agit de porter tous ses efforts sur le développement d'heuristiques (cfr [MAHMOUD 76]), sans cependant négliger l'intérêt économique des solutions optimales (cfr [CHEN 80]) dans le cas de petits systèmes.

Nous profiterons de ce paragraphe I.6., pour présenter les aspects suivants de [MAHMOUD 76] : (1)

(i) nature du problème étudié;

(ii) hypothèses principales sous-jacentes au modèle proposé (2) et critiques énoncées dans la littérature;

(iii) techniques de solution suggérées par les auteurs;

(iv) tests de performance proposés par les auteurs.

Le lecteur pourra trouver une description de [CASEY 73], [IRANI 82]et [CHEM 80], dans l'appendice 9 .

1.2. Une première présentation de [MAHMOUD 76]

(i) Nature du problème étudié

Mahmoud et Riordon étudient simultanément les problèmes de l'allocation de fichiers aux noeuds (FAP) et de capacités aux lignes (CCAP) d'un réseau informatique distribué, dans le but de minimiser le coût global du système (3),

---

(1) Indépendamment de toute contribution personnelle (cfr chapitre II).

(2) Pour une formulation précise du problème étudié et de l'heuristique proposée par les auteurs, cfr chapitre II.

(3) Coûts de stockage des fichiers et de location des lignes de communication.

tout en satisfaisant la demande des utilisateurs [1], ainsi que des contraintes de disponibilité des fichiers et de délai du réseau. Ils supposent que tout fichier peut, à priori, avoir (ou ne pas avoir) une copie stockée en n'importe quel noeud du réseau. Ils supposent que la topologie du réseau est connue et fixée, et que le routage est fixe.

Ils utilisent l'analyse de **délai** de Kleinrock (cfr I.5.3) et imposent une limite supérieure sur le délai (de bout-en-bout) moyen (noté T) d'un message [2].

La **disponibilité d'un fichier** est définie comme étant la probabilité que ce fichier soit accessible lorsque des utilisateurs le sollicitent. Cette grandeur dépend de la fiabilité du réseau (cfr I.5.2)) [3]

---

(1) Cette demande s'exprime sous la forme de quatre matrices de trafic (dont l'élément ij de chacune représente le trafic émanant du noeud i et à destination du fichier j) relatives, respectivement, au "query traffic", à l'"update traffic", au "query return traffic" (les réponses aux "queries") et à l'"update return traffic" (les ACKs, par exemple).

(2) Cfr expression du délai moyen d'un message (T) p.72 ; rappelons que la moyenne est prise sur toutes les classes de messages.

(3) Ils supposent disposer des r_{ik}, pour tout couple de noeuds (i, k), où r_{ik} désigne la probabilité d'une communication réussie entre les noeuds i et k (cfr [HANSLER 72] dans I.5.2.), pour le calcul des ces r_{ik}, à partir de la topologie du réseau et des fiabilités des noeuds et des lignes.

ainsi que des emplacements de stockage des copies du fichier [1]. Ils imposent une limite supérieure sur la disponibilité de chaque fichier.

Il existe un **compromis** à atteindre entre le coût de stockage, le coût de communication, la disponibilité et le délai. Nous savons déjà qu'il existe un premier compromis, quant au coût de communication, entre le "query traffic" et l'"update traffic" (cfr remarque p.15). Il est évident, par définition, que la disponibilité d'un fichier croît et décroît avec le nombre de copies de ce fichier stockées dans le réseau. La contrainte de délai réalise un couplage (que l'on constatera analytiquement lors de la formulation précise du modèle, cfr chapitre II) entre le FAP et le CCAP. En effet, le délai moyen d'un message (cfr expression de T p.72) dépend des capacités (C_i) et des flux (λ_i), eux-mêmes dépendant de l'allocation des fichiers, via la procédure de routage fixe retenue.

(ii) Hypothèses et critiques

Nous aimerions, à présent, souligner quelques-unes des hypothèses principales sous-jacentes au modèle (cfr chapitre II) proposé par les auteurs, et rendre compte de diverses critiques que l'on pourrait émettre à leur sujet.

Remarquons tout d'abord que la **topologie** est fixée, en d'autres termes, on suppose que les emplacements des noeuds et des lignes de communication sont connus et fixés. Le **routage** est supposé fixe et la table de routage est supposée donnée. Ainsi, le CCNDP se réduit-il simplement au CCAP (cfr I.5). Soulignons bien le fait que l'analyse de délai, ainsi que les routines d'addition et de suppression de copies de fichiers (qui font partie de l'heuristique proposée par les auteurs, cfr chapitre II), dépendent crucialement de l'hypothèse sur le routage (afin d'obtenir les changements relatifs aux trafics sur les lignes).

---

(1) Cfr chapitre II, pour une expression analytique de la mesure de disponibilité retenue par les auteurs.

Il est exclu d'envisager l'utilisation d'un routage adaptatif dans le cadre de cette heuristique (cfr I.5.5 pour une brève discussion sur ce problème; cfr [LANING 83] pour une technique de solution, à base de simulation, au FAP sur réseaux utilisant le routage adaptatif). Remarquons également qu'ils supposent que le **"query traffic"** (et le "query return traffic"), relatif à une requête originaire d'un noeud donné et à destination d'un fichier donné, est **réparti uniformément** sur toutes les copies de ce fichier stockées dans le système. On aimerait, en fait, que ce trafic soit dirigé vers le noeud le plus proche contenant une copie du fichier. Le **délai moyen** d'un message est calculé **sur toutes les classes de messages** (cfr I.5.3)) : imposer une contrainte de délai par classe de messages eut été une approche plus fine et plus réaliste (cfr par exemple [LANING 83]). En effet, les utilisateurs du réseau, à chaque noeud, pourraient alors spécifier une mesure significative pour les destinations les intéressant, et ne seraient pas limités à un délai global. Remarquons, en particulier, que certaines classes de messages pourraient fort bien ne pas subir de contrainte de délai.

Les mesures de **disponibilité** (retenues par les auteurs) calculent la probabilité qu'au moins une copie (du fichier) soit accessible dans le réseau. Cela risque de poser des problèmes aux processus de mise à jour (qui exigent, en principe, que toutes les copies du fichier soient accessibles). On pourrait introduire une contrainte "de fiabilité de mise à jour" dans le modèle ("probabilité que toutes les copies . . . "), ce qui imposerait une limite supérieure sur le nombre de copies du fichier disponibles dans le réseau. Remarquons que le **processus de mise à jour** d'un fichier consiste à envoyer un message de mise-à-jour distinct à chaque copie du fichier; on pourrait imaginer envoyer un unique message visitant chaque copie du fichier le long d'un MST (minimum spanning tree, cfr I.5.4)).

Les auteurs considèrent les coûts de transmission comme étant des charges fixes mensuelles (lignes louées). Les lignes sont supposées unidirectionnelles.

Les auteurs supposent que tout fichier peut, à priori, avoir (ou ne pas avoir) une copie stockée en n'importe quel noeud. On aimerait, cependant, exiger (à priori) qu'une copie de tel fichier soit (ou ne soit pas) stockée en tel noeud (contrainte politique,sociale, de sécurité, ..., etc).

Les auteurs n'établissent pas de distinction entre **noeuds et stations** (cfr I.5.1)). Une telle distinction, qui n'est pas sans rapport avec la remarque précédente, permettrait sans doute de prendre en considération des topologies plus générales.

Ils n'imposent aucune **contrainte sur les capacités de stockage** aux noeuds. Nous avons déjà parlé de ce problème (cfr I.3.1)). Disons que cela peut conduire, dans certains cas, à des résultats indésirés. Par exemple, si l'"update traffic" est négligeable, les coûts de communication sont minimisés en allouant des copies de tous les fichiers à tous les noeuds. Remarquons également qu'ils n'imposent aucune contrainte sur le flux entrant et sortant de chaque noeud.

Ils n'introduisent pas la notion de dépendance programme-fichier (cfr I.3).

(iii) Techniques de solution proposées par les auteurs

Le problème d'optimisation résultant de leur modèle est un **problème de programmation entière non linéaire.** La fonction objective est linéaire, mais les contraintes sont non linéaires (cfr chapitre II); ainsi, les techniques déterministes permettant de résoudre ce problème ne sont-elles applicables qu'à des exemples de de très petite taille.

Les auteurs utilisent, dès lors, **deux types d'algorithmes** :

(i) une heuristique "add-drop";

et (ii) un algorithme de programmation entière (un "branch-and-bound") dont le but est d'estimer les performances de l'heuristique sur des problèmes de petite taille.

Nous aimerions insister sur la **complexité du problème** étudié, en faisant remarquer que l'expression suivante donne le nombre de configurations possibles des fichiers aux noeuds et des capacités aux lignes :

$$(nb\_cap)^{nb\_lignes} \cdot 2^{(nb\_noeuds \cdot nb\_fich)}$$

où nb_noeuds (resp. nb_lignes, nb_fich) représente le nombre de noeuds (resp. de lignes, de fichiers) et nb_cap, le nombre d'options disponibles pour les valeurs des capacités.

Nous ne décrivons ici que la **philosophie générale de l'heuristique** (nous en proposons une présentation plus détaillée au chapitre II).

La **technique adoptée** par les auteurs se base sur une décomposition en une "starting routine" (SR) et une "optimizing routine" (OR). La SR trouve des solutions initiales faisables (i.e. satisfaisant les contraintes de délai et de disponibilité) construites à partir des nombres minimaux de copies de fichiers nécessaires pour satisfaire les contraintes de disponibilité (nombres calculés par SR) (1).

---

(1) Les auteurs allouent les fichiers aux noeuds (ce qui permet de calculer le trafic entre paires de noeuds, via la table de routage), puis les capacités aux lignes.

OR soumet les solutions faisables initiales à une série d'additions et de suppressions de copies de fichiers qui garantissent la faisabilité de la solution. Au travers de ces deux routines, seules des solutions faisables sont recherchées. OR se termine lorsqu'il n'y a plus moyen d'améliorer le coût par additions ou suppressions de copies de fichiers. On applique OR séparément à chaque solution faisable initiale trouvée par SR, produisant ainsi un "optimum local" pour chacune. On suppose qu'au moins un de ces optima locaux est proche, en termes de la fonction objective, de la solution optimale globale.

(iv) Tests de performance

Il s'agit de tester une heuristique (1) ne fournissant aucun renseignement sur la "qualité" des solutions qu'elle génère. Idéalement, il faudrait disposer des solutions optimales aux problèmes sur lesquels on désire tester l'heuristique . Les auteurs ont développé, dans ce but, un algorithme optimal de type "branch-and-bound". Cet algorithme n'est, cependant, applicable qu'à des problèmes de petite taille. Pour des problèmes de taille plus importante (sans être encore très réaliste), les auteurs en sont réduits à arrêter l'exécution du "b & b" et à comparer sa solution (non optimale) à celle de l'heuristique.

---

(1) Les critères sont : l'espace mémoire maximum occupé, le temps CPU dépensé et la déviation de la solution par rapport à l'optimum ("qualité" de la solution).

Les auteurs proposent cinq exemples [1] [2]. L'heuristique et le "b & b" trouvent la solution optimale pour les trois premiers (et petits) [3] exemples. Les auteurs ont dû arrêter l'exécution du "b & b" sur le quatrième exemple (après 25 minutes CPU !). Il apparaît que l'heuristique se trompe d'au moins 10 % sur cet exemple !

Remarquons, en guise de conclusion, que les exemples proposés sont peu nombreux et de taille modérée [4]; ils ne permettent donc pas de conclure quant à l'efficacité éventuelle de l'heuristique (l'exemple 4 est cependant inquiétant).

---

(1) Dont le plus grand est tel que nb_noeuds . nb_fich = 95 et nb_lignes . nb_cap = 170.

(2) Ils n'en décrivent qu'un (partiellement) !

(3) Le plus grand est tel que nb_noeuds . nb_fich = 9 et nb_lignes . nb_cap = 52.

(4) Ce qui est le cas de presque tous les articles, relatifs au FAP et au CCNDP, que nous avons eu l'occasion de consulter !

CHAPITRE II

ETUDE DU PROGRAMME BASE SUR L'ALGORITHME DONNE PAR MAHMOUD 76

II.1. SPECIFICATIONS DU PROGRAMME

II.1.1. DONNEES

1) Topologie.

La topologie du réseau informatique est supposée connue et fixe.

La topologie considérée répond au modèle du paragraphe I.5.1) mais les stations sont intégrées au réseau de communication en ce sens qu'elles peuvent jouer le rôle de noeuds, en plus de leur rôle spécifique.

Le nombre de stations sera noté s et le nombre de noeuds n.

La notion de ligne est donc également généralisée : une ligne relie soit deux stations, soit deux noeuds, soit une station et un noeud, soit un noeud et une station. De plus, la communication entre stations (resp. entre noeuds, entre une station et un noeud, entre un noeud et une station) pourra utiliser, plutôt qu'une ligne physique, un réseau de communication existant (par exemple DCS : réseau public de communication par paquets pour la Belgique). Dans ce cas, on peut considérer que ces stations (resp. noeuds, station et noeud, noeud et station) sont reliées par une ligne virtuelle. Ce choix entre l'utilisation d'une ligne physique (qui sera louée) et l'utilisation d'un réseau existant ne doit pas être fait par l'utilisateur du programme mais est déterminé, pour chaque ligne, par le programme. Une ligne physique sera dorénavant appelée ligne louée. Une ligne virtuelle sera dorénavant appelée ligne non louée.

Il est nécessaire de distinguer :

- ligne unidirectionnelle : une telle ligne ne peut transmettre de l'information que dans un sens déterminé;

- ligne bidirectionnelle simultanée : une telle ligne peut transmettre de l'information dans les deux sens et ce, simultanément.

L'utilisateur du programme aura le choix entre :

- toutes les lignes sont unidirectionnelles et l'existence d'une ligne unidirectionnelle implique l'existence de la ligne unidirectionnelle réciproque (c'-à-d. de sens opposé);

- toutes les lignes sont bidirectionnelles simultanées

Lorsque dans ce chapitre II le terme "ligne" sera utilisé, il désignera, selon le choix signalé ci-dessus, soit une ligne unidirectionnelle, soit un des deux sens d'une ligne bidirectionnelle simultanée. De même, nbl désignera soit le nombre de lignes unidirectionnelles, soit le nombre de lignes bidirectionnelles simultanéesmultiplié par 2.

La figure 2.1. qui suit nous donne un exemple de topologie considérée dans le programme.

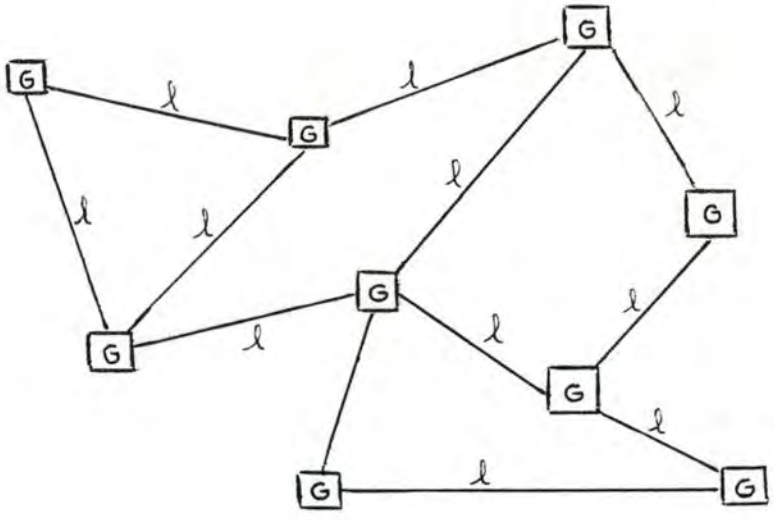


fig 2.1. Exemple de topologie.

Dans cette figure 2.1. :

- G est soit une station, soit un noeud (ce choix est déterminé par l'utilisateur du programme);

- G est soit une ligne louée, soit une ligne non louée (ce choix est déterminé par le programme);

- G est soit une ligne unidirectionnelle, soit une ligne bidirectionnelle simultanée (ce choix est déterminé par l'utilisateur du programme).
  Rappelons que ce dernier choix, contrairement à la location des lignes, est global en ce sens que les lignes sont soit toutes unidirectionnelles, soit toutes bidirectionnelles full-duplex.

L'utilisateur donnera la liste des stations de la topologie qu'il considère et pour chaque station, il donnera :

- son nom (différent pour chaque station);

- sa capacité de stockage (exprimée en bits; si on ne désire pas tenir compte de celle-ci, on peut la considérer comme étant infinie);

- le tarif mensuel pour un bit stocké (exprimé en francs);

- la valeur maximale du nombre de bits par seconde que la station peut émettre (exprimée en bits par seconde; si on ne désire pas tenir compte de cette valeur, on peut la considérer comme étant infinie);

- la valeur maximale du nombre de bits par seconde que la station peut recevoir (exprimée en bits par seconde; si on ne désire pas tenir compte de cette valeur, on peut la considérer comme étant infinie);

- la probabilité que la communication entre la station et le reste du réseau informatique soit coupée.

L'utilisateur donnera la liste de noeuds de la topologie qu'il considère et pour chaque noeud, il donnera :

- son nom (différent pour chaque noeud et différent du nom de chaque station);

- les trois derniers renseignements donnés pour une station où le terme station est remplacé par noeud.

L'utilisateur donnera la liste des lignes de la topologie qu'il considère et pour chaque ligne, il donnera :

- le nom de la station (ou noeud) origine;

- le nom de la station (ou noeud) extrémité;

- sa longueur (en kilomètres).

De plus, il signalera pour chaque ligne :

- si son prix dépend de sa longueur lorsque la ligne est louée (voir paragraphe II.1.1.4) sur les capacités);

- le tarif pour une minute de communication si la ligne est non louée (voir paragraphe II.1.1.4) sur les capacités).

On peut signaler que dans le programme de Mahmoud, seules les stations sont prises en considération et que seuls le tarif mensuel pour un bit stocké et la probabilité de rupture de communication sont considérés. La capacité de stockage et les valeurs maximales de flux entrant et sortant des stations sont supposées infinies. De plus, seule la notion de ligne physique (louée) est considérée et les lignes sont nécessairement unidirectionnelles.

2) Les fichiers.

Un fichier stocké dans une station du réseau est accessible aux abonnés de cette station ainsi qu'à ceux de toutes les autres stations du réseau.

On suppose que nbf fichiers distincts sont utilisés sur le réseau.

Différents exemplaires d'un même fichier peuvent être stockés dans différentes stations du réseau mais il est indispensable qu'au moins une station possède un exemplaire du fichier.

L'utilisateur donnera la liste des fichiers et pour chacun, il donnera :

- son nom (différent pour chaque fichier);

- la longueur du fichier (exprimée en bits);

- la disponibilité souhaitée c'-à-d. la probabilité souhaitée qu'au moins un des exemplaires du fichier soit accessible lorsqu'un accès à ce fichier est demandé par un quelconque abonné. La probabilité effective ne vaudra pas 1 vu que la fiabilité des lignes et des noeuds (ou stations) n'est pas totale (voir à ce sujet le paragraphe II.1.2.3 sur les contraintes de disponibilité).

La matrice x de dimension s × nbf et dont les éléments sont :

- $x_{ij} = 1$ si la station i possède un exemplaire du fichier j;

- $x_{ij} = 0$ sinon ;

sera appelée allocation de fichiers.

3) Les messages.

On appelle message l'information codée que s'échangent les stations. L'accès des abonnés aux fichiers se traduit par un trafic de messages au travers du réseau, trafic dont on peut distinguer quatre composantes :

(i) le "query traffic" : trafic résultant d'une demande de telle station pour tel fichier;

(ii) l'"update traffic"" : trafic de mise-à-jour des exemplaires multiples d'un fichier résultant d'une modification, par un abonné, d'un exemplaire du fichier;

(iii) le "query-return traffic" : trafic de retour (vers la station qui est origine de la demande) correspondant au "query-traffic";

(iv) l'"update-return traffic" : trafic de retour (vers la station qui est origine de la demande) correspondant à l'"update traffic".

Pour chaque paire (station, fichier), l'utilisateur du programme donne le trafic correspondant en nombre de messages par seconde et par type de trafic ((i) à (iv)).

Le trafic de type (iv) a été considéré par Mahmoud mais on peut se demander si en pratique ce trafic n'est pas nul. Il serait lourd d'envoyer un accusé de réception après chaque mise-à-jour. Aussi ce trafic est-il considéré comme nul dans notre programme.

L'utilisateur donnera aussi, pour chaque type de trafic, la longueur moyenne d'un message. On peut signaler que Mahmoud considère que la longueur moyenne est la même quel que soit le type de trafic.

On notera :

- u_{ij} le "query traffic"de la station i pour le fichier j et μ- query l'inverse de la longueur moyenne d'un message de ce type;

- v_{ij} l'"update traffic" de la station i pour le fichier j et μ- update l'inverse de la longueur moyenne d'un message de ce type;

- u'_{ij} le "query-return traffic" de la station i pour le fichier j et μ -ret-query l'inverse de la longueur moyenne d'un message de ce type;

- v'_{ij} l'"update-return traffic" de la station i pour le fichier j et μ -ret-update l'inverse de la longueur moyenne d'un message de ce type.

4) Les capacités et les tarifs de communication.

Les capacités sont exprimées en bits par seconde.

L'utilisateur du programme donnera la liste des capacités dont une ligne louée peut être munie.

Le tarif mensuel pour une ligne louée l munie d'une capacité c, noté tarif (l, loue, c), s'exprime par :

-soit A(c) + B(c) x longueur de la ligne l
où A(c) et B(c) sont des constantes qui dépendent de c;

-soit E(l,c)
où E(l,c) est une constante qui dépend de l et de c.

Cette dernière expression du tarif provient de données recueillies pour l'exécution du cas F.N. (voir annexes) et qui avaient une telle forme.

L'utilisateur donnera la liste des capacités dont une ligne non louée peut être munie.

Le tarif mensuel pour une ligne non louée l munie d'une capacité c, noté tarif (l, non loue, c) s'exprime par :

C(c)
+ prix au segment transmis

x ENTIER-SUP [(flux sur la ligne l/longueur d'un segement) x durée d'un mois]
+ prix à la minute de communication sur la ligne l

x ENTIER-SUP [temps d'utilisation de la ligne l]

où :

- C(c) est une constante qui dépend de c;

- le prix du segment transmis est exprimé en francs par segment;

- ENTIER-SUP (r) désigne le plus petit entier supérieur au réel r;

- le flux sur la ligne l est exprimé en bits par seconde et sera étudié au paragraphe II.1.1.6);

- la longueur d'un segment est exprimée en bits;

- la durée d'un mois est exprimée en secondes;

- le prix à la minute de communication sur la ligne l est exprimée en francs par minute;

- le temps d'utilisation de la ligne l est exprimé en minutes et vaut en première approximation :
  durée d'un mois x (flux sur la ligne l)/c

  où la durée d'un mois est exprimée en minutes.

Pour chaque capacité c disponible pour une ligne louée, l'utilisateur du programme donnera :

- la valeur de la capacité (exprimée en bits par seconde);

- la constante A(c);

- la constante B(c).

Il a déjà été dit (paragraphe II.1.1.1) que l'utilisateur du programme signalera pour chaque ligne, si son prix dépend de sa longueur lorsque la ligne est louée. Si la réponse est positive, l'utilisateur donnera la valeur de E(l,c).

Pour chaque capacité c disponible pour une ligne non louée, l'utilisateur du programme donnera :

- la valeur de la capacité (exprimée en bits par seconde);

- la constante C(c).

L'utilisateur du programme fournira la longueur d'un segment, le prix au segment transmis et le prix à la minute de communication. Rappelons que ce dernier prix peut être différent suivant les lignes. Si le prix de la communication sur une ligne non louée ne dépend pas de la durée de la communication, il suffit de mettre ce prix à zéro.

Le vecteur de dimension nbl dont l'élément i sera le couple (loué ou non loué , valeur de la capacité allouée à la ligne i) est appelé allocation de capacités.

Par abus de langage, on parlera de capacité louée pour une capacité disponible pour une ligne louée et de capacité non louée pour une capacité disponible pour une ligne non louée.

Le nombre de capacités louées sera noté nbcap-loué. Le nombre de capacités non louées sera noté nbcap-non-loué.
La somme de ces deux nombres sera notée nbcap.

5) La procédure de routage.

La procédure de routage est déterministe.

Deux cas peuvent se présenter :

- soit l'utilisateur du programme possède une table de routage : le programme lui permet d'entrer cette table;

- soit l'utilisateur du programme ne possède pas de table de routage : le programme considère alors comme table de routage celle obtenue à partir du principe suivant. Le chemin parcouru par un message pour aller d'une station à une autre est le plus court des chemins reliant ces deux stations.

6) Les options.

6.1) Lignes unidirectionnelles et bidirectionnelles simultanées

Cette option a été envisagée au paragraphe II.1.1.1).

6.2) Le "query traffic".

Le "query traffic" peut être envisagé de deux façons :

- Si un exemplaire du fichier f est stocké à la station s alors un abonné à cette station utilise cet exemplaire, sinon il utilise uniformément tous les exemplaires de ce fichier. Il est peu réaliste d'exiger d'un abonné à une station qu'il utilise uniformément les exemplaires d'un fichier. On peut supposer qu'il préfèrera utiliser toujours le même exemplaire.
- Un abonné à la station s utilise l'exemplaire le plus proche.

Dans le premier cas, le flux (noté λ_l) sur la ligne l vaut :

$$\lambda_l = \sum_{(i,k) \in S(l)} \gamma_{ik}$$

où $S(l) = \{(i,k) \in [1,s] \times [1,s]$: le chemin, déterminé par la table de routage joignant la station s_i à la station s_k, passe par la ligne l $\}$;

γ_{ik} désigne le nombre moyen de messages engendrés par seconde dont la station origine est s_i et la station destination est s_k

$$= \sum_{j=1}^{nbf} (u_{ij} / q_j) * (1-x_{ij}) + v_{ij} * x_{kj} + (u'_{kj} / q_j) * (1-x_{kj}) + v'_{kj} * x_{ij}$$

où q_j désigne le nombre d'exemplaires du fichier j pour $j \in [1, nbf]$.

Dans le second cas, le flux (noté λ_l) sur la ligne l vaut :

$$\lambda_l = \sum_{(i,k) \in S(l)} \gamma_{ik}$$

où S(l) est le même ensemble que ci-dessus;

γ_{ik} a la même signification que ci-dessus mais il a une valeur différente

$$= \sum_{j=1}^{nbf} u_{ij} * z_{ijk} (1-x_{ij}) + v_{ij} * x_{kj} + u'_{kj} * z_{kji} * (1-x_{kj}) + v'_{kj} * x_{ij}$$

où z_{ijk} = 1 si la station k est, parmi les stations contenant un exemplaire du fichier j, la plus proche de la station i (la distance est calculée à partir de la table de routage);

= 0 sinon.

Dans ce second cas, il est donc demandé à l'abonné d'une station d'utiliser toujours le même exemplaire d'un fichier. Si cet exemplaire n'était pas disponible (voir contraintes de disponibilité au paragraphe II.1.2.3)), un autre exemplaire du fichier serait utilisé s'il en existe. Il n'a pas été tenu compte de cette non disponibilité éventuelle dans l'expression du flux donnée ci-dessus. Une amélioration à ce niveau serait souhaitable.

Mahmoud a envisagé la première des deux solutions, par contre dans notre programme l'utilisateur de celui-ci a le choix entre les deux solutions.

6.3) Régime permanent et régime jour-nuit.

L'utilisateur du programme a le choix entre :

- régime permanent :

L'utilisation des fichiers est caractérisée par un seul jeu de matrices de trafic : les matrices de "query traffic", d'"update traffic", de "query-return traffic" et d'"update-return traffic".

- régime jour-nuit :

Dans la pratique on constate que souvent la mise-à-jour des différents exemplaires d'un fichier ne se fait pas immédiatement mais qu'elle se fait la nuit en batch. Il est dès lors intéressant de considérer un premier jeu de matrices de trafic relatif au jour et un second jeu relatif à la nuit.

Seul le régime permanent a été considéré par Mahmoud.

II.1.2 CONTRAINTES.

1) Les contraintes de capacité.

Les contraintes de capacité sont :
pour $l \in [1,nbl]$: la capacité d'une ligne l doit évidemment être strictement supérieure au flux sur cette ligne, le caractère strict de l'inégalité est lié à la formule de délai utilisée (voir paragraphe II.1.2.2)).

2) La contrainte de délai.

Le délai d'un message est le temps total qu'il passe dans le réseau, entre le moment où il est émis par une station et reçu par une autre.

La contrainte de délai est la suivante :

$$\overline{T} \leq T_{max}$$

où T_{max} est une valeur réelle donnée par l'utilisateur du programme;

$\overline{T}$ est le délai moyen de transmission d'un message.

Dans les hypothèses de Kleinrock (voir paragraphe I.5.3)), $\overline{T}$ est donné par :

$$\sum_{i=1}^{nbl} (\lambda_i / \gamma) \quad (1/(\mu C_i - \lambda_i))$$

où λ_i désigne le flux sur la ligne i (exprimé en nombre de messages par seconde);
γ désigne le flux total sur le réseau (exprimé en nombre de messages par seconde).

$$= \sum_{i=1}^{nbl} \lambda_i ;$$

μ désigne l'inverse de la longueur moyenne d'un message;
C_i désigne la capacité dont est munie la ligne louée i.

Comme les longueurs moyennes des messages sont différentes suivant le type de trafic, il est nécessaire de remplacer, dans le terme relatif à la ligne i, μ par μ_i où μ_i désigne l'inverse de la longueur moyenne des messages circulant sur la ligne i.
De la formule évidente qui suit, on tire aisément la valeur de μ_i :

$$\lambda_i / \mu_i = (\lambda - \text{query})_i / \mu - \text{query} + (\lambda - \text{ret-query})_i / \mu - \text{ret-query} + (\lambda - \text{update})_i / \mu - \text{update} + (\lambda - \text{ret-update})_i / \mu - \text{ret-update}$$

où $(\lambda - \text{query})_i$ désigne le flux sur la ligne i provenant du "query traffic" (exprimé en nombre de messages par seconde);

$(\lambda - \text{update})_i$ désigne le flux sur la ligne i provenant de l'"update traffic" (exprimé en nombre de messages par seconde);

$(\lambda - \text{ret-query})_i$ désigne le flux sur la ligne i provenant du "query-return traffic" (exprimé en nombre de messages par seconde);

$(\lambda - \text{ret-update})_i$ désigne le flux sur la ligne i provenant de l'"update-return traffic" (exprimé en nombre de messages par seconde).

De plus certaines lignes pouvant être non louées, le délai moyen de transmission d'un message devient :

$$\sum_{i=1}^{nbl} (\lambda_i / \gamma) \quad (1/(\mu_i C_i - \lambda_i)) + \text{délai-réseau} * \delta_i$$

où délai-réseau est le délai assuré par le réseau pour le transfert d'un message, la valeur de ce délai est donnée par l'utilisateur du programme;

δ_i vaut 1 si la ligne i est non louée;

0 si la ligne i est louée.

Si l'option jour-nuit a été choisie (voir paragraphe II.1.1.6)) alors la contrainte de délai n'est effective que le jour. Dès lors le flux sur les lignes et la longueur moyenne des messages à considérer sont ceux relatifs au matrices de trafic du jour.

3) Les contraintes de disponibilité.

La disponibilité effective d'un fichier f est la probabilité effective qu'au moins un de ses exemplaires soit accessible lorsqu'un accès à ce fichier f est demandé par un quelconque abonné.

Pour le calcul de la disponibilité effective des fichiers, l'utilisateur du programme doit donner, outre la probabilité (notée F_i) que la communication entre une station i et le reste du réseau soit coupée (voir paragraphe II.1.1.1)), la probabilité (notée r_{ik}) de succès d'une communication entre les stations i et k utilisant le chemin, déterminé par la table de routage, reliant la station i à la station k (pour $i,k \in [1,s]$).

La disponibilité effective du fichier j vaut alors (voir paragraphe I.5.2))

$$\left(\sum_{i=1}^{s} w_{ij}\, a_{ij}\right) / \left(\sum_{i=1}^{s} w_{ij}\right)$$

où $w_{ij} = u_{ij} + v_{ij} + u'_{ij} + v'_{ij}$

$$a_{ij} = \left(1 - \sum_{k=1}^{s} (1-r_{ik}\, x_{kj})\right) \quad (1-F_i * (1-x_{ij}))$$

On peut signaler que si, pour le calcul de r_{ik} (pour $i,k \in [1,s]$), on considère un chemin quelconque (et non uniquement celui déterminé par la table de routage) reliant la station i à la station k, alors la disponibilité effective des fichiers sera améliorée.
Mais l'expression du flux telle qu'elle a été donnée au paragraphe II.1.1.6) ne sera plus correcte. Cette amélioration pourrait être envisagée.

4) Les contraintes de stockage.

Les contraintes de stockage sont :
pour $i \in [1,s]$:

$$\sum_{j=1}^{nbf} x_{ij} * \text{longueur du fichier } j * \text{capacité de stockage de la station } i.$$

On peut rappeler que ces contraintes de stockage ne sont pas envisagées par Mahmoud, la capacité de stockage des différentes stations étant considérée comme infinie.

5) Les contraintes de flux.

Les contraintes de flux sont :
- pour $i \in [1,s]$:

$$\sum_{l \in E_i} \lambda_l / \mu_l \leq \text{valeur maximale du nombre de bits que la station i peut émettre.}$$

où $E_i = \{$ ligne l dont la station origine est la station i $\}$;

$$\sum_{l \in E'_i} \lambda_l / \mu_l \leq \text{valeur maximale du nombre de bits que la station i peut recevoir.}$$

où $E'_i = \{$ ligne l dont la station extrémité est la station i $\}$;

- pour $p \in \{1,n\}$:

$\sum_{l \in E_p} \lambda_l / \mu_l \leq$ valeur maximale du nombre de bits que le noeud p peut émettre.

où $E_p = \{$ ligne l dont le noeud origine est le noeud p $\}$;

$\sum_{l \in E'_p} \lambda_l / \mu_l \leq$ valeur maximale du nombre de bits que le noeud p peut recevoir.

où $E'_p = \{$ ligne l dont le noeud extrémité est le noeud p $\}$.

On peut rappeler que ces contraintes de flux ne sont pas envisagées par Mahmoud, les différentes valeurs maximales considérées ici étant supposées infinies.

6) Les contraintes sur l'emplacement des fichiers.

Pour des raisons quelconques (politiques par exemple), il est possible qu'un fichier doive avoir absolument un exemplaire stocké dans une certaine station ou qu'au contraire il ne puisse avoir d'exemplaire à cette station.

Les restrictions sur l'allocation de fichiers doivent être introduites par l'utilisateur du programme.

Mahmoud n'envisage pas de telles contraintes.

II.1.3 ENONCE DU PROBLEME.

Moyennant les données (paragraphe II.1.1.),

déterminer : - l'allocation de fichiers (x)

- et l'allocation de capacités $((c_1, loc_1),\ldots,(c_{nbl}, loc_{nbl}))$

où loc_l $(l \in [1,nbl])$ vaut soit loué, soit non loué suivant que la ligne l est louée ou non

qui minimisent la fonction de coût (notée d) :

$$d = \sum_{l=1}^{nbl} \text{tarif}\ (l, loc_l, c_l)$$

$$+ \sum_{i=1}^{s} \sum_{j=1}^{nbf} (\text{tarif au bit stocké pour la station i} * x_{ij} * \text{longueur du fichier j})$$

sous les contraintes de capacité, de délai, de disponibilité, de stockage, de flux et d'emplacement des fichiers (paragraphe II.1.2.).

II.2 MISE EN OEUVRE DU PROGRAMME.

II.2.1 DEMARCHE GLOBALE.

(1) Première approche : l'approche exhaustive.

Si on note FICH l'ensemble des allocations de fichiers et CAP l'ensemble des allocations de capacités, alors le nombre d'éléments de FICH vaut $2^{(s * nbf)}$ et le nombre d'éléments de CAP vaut $nbcap^{nbl}$.

L'algorithme exhaustif consiste à considérer pour chaque élément de FICH, chaque allocation de capacités de CAP. Le nombre de cas à envisager est par conséquent :

$$2^{(s * nbf)} * nbcap^{nbl}.$$

Pour des valeurs peu élevées telles

s = 6

nbf = 8

nbcap = 4

nbl = 10

ce nombre vaut $2^{(6 * 8)} * 4^{10}$ c'-à-d plus de 10^{20}.

Si on considère qu'un cas peut être traité en 10^{-6} sec, le temps d'exécution du programme exhaustif s'élève à 10^{14} sec c'-à-d à plus d'un million d'années.

Ce calcul montre que l'approche exhaustive ne peut être mise en oeuvre.

(2) Seconde approche : l'approche heuristique.

Quelques définitions préalables :

- une solution du problème est une allocation de fichiers et une allocation de capacités qui vérifient les six contraintes du problème.
- une solution optimale du problème est une solution du problème qui minimise la fonction de coût.
- une allocation de fichiers x est cfaisable (resp.tfaisable) si il existe au moins une allocation de capacités qui, associée à x, vérifie la contrainte de capacité (resp.délai).

- une allocation de fichiers est ffaisable (resp.sfaisable, dfaisable, efaisable) si elle vérifie les contraintes de flux (resp.stockage, disponibilité, emplacement de fichiers).

On peut considérer les deux stratégies suivantes :

La première consiste à :

- considérer un sous-ensemble Y de CAP ;
- pour chaque élément y de Y, considérer un voisinage V_y de y ;
- pour chaque élément y' de V_y, déterminer heuristiquement l'allocation de fichiers qui, associée à y', donne le coût minimum tout en respectant les six contraintes.

La seconde consiste à :

- considérer un sous-ensemble X de FICH ;
- pour chaque élément x de X, considérer un voisinage V_x de x ;
- pour chaque élément x' de V_x, déterminer heuristiquement l'allocation de capacités qui, associée à x', donne le coût minimum tout en respectant les six contraintes.

Dans les deux stratégies, il faut ajouter cette quatrième étape :

- Parmi toutes ces solutions du problème, est retenue celle qui donne le coût minimum : rien ne prouve que cette solution sera optimale. La qualité de cette solution dépend des choix effectués dans les trois premières étapes.

La seconde stratégie a été choisie par Mahmoud, aussi va-t-on se borner à préciser les différentes étapes dans le cas de la seconde stratégie.
Quels algorithmes de choix de X et de V_x (pour $x \in X$), faut-il considérer?
Le but unique est de considérer X et V_x(pour $x \in X$) tels qu'il existe $x \in X$ et $x' \in V_x$: x' est la meilleure allocation de fichiers possible c'-à-d l'allocation de fichiers qui munie de l'allocation de capacités adéquate donne le coût le plus petit possible.
On constate que les choix de X et de V_x (pour $x \in X$) ne peuvent être fait indépendamment les uns des autres. Dans notre démarche, l'algorithme qui détermine X est choisi en fonction du choix de l'algorithme qui détermine V_x (pour $x \in X$). Aussi pour bien comprendre l'algorithme de choix de X faut-il d'abord étudier l'algorithme de choix de V_x, pour x appartenant à FICH, cfaisable, dfaisable et efaisable. De plus ce dernier algorithme dépend de l'algorithme d'allocation de capacités. Aussi est-il souhaitable d'étudier d'abord la troisième étape relative à l'allocation de capacités.

II.2.2. ALGORITHME D'ALLOCATION DE CAPACITES.

Le problème est le suivant :

Etant donné une allocation de fichiers x qui est tfaisable, sfaisable, ffaisable, dfaisable et efaisable :
trouver l'allocation de capacités qui fournira le coût le plus bas tout en respectant la contrainte de temps (les autres contraintes sont indépendantes du choix de l'allocation de capacités).
L'allocation de capacités donnée par l'algorithme sera notée allocation de capacités fournie par la procédure ALLOC-CAPACITES.

La fonction de coût dont on dispose n'est définie que pour les seules allocations de capacités disponibles : cette fonction est donc discrète.

Une première idée consiste à prolonger continûment, dans $\mathbb{R}^{nbl}$, cette fonction de coût et d'utiliser les techniques mathématiques existantes (ainsi la technique des coéfficients de Lagrange) pour trouver le minimum de cette fonction continue dans $\mathbb{R}^{nbl}$ et à partir de ce minimum trouver l'allocation de capacités qui conduit au coût le plus bas.
Cette idée ne peut être appliquée que si une même capacité ne peut être simultanément louée et non louée. Cette condition est évidemment satisfaite si soit nbcap-loue = 0, soit nbcap-non-loue = 0.

Deux questions peuvent se poser :
- Comment prolonger continûment la fonction de coût discrète pour que la résolution mathématique décrite ci-dessus ne nécessite pas un temps CPU trop important. Il faut signaler que du temps CPU sera nécessaire pour déterminer la fonction continue mais aussi et surtout pour résoudre le problème mathématique de recherche du minimum.
- Comment déterminer l'allocation de capacités à partir du minimum de la fonction continue.

Les fonctions continues les plus fréquemment envisagées sont les fonctions linéaires et exponentielles :

- les fonctions linéaires :

minimiser la fonction :

$$f : R^{nbl} \rightarrow R : (x_1,\ldots,x_{nbl}) \rightarrow \sum_{i=1}^{nbl} d_i\, x_i$$

sous la contrainte

$$\sum_{i=1}^{nbl} (\lambda_i / \gamma)\ (1 / (\mu x_i - \lambda_i)) + \text{délai-réseau} * \delta_i \leq T_{max}$$

où . λ_i, γ, μ, T_{max} sont des constantes positives ;

. $\delta_i = 1$ si la ligne i est louée ;
$= 0$ sinon ;

. d_i est une constante telle que :

quelle que soit l'allocation de capacités $((c_1,loc_1),\ldots,(c_{nbl},loc_{nbl}))$

où loc_i ($i \in [1,nbl]$) vaut soit loué, soit non loué suivant que la ligne

i est louée ou non

$$\sum_{i=1}^{nbl} \text{tarif}\ (i,loc_i,c_i) \geq \sum_{i=1}^{nbl} d_i\, c_i$$

l'égalité stricte est en général impossible à obtenir; les coûts de stockage n'ont pas été repris dans la fonction de coût vu qu'ils sont indépendants de l'allocation de capacités.

la solution vaut, dans le cas où toutes les capacités sont louées :

$$x_i = \frac{\lambda_i}{\mu} + \left(\frac{\lambda_i}{\mu\gamma\, T_{max}}\right) \frac{\sum_{j=1}^{nbl} \sqrt{\lambda_j\, d_j}}{\sqrt{\lambda_i\, d_i}}$$

(voir appendice 1 p A93 pour la démonstration)

- les fonctions exponentielles :

minimiser la fonction :

$$f : R^{nbl} \rightarrow R : (x_1,\ldots,x_{nbl}) \rightarrow \sum_{i=1}^{nbl} a_i\, x_i^{\,b_i}$$

sous la contrainte

$$\sum_{i=1}^{nbl} (\lambda_i / \gamma)\ (1 / (\mu x_i - \lambda_i)) + \text{délai-réseau} * \delta_i \leq T_{max}$$

où . λ_i, γ, μ, T_{max} sont des constantes positives ;

. δ_i = 1 si la ligne i est louée ;
= 0 sinon

. a_i et b_i sont des constantes telles que :

quelle que soit l'allocation de capacités $((c_1,loc_1),\ldots,(c_{nbl},loc_{nbl}))$ où loc_i ($i \in [1,nbl]$) vaut soit loué, soit non loué suivant que la ligne i est louée ou non

$$\sum_{i=1}^{nbl} \text{tarif}\ (i,loc_i,c_i) \simeq \sum_{i=1}^{nbl} a_i\ c_i^{\ b_i}$$

l'égalité stricte est en général impossible à obtenir; les coûts de stockage n'ont pas été repris dans la fonction de coût vu qu'ils sont indépendants de l'allocation de capacités.

Dans le cas où toutes les capacités sont louées, voir solution et démonstration en appendice 12 p A95.

On peut signaler que la détermination de la fonction f ne doit se faire qu'une seule fois dans le cas où toutes les capacités sont louées. En effet, dans ce cas, la fonction $\sum_{i=1}^{nbl} \text{tarif}\ (i,loc_i,c_i)$ est indépendante de l'allocation de fichiers et plus précisément du flux. Par contre, la fonction f devrait être déterminée pour chaque allocation de fichiers lorsque certaines capacités sont non louées.

La deuxième question, qui concerne la détermination de l'allocation de capacités à partir du minimum de la fonction continue, présente davantage de difficultés. Ainsi, si les capacités disponibles sont de 1200 (bits/sec), 9600 et 19200, si nbl = 4 et si le minimum de la fonction continue est atteint pour le nbl-uple (15368, 17621.3, 14638, 12621), comment peut-on déterminer la meilleure allocation de capacités à partir de ce nbl-uple. Utilisée seule, cette technique de prolongation de la fonction de coût par une fonction continue semble vouée à l'échec.

L'utilisation d'une prolongée continue peut permettre de réduire fortement l'étude exhaustive des différentes allocations de capacités.
En effet, considérons un parcours exhaustif des différentes allocations de capacités signalées dans l'exemple ci-dessus et notons d-interm le plus petit coût trouvé à un moment donné de ce parcours. Comment réduire ce parcours :

- Pour une ligne donnée, le flux la traversant est connu vu que l'allocation de fichiers x est déterminée. Il est par conséquent inutile de considérer, pour cette ligne, toute capacité inférieure à ce flux. Cette première méthode de réduction suppose donc de classer les capacités disponibles par ordre de valeur croissante.

- La deuxième méthode est expliquée ci-dessous à partir d'un exemple : si on affecte la capacité 19200 à la 2ème ligne et si le minimum de la fonction prolongée, calculé quand la variable associée à la 2ème ligne vaut 19200, est supérieur à d-interm alors il est évidemment inutile de considérer les allocations de capacités suivantes :

* 19200 * *

où * ∈ { 1200, 9600, 19200 }

Cette méthode de réduction repose évidemment sur l'approximation due au fait que la prolongée, en général, est approchée. La validité de la réduction dépend donc de la qualité de l'approximation.

- Pour chaque ligne on peut classer les capacités par coût croissant. Comment réduire le parcours sur base de ce classement. Prenons un exemple : si l'allocation de capacités 9600, 19200, 9600, 1200 fournit un coût supérieur à d-interm et si les classements sont :

  - pour la 1ère ligne : 9600 1200 19200 ;
  - pour les autres lignes : 1200 9600 19200

alors il est évidemment inutile de considérer les allocations de capacités suivantes :

$*_1$ 19200 $*_2$ $*_3$

où $*_1 \in \{ 9600, 1200, 19200 \}$

$*_2 \in \{ 9600, 19200 \}$

$*_3 \in \{ 1200, 9600, 19200 \}$

Dans le cas dégénéré où soit nbcap-loué = 0 ou soit nbcap-non-loué = 0, ce classement coïncide avec l'ordre croissant des valeurs de capacité. Ce n'est pas nécessairement le cas lorsque nbcap-loue ≠ 0 et nbcap-non-loue ≠ 0, car une capacité louée, placée sur une ligne parcourue par un flux faible et peu utilisée, peut être plus coûteuse qu'une capacité non louée de plus grande valeur placée sur cette même ligne.

- Pour chaque ligne, on peut classer les capacités par contribution décroissante de cette paire (ligne, capacité) au délai moyen d'un message. Comment réduire le parcours exhaustif sur base de ce classement. Prenons un exemple : si pour l'allocation de capacités 9600, 19200, 9600, 1200 le délai moyen d'un message est supérieur au délai moyen exigé et si les classements sont, pour chaque ligne, 1200 9600 19200
alors il est évidemment inutile de considérer les allocations de capacités suivantes :

$*_1$ $*_2$ $*_3$ 1200

où $*_1 \in \{1200, 9600\}$

$*_2 \in \{1200, 9600, 19200\}$

$*_3 \in \{1200, 9600\}$

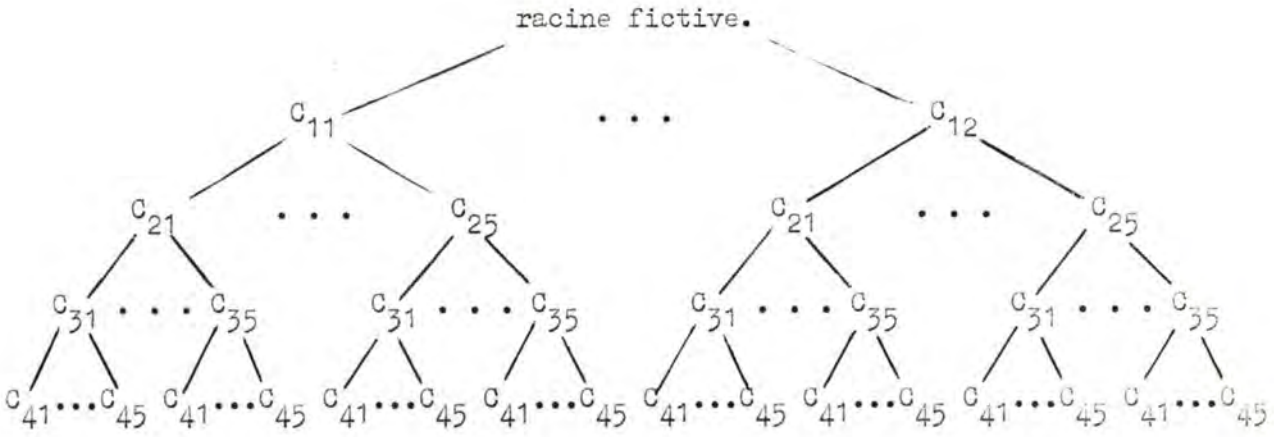
Dans le cas dégénéré où soit nbcap-loué = 0 ou soit nbcap-non-loué = 0, ce classement coïncide avec l'ordre croissant des valeurs de capacité. Ce n'est pas nécessairement le cas lorsque nbcap-loué $\neq$ 0 et nbcap-non-loué $\neq$ 0, car une capacité non louée peut donner une contribution au délai moyen d'un message supérieure à celle donnée par une capacité louée de valeur plus faible.

A propos de ces quatre méthodes de réduction, une question se pose :

Comment mémoriser les allocations de capacités dont la considération est inutile? Il est évidemment exclu d'utiliser des fichiers sous peine de temps d'accès, à ces fichiers, très importants. Mémoriser ces allocations en mémoire centrale exigerait une place mémoire considérable. Une solution consiste à utiliser un algorithme de parcours des allocations de capacités, apte à tenir compte dans son exécution, de ces allocations à ne pas considérer.
Voici cet algorithme :

- Pour chaque ligne l, considérer le classement des capacités $(c_{l1},\ldots,c_{lnbcap})$ selon un des deux classements décrits dans les troisième et quatrième méthodes de réduction et où sont exclues les capacités de valeur inférieure au flux qui parcourt la ligne l.

- Considérer l'ordre schématisé ci-après dans le cas particulier où nbl = 4 et nbcap = 5 :



Cet arbre sera parcouru selon la technique dite RGD si on considère la troisième méthode de réduction (ou RDG pour la quatrième méthode).
Cette technique signifie pour un arbre binaire :

```
PARCOURIR (arbre) : Si arbre ≠ vide
                    alors
                        début-alors
                            visiter la racine ;
                            PARCOURIR (sous-arbre de gauche) ;
                            PARCOURIR (sous-arbre de droite).
                        fin-alors.
```

Dans le cas où l'arbre n'est pas binaire (c'-à-d. où nbcap ≠ 2) les sous-arbres sont parcourus en allant de la gauche vers la droite.
Une allocation de capacités sera représentée par une branche allant de la racine fictive à une feuille de l'arbre. L'ordre de considération des allocations de capacités sera l'ordre dans lequel les feuilles seront visitées lors du parcours de l'arbre ci-dessus.

Considérons le cas de la troisième méthode : il est facile de l'exploiter partiellement. Ainsi, si le noeud C_{23}, correspondant à la branche C_{12}, C_{23} est visité et que le calcul du coût pour l'allocation de capacités :

- C_{12} pour la première ligne ;
- C_{23} pour la deuxième ligne ;
- la première capacité dans le classement relatif à la troisième ligne (à savoir C_{31});
- la première capacité dans le classement relatif à la quatrième ligne (à savoir C_{41});

est supérieur à d-interm
alors il est inutile de poursuivre le parcours du sous-arbre dont la racine est la capacité C_{12} considérée ci-dessus.

L'exploitation de la troisième méthode est partielle en ce sens que, en fait, toute allocation de capacités du type ci-dessous est inutile à considérer :

$*_1 \quad *_2 \quad *_3 \quad *_4$

où $*_1 \geqslant C_{12}$

$*_2 \geqslant C_{24}$

$*_3 \geqslant C_{32}$

$*_4 \geqslant C_{41}$

$\geqslant$ est le signe d'inégalité défini par l'ordre considéré dans le classement des capacités.

Le raisonnement est semblable dans le cas de la quatrième méthode. Quant à la deuxième méthode elle s'adapte aisément à l'algorithme de parcours qu'il soit relatif à la $3^{ème}$ ou à la $4^{ème}$ méthode.
Dans le cas où les classements des méthodes 3 et 4 coïncident (c'est le cas lorsque nbcap-loue = 0 ou nbcap-non-loue = 0) et où par conséquent les deux arbres sont les mêmes, on peut exploiter simultanément les deux méthodes, malgré qu'à priori l'une et l'autre exigent des parcours différents de l'arbre : rappelons que la méthode 3 exige un parcours RGD et la méthode 4, un parcours RDG.
En effet, si on considère un parcours RGD de l'arbre et si le noeud C_{23}, correspondant à la branche C_{12} C_{23}, est visité et que le calcul du délai moyen d'un message pour l'allocation de capacités :

- C_{12} pour la première ligne ;
- C_{23} pour la deuxième ligne ;
- la dernière capacité dans le classement relatif à la troisième ligne (à savoir C_{35});
- la dernière capacité dans le classement relatif à la quatrième ligne (à savoir C_{45})

est supérieur au délai moyen exigé
alors il est inutile de poursuivre le parcours du sous-arbre dont la racine est la capacité C_{23} considérée ci-dessus.

Dans notre programme, les méthodes de réduction 1 et 3 ont été implémentées. Dans le cas où nbcap-loué = 0 ou nbcap-non-loué = 0, l'amélioration signalée ci-dessous a aussi été implémentée.
Enfin, pour exploiter efficacement les méthodes des réductions 2 et 3, il est intéressant de disposer, dès le début du parcours de l'arbre, d'une valeur du d-interm la plus petite possible c'-à-d d'une solution dont le coût sera le plus petit possible et ce, sans trop utiliser de temps CPU.

Deux méthodes sont ici présentées :

1. Utiliser la prolongée continue et à partir du minimum de celle-ci, on pourrait par exemple considérer l'allocation de capacités obtenue en munissant chaque ligne de la capacité qui, parmi les capacités de valeur supérieure à la valeur réelle trouvée pour cette ligne, engendre le coût le plus bas. Encore faudrait-il que cette allocation de capacités vérifie la contrainte de temps.

2. La deuxième méthode consiste à :

   -pour chaque ligne, considérer le classement des capacités par coût croissant;

   -considérer l'allocation de capacités qui, parmi les allocations de capacités vérifiant la contrainte de capacité, engendre le coût le plus bas (pour plus de facilité, cette allocation de capacités sera désignée par l'expression : "allocation de capacités calculée par la procédure INIT-CF-CAPSOL)

   -ensuite

      répéter

         .considérer la ligne qui provoque l'augmentation minimale de coût lorsqu'on lui affecte la capacité qui suit, dans le classement, celle dont elle est munie ;

         .effectuer l'incrémentation de capacité sur cette ligne.

      jusqu'à ce que le délai moyen obtenu soit inférieur au délai moyen exigé.

Une question se pose à propos de cet algorithme :

Cet algorithme ne va-t-il pas boucler indéfiniment?
L'allocation de fichier x considérée est, par hypothèse, tfaisable mais en réalité rien ne prouve que par ce système d'incrementation, l'algorithme aboutira à une allocation de capacités, qui associée à x, vérifiera la contrainte de délai.
Aussi faut-il compléter la condition d'arrêt :
jusqu'à ce que
(le délai moyen obtenu soit inférieur au délai moyen exigé) ou
(les capacités de toutes les lignes sont maximales).

Une amélioration de cette algorithme est présentée ci-dessous. Dans le corps de la boucle de cet algorithme, il est possible qu'une ligne, non retenue par le critère de choix, fournisse, si elle était retenue, une allocation de capacités qui, associée à x, réponde à la contrainte de temps. La meilleure de ces alloca-

tions de capacités sera mémorisée (coût noté d-aux)et finalement comparée à l'allocation de capacités prodiguée par l'algorithme initial : la meilleure sera évidemment conservée.
En fait, la condition d'arrêt de l'algorithme devient :
jusqu'à ce que
(le délai moyen obtenu soit inférieur au délai moyen exigé) ou
(le coût obtenu soit supérieur à d-aux) ou
(les capacités de toutes les lignes soient maximales).

Si l'algorithme ne fournissait aucune solution, on pourrait en trouver une en procédant de la façon suivante : munir chaque ligne de la capacité dont la contribution au délai moyen serait minimale. Cette solution, associée à x, respectera la contrainte de délai car x est tfaisable. Evidemment rien ne prouve que le coût, relatif à cette solution, sera bas.
La méthode 2 est implémentée dans notre programme.

II.2.3 ALGORITHME DU CHOIX DE V_x.

Les allocations de fichiers x considérées sont cfaisables, dfaisables et efaisables. Le voisinage V_x de x sera, par définition, l'ensemble des allocations de fichiers étudiées dans la procédure OPTIMIZING-ROUTINE (x) décrite dans la fig.2.2 ci-dessous.

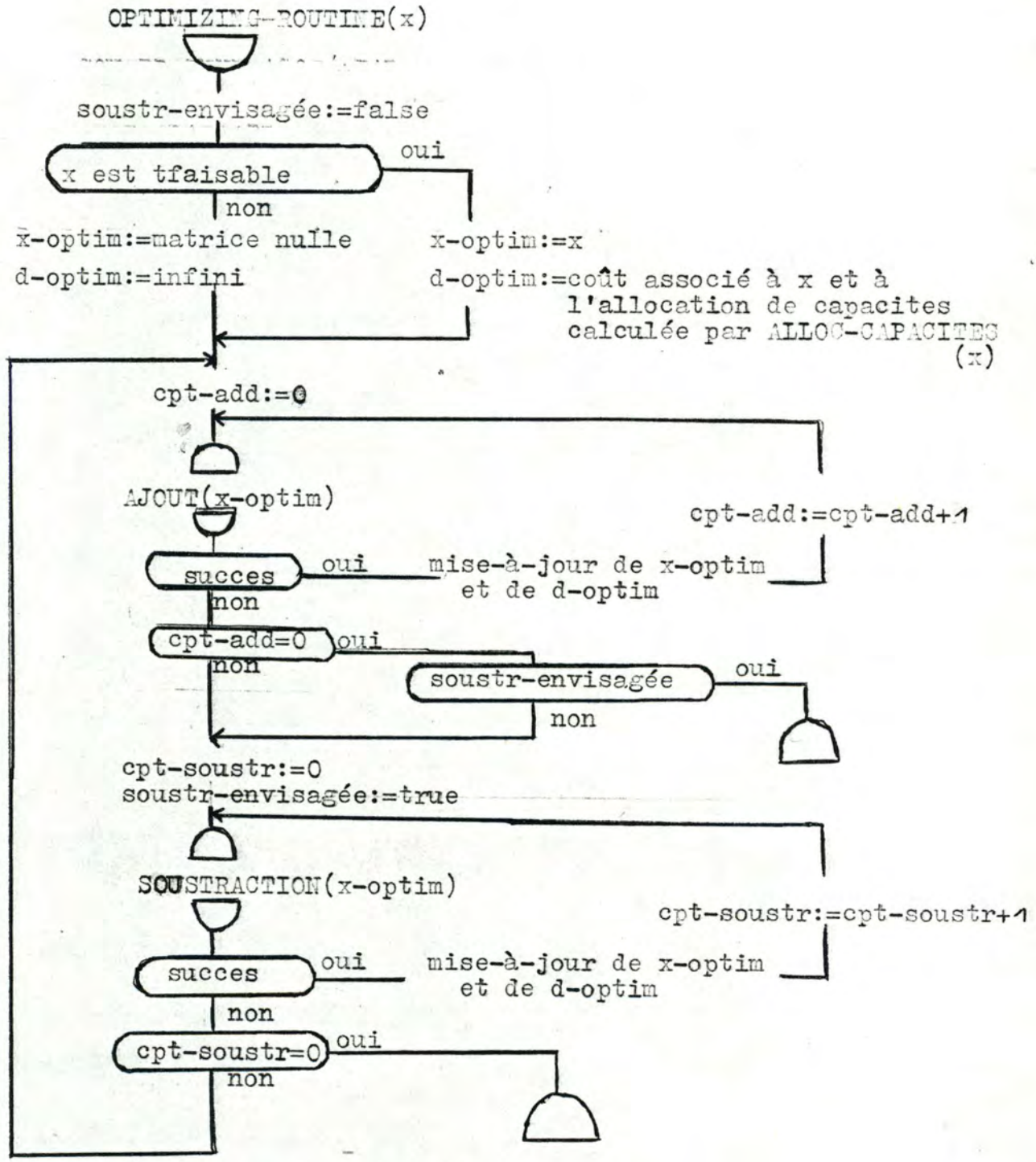


fig. 2.2

Description des procédures et variables évoquées dans la figure 2.2.

1. procédure AJOUT :

argument :

Allocation de fichiers x qui est cfaisable, dfaisable et efaisable.

résultat :

Allocation de fichiers z qui est tfaisable et qui, munie de l'allocation de capacités calculée par ALLOC-CAPACITES fournit un coût le plus bas possible. Il est possible qu'aucune allocation z ne soit trouvée ou que cette allocation de fichiers soit x elle-même, dans ces deux cas on dit que l'ajout a été sans succès (variable succès utilisé dans l'organigramme).

mise en oeuvre :

Notons d-optim (variable d-optim utilisée dans l'organigramme) le coût le plus bas trouvé à un moment donné de l'exécution de OPTIMIZING-ROUTINE. Pour chaque fichier f, est choisie la station qui, non munie d'un exemplaire de f pour l'allocation x et qui munie d'un tel exemplaire, donnera l'allocation de fichiers cfaisable qui fournit le coût minimal lorsqu'elle est munie de l'allocation de capacités calculée par la procédure INIT-OF-CAPSOL.

On dispose donc d'un ensemble E d'au plus nbf exemplaires de fichiers (ou de façon équivalente de nbf couples (station, fichier)), au plus car il est possible que pour certains fichiers il soit impossible d'ajouter un exemplaire ou que l'ajout d'un exemplaire quelconque fournisse une allocation de capacités qui ne soit pas cfaisable.

On étudiera chacun des éléments de l'ensemble des allocations de fichiers obtenues en ajoutant à x les différents sous-ensembles d'exemplaires de fichiers de l'ensemble E.

Voici une amélioration de la mise en oeuvre.

Dans l'étape relative au choix de la station, il est possible que pour un fichier, une station ne soit pas retenue malgré que l'allocation de fichiers qu'elle engendrerait serait tfaisable et fournirait un coût inférieur à d-optim si elle était munie de l'allocation de capacités calculée par la procédure ALLOC-CAPACITES. Les allocations de fichiers, obtenues en ajoutant un exemplaire quelconque à x, sont donc également étudiées.

2. procédure SOUSTRACTION :

Les spécifications de cette procédure sont celles de la procédure AJOUT où l'ajout d'un exemplaire de fichier est remplacé par l'enlèvement d'un exemplaire de fichier.

3 Les variables :

- soustr- envisagée est true
  ssi la procédure SOUSTRACTION a été appelée au moins une fois dans l'exécution de la procédure OPTIMIZING-ROUTINE.

- cpt-add = nombre de fois que la procédure AJOUT a été appelée consécutivement (c'-à-d sans appel intermédiaire à la procédure SOUSTRACTION) et avec succès.

- cpt-soustr = nombre de fois que la procédure SOUSTRACTION a été appelée consécutivement (c'-à-d sans appel intermédiaire à la procédure AJOUT) et avec succès.

- x-optim = meilleure allocation de fichiers tfaisable trouvée à un moment donné de l'exécution de la procédure OPTIMIZING-ROUTINE si une telle allocation a été trouvée ;
  = matrice nulle sinon

- d-optim = coût relatif à x-optim et à l'allocation de capacités calculée par ALLOC-CAPACITES (x-optim) si une meilleure allocation de capacités tfaisable a été trouvée ;
  = l'infini sinon.

II.2.4 ALGORITHME DU CHOIX DE X ET ALGORITHME GENERAL DU PROGRAMME.

Le problème est le suivant :
Tenant compte de l'algorithme général décrit dans la figure 2.3 ci-dessous, comment déterminer l'ensemble X, dont les éléments seront appelés germes.

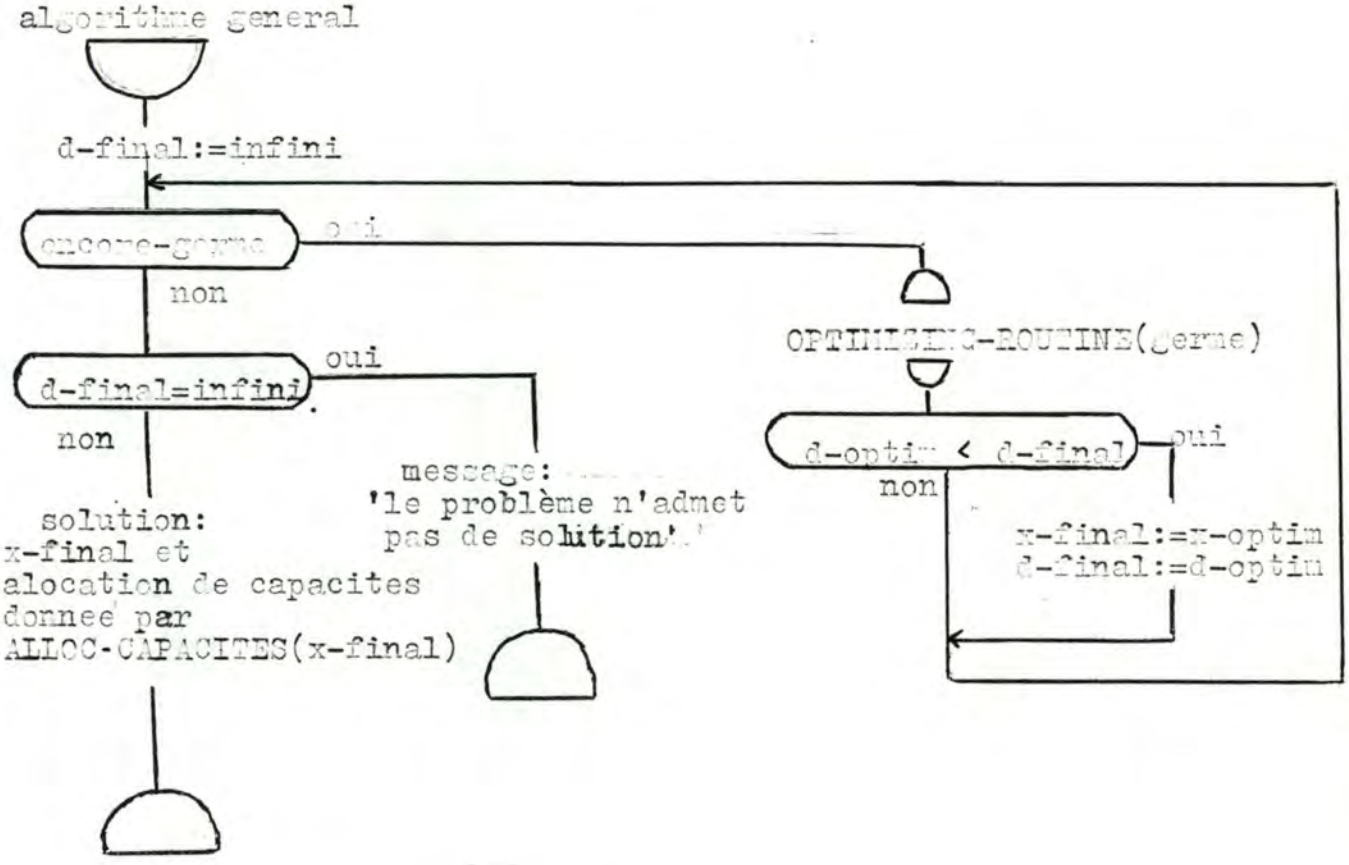


fig. 2.3

Description des variables évoquées dans cette figure 2.3.

- x-final = allocation de fichiers correspondant à la meilleure solution du problème général, trouvée à un moment donné de l'exécution du programme général, si une solution au problème général a été trouvée ;
  = matrice nulle sinon.
- d-final = coût relatif à x-final muni de l'allocation de capacités fournie par ALLOC-CAPACITES si une solution au problème général a été trouvée ;
  = infini sinon.

- encore-germe = true si tous les germes de X n'ont pas été étudiés ;
  = false sinon.

L'algorithme de choix de X se décompose en 3 phases :

- phase 1 :

Détermination, pour chaque fichier f, du nombre minimum d'exemplaires du fichier f nécessaires pour que la contrainte de disponibilité soit respectée pour au moins une allocation de fichiers respectant ce nombre d'exemplaires du fichier f.
Si ce nombre existe pour chaque fichier
alors on obtient un tableau de nbf entiers, appelé moule et noté qtilde,
sinon le problème n'admet pas de solution vu que la contrainte de disponibilité ne serait satisfaite pour aucune allocation de fichiers.

- phase 2 :

Un élément de X sera une allocation de fichiers :
- pour laquelle le nombre d'exemplaires d'un fichier j sera qtilde [j] pour j ∈ [1,nbf] ;
- et qui est cfaisable, dfaisable et efaisable.

- phase 3 :

Trois défauts caractérisent cet ensemble X :
1. X peut être de cardinal petit ou même vide ;
2. Si le moule est caractérisé par des nombres petits d'exemplaires et que la (les) solution optimale du problème est caractérisée par des nombres élevés d'exemplaires, alors l'exécution de la procédure OPTIMIZING-ROUTINE risque d'être longue et coûteuse.
3. Les éléments de X, étant par définition issus du même moule, manquent peut-être de diversité. La notion de diversité est très difficile à cerner. Aussi est-il peut-être intéressant de considérer plusieurs moules différents. Seuls des tests très poussés pourraient infirmer ou confirmer cette affirmation.

Pour essayer de pallier ces trois défauts, l'algorithme prévoit d'incrémenter uniformément les éléments du moule qtilde et d'élargir l'ensemble X suivant la même définition que dans la phase 3.
Précisons cette notion d'incrémentation uniforme.
Par définition, l'incrémentation d'un élément i du moule qtilde est la différence entre la nouvelle valeur et l'ancienne valeur de cet élément. L'incrémentation du moule qtilde vaut par définition la somme des incrémentations de ses éléments.

L'incrémentation uniforme du moule qtilde est basée sur les trois principes :

. l'incrémentation du moule qtilde vaut successivement

1,2,3,4,... jusqu'à $(s * nbf) - \sum_{j=1}^{nbf}$ qtilde [j] initial

- le nombre de contraintes d'emplacement de fichier du type interdiction ;

. pour une incrémentation fixée, l'incrémentation des éléments est uniforme ;
. pour une incrémentation fixée, l'incrémentation des éléments porte d'abord sur les premiers éléments.

Le principe n° 2 est prioritaire par rapport au principe n° 3.
Un exemple aidera à mieux comprendre :

soit s=4, nbf=3 et le moule initial= [1,1,1] .

L'incrémentation uniforme de ce moule fournira sucessivement :

2-1-1 (et non 1-2-1, suivant le principe 3)
1-2-1
1-1-2
2-2-1 (et non 3-1-1, suivant le principe 2)
2-1-2
1-2-2
3-1-1
1-3-1
1-1-3
2-2-2
3-2-1
.
.
.

On peut signaler que Mahmoud ne considérait pas la phase 3.

Cet algorithme de choix de X conduit à un ensemble X dont les éléments sont les éléments de FICH qui sont cfaisables, dfaisables et efaisables. Cet algorithme est donc exhaustif et donc inacceptable.

Deux modifications peuvent être suggérées.

1 Restreindre le nombre de germes étudiés, relatifs à une même valeur du moule qtilde, si ce nombre est élevé. En effet,ces germes sont tous caractérisés par un même nombre total d'exemplaires de fichiers. Cette restriction permet de réduire le défaut 3 signalé précédemment.

Deux questions se posent.
- comment fixer ce nombre?
- quels seront les germes choisis?

Ce nombre doit nécessairement être une fonction (croissante évidemment) de la taille du problème et plus précisément des valeurs de s et de nbf. De plus ce nombre peut être dynamique. La considération des éléments de X relatifs à un même moule, s'arrête après un certain nombre d'échecs successifs quant à l'amélioration de la solution optimale du moment. C'est donc plutôt ce nombre d'échecs successifs qui doit être une fonction (croissante) de la taille du problème. Ce nombre sera noté LIMITE-ECHECS-SUCCESSIFS. Il serait intéressant de choisir les germes de manière a augmenter la diversité des éléments de X (cfr défaut 3 relatif à la diversité). Dans l'implémentation du problème, rien n'est fait dans ce sens. L'algorithme de choix de germe consiste à court-circuiter l'algorithme de parcours des éléments de X dès que le nombre d'échecs successifs vaut LIMITE-ECHECS-SUCCESSIFS. Une amélioration serait nécessaire de ce côté.

2 Permettre à l'utilisateur de stopper l'exécution du programme après l'étude d'un germe quelconque, suivant le temps d'exécution qu'il peut s'offrir.

L'algorithme général devient (figure 2.4) :

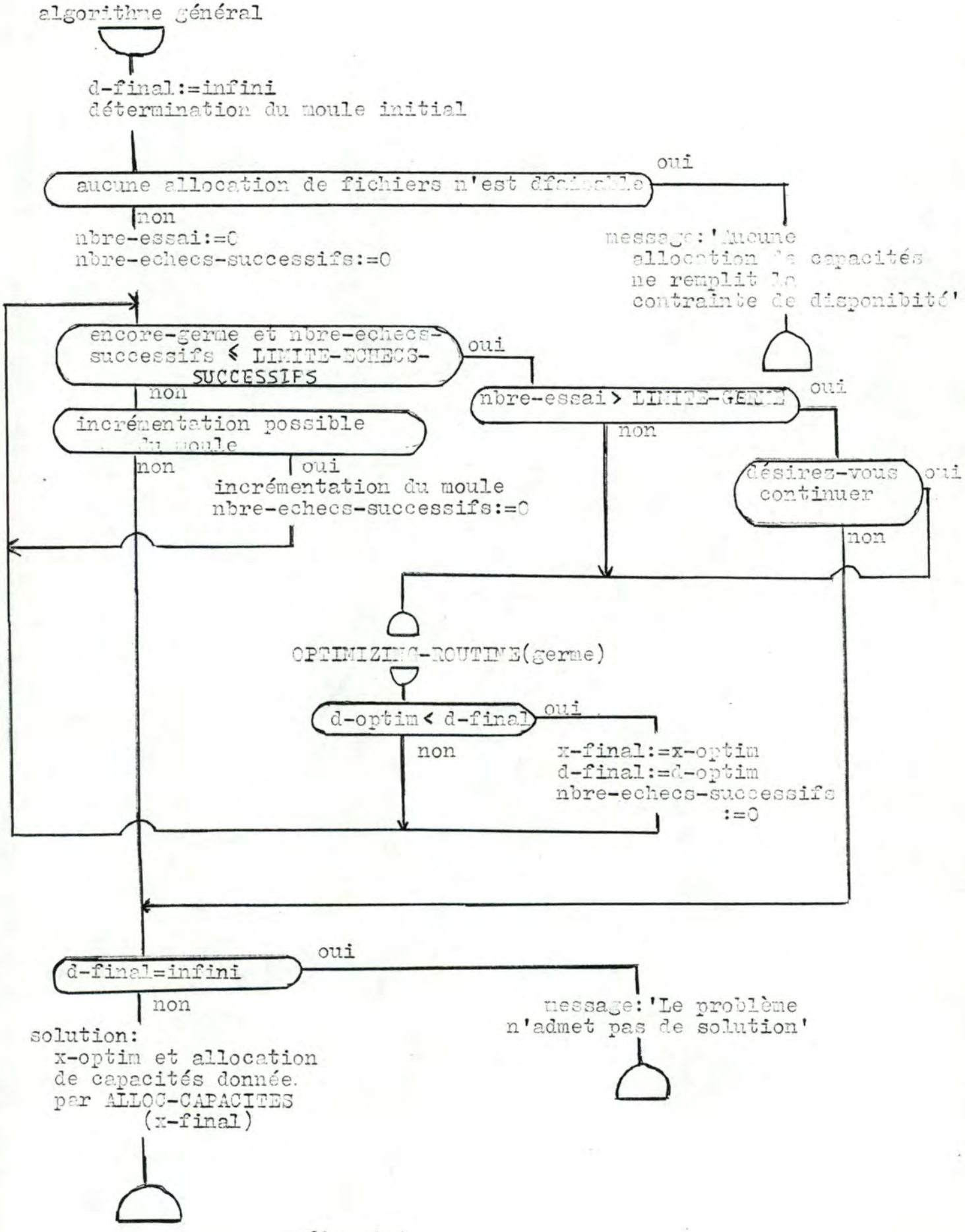

algorithme général
d-final:=infini
détermination du moule initial
aucune allocation de fichiers n'est dfaisable
oui
non
nbre-essai:=0
nbre-echecs-successifs:=0
message:'Aucune allocation de capacités ne remplit la contrainte de disponibité'
encore-germe et nbre-echecs-successifs < LIMITE-ECHECS-SUCCESSIFS
oui
non
nbre-essai > LIMITE-GERM
oui
non
incrémentation possible du moule
non
oui
incrémentation du moule
nbre-echecs-successifs:=0
désirez-vous continuer
oui
non
OPTIMIZING-ROUTINE(germe)
d-optim < d-final
oui
non
x-final:=x-optim
d-final:=d-optim
nbre-echecs-successifs :=0
d-final=infini
oui
non
message:'Le problème n'admet pas de solution'
solution:
x-optim et allocation de capacités donnée par ALLOC-CAPACITES (x-final)

fig. 2.4

Description des variables et constantes utilisées dans la figure 2.4.

- nbre-essai = nombre de germes étudiés.
- nbre-échecs-succ = nombre de germes étudiés successivement pour un moule déterminé sans diminuer la valeur de d-final.
- LIMITE-GERME : l'utilisateur a le choix après un nombre de germes étudiés égal à LIMITE-GERME, de faire apparaître la meilleure solution du moment ou pas, de continuer ou d'arrêter l'exécution du programme.

Une dernière amélioration globale peut être considérée. Elle part de la constatation qu'une même allocation de fichiers peut appartenir au voisinage de germes différents. Aussi est-il intéressant de mémoriser les allocations de fichiers déjà étudiées pour éviter de les étudier à nouveau. Mémoriser toutes ces allocations n'est évidemment pas envisageable (temps d'accès et place si la mémorisation se fait sur fichier, place si la mémorisation se fait en mémoire centrale) :

Deux idées peuvent être suggérées :

1 Mémoriser les germes déjà étudiés. Cette solution n'est pas envisageable à cause de l'impossibilité de retrouver, sans trop de consommation de temps CPU, les éléments du voisinage de ces germes.

2 Mémoriser les allocations de fichiers x qui ont servi d'arguments soit à l'algorithme d'ajout, soit à celui de soustraction. Pour pouvoir retrouver les allocations de fichiers qui ont été étudiées à partir de ces allocations mémorisées, il suffit de mémoriser, plutôt que la matrice x (x a servi d'argument à un algorithme d'ajout), une matrice m dont les éléments m_{ij} ($i \in [1,s]$, $j \in [1,nbf]$) sont :

0 : si $x_{ij} = 0$ et si la station i n'a pas été choisie pour recevoir un exemplaire du fichier j ;

1 : si $x_{ij} = 1$;

2 : si $x_{ij} = 0$ et si la station i a été choisie pour recevoir un exemplaire du fichier j.

Dans le cas où x a servi d'argument à un algorithme de soustraction alors m_{ij} ($i \in [1,s]$, $j \in [1,nbf]$) vaut :

0 : si $x_{ij} = 0$;

1 : si $x_{ij} = 1$ et si la station i n'a pas été choisie pour abandonner son exemplaire du fichier j ;

2 : si $x_{ij} = 1$ et si la station i a été choisie pour abandonner son exemplaire du fichier j.

Il va de soi que deux zones de mémoire sont nécessaires pour séparer les arguments de la procédure d'ajout et ceux de la procédure de soustraction.
On dira qu'une allocation de fichiers est mémorisée dans l'une de ces deux zones, si elle peut être retrouvée à partir d'une des matrices physiquement mémorisées. Ainsi une matrice w peut être retrouvée à partir d'une matrice m mémorisée dans la zone de mémoire relative aux arguments de la procédure d'ajout, si :

$w_{ij} = m_{ij}$ si $m_{ij} = 1$ ou 0 (pour $i \in [1,s]$, $j \in [1,nbf]$) ;

ou

$w_{ij} = m_{ij}$ si $m_{ij} = 1$;

$w_{ij} = 0$ ailleurs sauf pour un couple (i,j) tel que $m_{ij} = 0$.

Vu la taille nécessairement limitée des zones de mémoire accordées à ces arguments, une stratégie doit être utilisée pour remplir ces deux zones de mémoire. Si une des deux zones de mémoire est remplie alors une nouvelle matrice m est mémorisée à la place de la matrice la plus anciennement mémorisée (FIFO). Cette stratégie n'a pas de justification précise si ce n'est peut-être la similarité éventuelle de germes successivement étudiés et relatifs à un même moule. Cette similarité pourrait alors se répercuter sur les allocations de fichiers étudiées à partir de ces germes.

Dans l'implémentation du programme, un germe spécial a été préalablement étudié à tout germe : celui pour lequel toute station est munie d'un exemplaire de chaque fichier, en tenant compte des contraintes d'emplacement de fichiers. L'étude préalable de ce germe peut être intéressante en ce sens que, contrairement à l'approche à partir du moule initial où le nombre d'exemplaires de chaque fichier est minimal (voir phase 1 pour l'algorithme de choix de x), ce nombre sera maximal. Pour pouvoir valablement comparer ces deux approches, les zones de mémoire, relatives aux allocations de fichiers déjà étudiées, sont réinitialisées à zéro à la fin de l'étude de ce germe initial; de plus, beaucoup d'exécutions du programme seraient nécessaires.

CONCLUSION

Nous avons pu constater l'extrême complexité du TAP et du CCNDP; en outre, la complexité de l'interdépendance entre ces deux problèmes semble encore plus évidente (cfr chapitre I). En conséquence, il s'agit de porter tous ses efforts sur le développement d'heuristiques (p.ex. [MAMOUD 76] , cfr chapitre II), sans cependant négliger l'intérêt économique des solutions optimales dans le cas de petits systèmes.

Le programme considéré au chapitre II pourrait être amélioré tant au point de vue de sa mise en oeuvre que de sa spécification. Pour ce qui concerne la mise en oeuvre, différents algorithmes ont été suggérés aux paragraphes II.2.2, II.2.3 et II.2.4. Des tests seraient très intéressants pour comparer ces algorithmes au point de vue temps d'exécution et validité de la solution fournie. Rappelons aussi l'alternative signalée dans le paragraphe II.2.1.(2) : il serait intéressant de développer la première stratégie et de la comparer avec la seconde.
Pour ce qui concerne les spécifications du programme, il serait intéressant que la topologie ne soit pas fixée par l'utilisateur mais que le programme puisse fournir la meilleure topologie possible. Un tel programme (cfr I.5 et I.6) considérerait différentes topologies, et, pour chacune d'elles, appellerait le programme exposé au chapitre II.

BIBLIOGRAPHIE

ABADIE 69[a] ABADIE J., "Une méthode arborescente pour les programmes partiellement discrets," R.I.R.O., 3ème année, Vol. 3, 1969.

ABADIE 69[b] ABADIE, J.,GUIGOU J., "Gradient Réduit Généralisé," note E.D.F. H.I. 069/02, 1969.

AKOKA 80 AKOKA J., "Design of optimal distributed database systems," Distributed Data Bases, C. Delobel and W. Litwin (eds.), North-Holland Publishing Company, INRIA, 1980.

ALCOUFFE 76 ALCOUFFE A., MURATET G., "Optimum location of plants," Manage. Sci.,vol.23,pp.267-274, Nov. 1976.

BAHL 72 BAHL L.R., TANG D.T., "Optimization of concentrator locations in teleprocessing networks", Proceedings of the Symposium on Computer-Communication Networks and Teletraffic, pp.355-362, 1972.

BALAS 76 BALAS E., PADBERG M.W., "Set partitioning: a survey", SIAM Rev. 18, pp.710-760, 1976.

BALL 75 BALL M., VAN SLYKE R., "Backtracking Algorithms for Network Reliability Analysis," Technical Report (1975), printed in Annals of Discrete Mathematics I:Studies in Integer Programming, P.L. Hammer, E.L. Johnson, B.H. Korte, and G.L. Nemhauser, Eds., North-Holland, New-York, pp.49-64, 1977.

BALL 80 BALL M., "Complexity of Network Reliability Computations," Networks, Vol.10, pp.153-165, 1980.

BALL 81 BALL M., MAGAZINE M., "The Design and Analysis of Heuristics", Networks, Vol.11, pp.215-219, 1981.

BASKETT 75 BASKETT F., CHANDY K.,MUNTZ R., PALACIOS F., "Open, closed and mixed networks of queues with different classes of customers," J. ACM. 22, 2, pp.248-260, April 1975.

BUZACOTT 70 BUZACOTT J.A., "Network Approaches to finding the Reliability o Repairable Systems," IEEE trans. reliab., R-19, pp.140-146, 1970.

BUZACOTT 80 BUZACOTT J.A., "A Recursive Algorithm for Finding Reliability Measures Related to the Connection of Nodes in a Graph," Networks, Vol. 10, pp. 11-327, 1980.

BYRNE 68 BYRNE J.L., PROLL L.G., "Algorithm 341. Solution of linear programs in 0-1 variables by implicity enumeration," CACM 11(11), p.782, 1968 .

CANTOR 72 CANTOR D.G., GERLA M., "The Optimal Routing of Messages in a Computer Network via Mathematical Programming," IEEE Computer Conference Proceedings, pp.167-170, sept. 1972.

CANTOR 74 CANTOR D.G., GERLA M., "Capacity Allocation in Distributed Computer Networks", Proceedings of the Seventh Hawaii International Conference on System Sciences, pp.115-117, Jan. 1974.

CASEY 72 CASEY R.J., "Allocation of copies of a file in an information network," Proc. AFIPS SJCC 40, pp.617-625, 1972.

CASEY 73 CASEY R.G., "Design of tree networks for distributed data", NCC 1973, pp. 251-257.

CERI 82[a] CERI S., PELAGATTI G.,"Allocation of Operations in Distributed Database Access," IEEE Trans. on Computers, Vol. C-31, No 2, feb. 1982.

CERI 82[b] CERI S., MARTELLA G., PELAGATTI G.,"Optimal File Allocation in a Computer Network : a Solution Method Based on the Knapsack Problem," Computer Networks 6, pp.345-357, 1982.

CERI 83 CERI S., SHAMKANT N., WIEDERHOLD G., "Distribution Design of Logical Databases Schemas," IEEE Trans. on Soft. Eng., Vol. SE-9, No 4, July 1983, pp.487-503.

CHANDY 72 CHANDY K.M., RUSSELL R.A., "The Design of Multipoint Linkages in a Teleprocessing Tree Network," IEEE Trans. on Comp., Vol. C-21, No 10, Oct. 1972, pp. 1062-1066.

CHANDY 73 CHANDY K.M., LO T., "The capacitated minimum spanning tree", Networks, 3, pp. 173-182, 1973.

CHANDY 76 CHANDY K.M., HEWES J.E., "File allocation in distributed systems", Proc. Intnl. Symp.Comp. Performance Modeling, Measurement and evaluation, pp. 10-13, 1976.

CHEN 80 CHEN P.P.-S., AKOKA J., "Optimal design of distributed information systems," IEEE Trans. Comput. C-29,12, pp. 1068-1080, 1980.

CHOU 73 CHOU W.,KERSHENBAUM A., "A unified Algorithm for designing multidrop teleprocessing networks," Proc. of the Third IEEE Symp. on Data Networks Analysis and Design, pp.148-156, 1973.

CHOU 78 CHOU W., FERRANTE F.,BALAGANGADHAR M., GERKE L., "An Integrated Approach to Optimally Locating Network Access Facilities," article 2234 in ICCC 1978.

CHU 69 CHU W.W., "Optimal File Allocation in a Multiple Computer System," IEEE Trans. on Comp., pp. 885-889, Oct. 1969.

CHU 73 CHU W.W., "Optimal file allocation in a computer network," in Computer-Communication Systems, Abramson and Kuo (eds.), Prentice-Hall, Englewood Cliffs, pp.82-94, 1973.

COFFMAN 80 COFFMAN E.G., GELENBE E., WOOD R.C., "Optimal replication of parallel-read, sequential-write systems," Performance Evaluation Rev., pp.209-216, 1980.

CORNAFION 81 GROUPE CORNAFION,"Systèmes informatiques répartis, concepts et techniques," DUNOD Série Informatique, Phase Spécialité, BORDAS, Paris 1981.

DE BACKER 78 DE BACKER C.R., "A Heuristic Approach to the Optimization of Centralized Communication Networks", article 2231 in ICCC 1978.

DIJKSTRA 59 DIJKSTRA E.W.,"A note on two problems in connection with graphs," Numer. Math., Vol. 1, pp.269-271, 1959.

DOLL 69 DOLL D.R., "The Efficient Allocation of Resources in Centralized Computer-Communication Network Design," Ph. D. Dissertation, The University of Michigan, 1969.

DOLL 71 DOLL D.R., "Topology and transmission rate considerations in the design of centralized computer-communication networks," IEEE Trans. Commun. Technol., Vol. COM-19, pp. 339-344, June 1971.

DOWDY 82 DOWDY L.W., FOSTER D.V., "Comparative Models of the File Assignment Problem," Computing Surveys, Vol. 14, No 2, June 1982.

EFROYMSON 66 EFROYMSON M.A., RAY T.L.,"A branch-and-bound algorithm for plant location," Oper. Res., pp. 361-368, May-June 1966.

ELIAS 74 ELIAS D., FERGUSON M.J., " opological design of multipoint teleprocessing networks," IEEE Trans. Comm., C-22, pp. 1753-1761, 1974.

ELLWEIN 74 ELLWEIN L.B., "A flexible enumeration scheme for zero-one programming," Oper. Res. 22, 2, pp.145-150, Feb. 1974.

ESAU 66 ESAU L.R., WILLIAMS K.C., "A method for approximating the optimal network," IBM Syst. J., Vol. 5, pp.142-147, 1966.

ESWARAN 74 ESWARAN K.P., "Placement of records in a file and file allocation in a computer network," Information Processing 74, IFIPS, 1974.

EVEN 75 EVEN S., "An Algorithm for Determining Wether the Connectivity of a Graph Is at Least k," SIAM J. Comput., vol.4, pp. 393-396, Sept. 1975.

FELDMAN 66 FELDMAN E., LEHNER F.A., RAY T.L., "Warehous location under continuous economies of scale," Manage. Sci., Vol. 12, pp.670-684, May 1966.

FISHER 80 FISHER M.L., HOCHBAUM D.S., "Database locations in computer networks," JACM, Vol. 27, pp.718-735, Oct. 1980.

FLOYD 62 FLOYD R., "Algorithm 97, Shortest Path," CACM, 5, p.345, 1962.

FRANK 70 FRANK H., FRISCH I.T., CHOU W., "Topological Considerations in the Design of the ARPA Network," AFIPS Conf. Proc., 1970 SJCC 36, pp.581-587.

FRANK 71[a] FRANK H., FRISCH I.T., CHOU W., VAN SLYKE R., "Optimal Design of Centralized Computer Networks," Networks, Vol.1, No 1, pp. 43-57, 1971.

FRANK 71[b] FRANK H., FRISCH I.T., "Communication, Transmission, and Transportation Networks," Addison-Wesley, 1971.

FRANK 72[a] FRANK H., CHOU W., "Topological optimization of computer networks," Proc. of IEEE, 60, pp.1385-1397, 1972.

FRANK 72[b] FRANK H., KAHN R.E., KLEINROCK L., "Computer Communication Network Design-Experience with Theory and Practice," AFIPS Conf. Proc.,1972, SJCC, 40,pp. 255-270.

FRATTA 73 FRATTA L., GERLA M., KLEINROCK L., "The Flow Deviation Method - An Approach to Store-and-forward Communication Network Design," Networks,3, pp. 97-133, 1973.

FRAZER 67 FRAZER W.D., "An approximate algorithm for plant location under piecewise linear concave costs," IBM Research Report RC 1875 IBM Watson Research Center Yorktown Heights N Y July 25 1967.

FRIEDMAN 73 FRIEDMAN T., "A system of APL functions to study computer networks," NCC 1973.

GAVISH 80 GAVISH B., "New algorithms for the capacitated minimal directed tree problem," Proc., ICCC 80, pp.996-1000.

GEOFFRION 72 GEOFFRION A.M., MARSTEN R.E., "Integer Programming Algorithms: A Framework and State-of-the-art Survey," Manage. Sci., 18, 465-491, 1972.

GEOFFRION 74 GEOFFRION A.M., "Lagrangian Relaxation and its uses in integer programming," Math. Prog. Study 2, pp. 82-114, 1974.

GERLA 73 GERLA M., "The Design of Store-and-forward (S/F) Networks for Computer Communications," Report UCLA-ENG-7319, 1973.

GERLA 74 GERLA M., FRANK H., ECKL J., "A Cut Saturation Algorithm for Topological Design of Packet Switched Communication Networks," Proc. NTC, pp. 1074-1085, 1974.

GERLA 77 GERLA M., KLEINROCK L., "Topological Design of Distributed Computer Networks," IEEE, Trans. Commun., Vol. COM-25, pp. 48-60, Jan. 1977.

GOMORY 63 GOMORY R., "All-integer integer programming algorithm," Industrial Scheduling, Muth and Thompson (eds.), Englewood Cliffs, N. J.: Prentice-Hall, 1963, pp.195-206.

GOSH 76 GOSH S.P., "Distributing a database with logical association on a computer network for parallel searching," IEEE-SE,2(2), pp. 106-113, 1976.

GRAPA 76 GRAPA E., "Characterization of a distributed database system," Ph. D. Thesis, University of Illinois, 1976.

GRAPA 77 GRAPA E., BELFORD G.G., "Some theorems to aid in solving the file allocation problem," CACM, Vol. 20, No 11, Nov.1977.

HAMMER 77 HAMMER M., "Self Adaptive Automatic Data Base Design," AFIPS, Proc. of the NCC, Vol. 46, pp. 123-129, 1977.

HANSLER 72 HANSLER E., McAULIFFE G.K., WILKOV R.S., "Exact calculation of computer network reliability," FJCC 1972.

HEVNER 79 HEVNER A.R., YAO S.B., "Query processing in distributed database," IEEE Trans. on Soft. Eng., Vol. SE-5, No 3, 1979.

IBM 70 "Communications Network Design Program/360, Service Description manual," IBM Data Processing Division, IBM Tech. Publications Dept., Jan. 1970.

ICCC 78 International Computer Communication Conference in Japan, 1978.

IRANI 82 IRANI K.B., KHABBAZ N.G., "A Methodology for the Design of Communication Networks and the Distribution of Data in Distributed Supercomputer Systems," IEEE Trans. on Computers, Vol. C-31, No 5, May 1982, pp.419-434.

JENSEN 69 JENSEN P.A., BELLMORE M., "An Algorithm to Determine the reliability of a complex system," IEEE Trans. on Reliability, Vol. R-18, pp.169-174, 1969.

KARP 72 KARP R.M., "Reducibility among combinational problems, complexity of computations," Plenum Press, 1972.

KERSHENBAUM 73 KERSHENBAUM A., VAN SLYKE R.M., "Recursive Analysis of Network Reliability," Networks, 3, pp.81-94, 1973.

KERSHENBAUM 75 KERSHENBAUM A., BOORSTYN R., "Centralized Teleprocessing Network Design," Proc. NTC, IEEE, pp. 27.11 to 27.14, Dec. 1975.

KERSHENBAUM 80 KERSHENBAUM A., BOORSTYN R., OPPENHEIM R., "Second order greedy algorithms for centralized teleprocessing network design," IEEE Trans. Comm., 28, pp.1835-1838, 1980.

KERSHENBAUM 83 KERSHENBAUM A., BOORSTYN R., "Centralized Teleprocessing Network Design," Networks, Vol. 13, pp.279-293, 1983.

KIJUNO 81 KIJUNO T. et al., "On a file initial-placement problem in distributed database systems," Hiroshima Univ., Japan, CSG Tech. Res. 81-08, Apr. 1981.

KIM 72 KIM Y.H., CASE K.E., GHARE P.M., "A method for computing complex system reliability," Vol. R-21, May 1972, pp.86-89.

KLEINROCK 64 KLEINROCK L., "Communication Nets;Stochastic Message Flow and Delay," McGraw-Hill(N Y), 1964. Out of Print. Reprinted by Dover Publications, 1972.

KLEINROCK 69 KLEINROCK L., "Models for computer networks," Proc. of the International Commun. Conf., pp.21-9 to 21-16, University of Colorado Boulder, June 1969.

KLEINROCK 70 KLEINROCK L., "Analytic and simulations methods in computer network design," SJCC 1970, pp.569-579.

KLEINROCK 72 (cfr KLEINROCK 64).

KLEINROCK 74 KLEINROCK L., NAYLOR W., "On measured behavior of the ARPA Network," in Proc. 1974 AFIPS NCC, vol.43, pp. 767-780.

KLEINROCK 76 KLEINROCK L., "Queueing Systems - Vol. 2:computer applications," John Wiley and Sons, N Y, 1976.

KLEINROCK 80 KLEINROCK L., KAMOUN F., "Optimal Clustering Structures for hierarchical topological Design of large Computer Networks," Networks, Vol. 10, pp.221-248, 1980.

KLEITMAN 69 KLEITMAN D., "Methods for Investigating the Connectivity of Large Graphs," IEEE Trans Circuit Theory, Vol. CT-16, pp. 232-233, May 1969.

KOLLIAS 80 KOLLIAS J.G., HATZOPOULOS M., "The file allocation problem under dynamic usage," Inform. Systems Vol. 5, pp.197-201, 1980.

KOLLIAS 81[a] KOLLIAS J.G., HATZOPOULOS M., "Criteria to aid in solving the problem of allocating copies of a file in a computer network," The Computer Journal, Vol. 24, No 1, pp. 29-30, 1981.

KOLLIAS 81[b] KOLLIAS J.G., HATZOPOULOS M., "Allocation of copies of s distinct files in an information network," Inform. Systems, Vol.6, No 3, pp.201-204, 1981.

KOMATSU 78 KOMATSU M., NAKANISHI H., SANADA H., TEZUKA Y., "Extended Optimum Channel Capacity Assignment Problems for Message-Switching Networks," in ICCC 1978, travail 2333.

KUEHN 63 KUEHN A.A., HAMBURGER M.J., "A Heuristic Program for Locating Warehouses," Manage. Sci. 10, pp.643-666, 1963.

LABETOULLE 76 LABETOULLE J., PUJOLLE G.A., "Study of queueing networks with deterministic service and applications to computer networks," Proc. Int. Symp. Computer Performance Modeling, Measurement and Evaluation, 1976, ACM, pp. 230-240.

LAND 60 LAND A.H., DOIG A.G., "An automatic method of solving discrete programming problems," Econometrica 28, pp.497-520, 1960.

LANING 83 LANING L.J., LEONARD M.S., "File allocation in a Distributed Computer Communication Network," IEEE Trans. Computers, Vol. C-32, No 3, pp. 232-244, March 1983.

LAVIA 75 LAVIA A., MANNING E.G., "Perturbation Techniques for Topological Optimization of Computer Networks," Proc. Fourth Data Commun. Symp., pp.4-16 to 4-23, 1975.

LAWLER 66 LAWLER E.L., WOOD D.E., "Branch-and-bound methods:a survey," Oper. Res., Vol. 14, No 4, pp.699-719, 1966.

LEVIN 75 LEVIN K.D., MORGAN H.L., "Optimizing distributed databases-a framework for research," Proc. AFIPS 1975 NCC, pp.473-478.

LEVIN 78 LEVIN K.D., MORGAN H.L., "A Dynamic Optimization Model for Distributed Databases," Oper. Res., Vol. 26, No 5, pp.824-835, 1978.

LEVIN 82 LEVIN K.D., "Adaptive structuring of distributed databases," NCC 1982, pp. 691-696.

LODISH 70 LODISH L.M., "Computational Limitations of Dynamic Programming for Warehouse Location," J. Marketing Res., 7, pp. 262-263, 1970

MAC GREGOR 77 MAC GREGOR P., SHEN D., "Network Design:An Algorithm for the Access Facility Location Problem," IEEE Trans. Commun., Vol. COM-25, PP.61-73, Jan. 1977.

MAHMOUD 76 MAHMOUD S., RIORDON J.S., "Optimal allocation of resources in distributed information networks," ACM Trans. Database Syst. 1,1, pp.66-78, March 1976.

MANNE 64 MANNE A.S., "Plant Location under Economies-of-scale - Decentralization and Computation," Manage. Sci. 11, pp.213-235, 1964.

MARTIN 67 MARTIN J., "Design of Real-time Computer Systems," Englewood Cliffs, N.J., Prentice-Hall, 1967.

MARUYAMA 78 MARUYAMA K., "Designing Reliable Packet-switched Communication Networks," travail 2434 in ICCC 1978.

MATSUI 78 MATSUI S., "A Network Optimization by Mixed Integer Programming," travail 2232 in ICCC 1978.

MEISTER 71 MEISTER B., MULLER H., RUDIN H., "New Optimization Criteria for Message-Switching Networks," IEEE Trans. on Commun. Techn., Vol. COM-19, pp.256-260, June 1971.

MISSIKOFF 80 MISSIKOFF M., PAGLI L., "A Note on Optimal Allocation of Files in a Symmetric and Homogeneous Network," Inform. Systems, Vol. 5, pp.149-150, 1980.

MITTEN 70 MITTEN L.G., "Branch-and-bounds methods : general formulation and properties," Oper. Res. 18, pp.24-34, 1970.

MORGAN 77 MORGAN H.L., LEVIN K.D., "Optimal program and data locations in computer networks," CACM 20,5, pp.315-322, May 1977.

MORIN 76 MORIN T.L., MARSTEN R.E., "Branch-and-bound strategies for dynamic programming," Oper. Res., Vol. 24, No 4, pp. 611-627, 1976.

MULLER-MERBACH 81 MULLER-MERBACH H., "Heuristics and their Design: a Survey," European Journal of Operational Research 8, pp.1-23, 1981.

PRICE 78 PRICE P.L., SMITH D.W., "Analysis and use of an integer programming model for optimally allocating files in a multiple computing system," Rep. DTNSRDC 78/102, David Taylor Naval Ship Research and Development Center, Bethesda, Md., Nov. 1978.

RAMAMOORTHY 79[a] RAMAMOORTHY C.V., WAH B.W., "Data management in distributed databases," NCC 1979, pp.667-680.

RAMAMOORTHY 79[b] RAMAMOORTHY C.V., WAH B.W., "The placements of relations on a distributed relational data base," Proc. 1st Int. Conf. Distributed Comput. Syst.,1979.

RAMAMOORTHY 83 RAMAMOORTHY C.V., WAH B.W., "The Isomorphism of Simple File Allocation," IEEE Trans. on Computers, Vol. C-32, No 3, March 1983.

SALKIN 75 SALKIN H.M., "Integer Programming," Addison-Wesley Publishing Company, 1975.

SHARMA 70 SHARMA R.L., EL BARDAI M.T., "Suboptimal communications network synthesis," Proc. Int. Conf. Communications, Vol.7, pp. 19-11 to 19-16, 1970.

SICKLE 77 SICKLE L.V., CHANDY K.M., "Computational complexity of network design algorithms," Information Processing 77, IFIPS, pp.235-239, 1977.

SLOANE 72 SLOANE N.J.A, "On Finding the Paths Through a Network," BSTJ, Vol. 51, No 2, pp.371-390, Feb. 1972.

SPIELBERG 69 SPIELBERG K., "An algorithm for the simple plant location problem with some side conditions," Oper. Res. 17, pp.85-115, 1969.

SPINETO 76 SPINETO R.D., "A Facility Location Problem," SIAM Review 18, pp.294-295, 1976.

SRINIVASAN 80 SRINIVASAN B., "Parallel Searching in Distributed Databases," Computer Networks 4, pp.157-166, 1980.

STEIGLITZ 69 STEIGLITZ K., WEINER P., KLEITMAN D.J., "The Design of Minimum-Cost Survivable Networks," IEEE Trans. Circuit Theory, Vol. CT-16, pp.455-460, Nov. 1969.

STONE 77 STONE H.S., "Multi-processor Scheduling with the Aid of Network Flows," IEEE Trans. on Soft. Eng., Vol. SE-3,No 1, pp. 85-93, Jan. 1977.

TANENBAUM 81 TANENBAUM A.S., "Computer Networks," Prentice-Hall Inc. Englewood-Cliffs, 1981.

TANG 74 TANG D.T., WOO L.S., BAHL L.R., "Optimization of TP networks with concentrators and multiconnected terminals," IBM Research Report, 1974.

WAH 84 WAH B.W., "File Placement on Distributed Computer Systems," IEEE Computer, Jan. 1984, pp.23-32.

WHITNEY 70 WHITNEY V.K.M., "A Study of Optimal File Assignment and Communication Network Configuration," Ph. D. Dissertation, Univ. of Michigan, 1970.

WHITNEY 72 WHITNEY V.K.M., "Comparison of Network Topology Optimization Algorithms," Proc. ICCC, Oct. 1972.

WILKOV 70 WILKOV R.S., "Construction of Maximally Reliable Communication Networks with Minimum Transmission Delay," Proc. of 1970 IEEE Int. Conf. on Commun., Vol.6, pp.4210-4215, 1970.

WONG 77 WONG E., "Restructuring Dispersed Data from SDD-1:a System for Distributed Data Bases," Comp. Corp. of America Tech. Rep. CCA-77-03, 1977.

WONG 78[a] WONG J.W., "Distribution of End-to-End Delay in Message-Switched Networks," Computer Networks 2, pp.44-49, 1978.

WONG 78[b] WONG J.W., "Queueing Network Modeling of Computer Communication Networks," Computing Surveys, Vol.10, No 3, Sept.1978.

WOO 73 WOO L., TANG D.T., "Optimization of Teleprocessing Networks," Proc. of 1973 National Telecommunications Conf., Vol.2, 37C1-C5, No 26-28, 1973.

YAGED 71 YAGED B., "Minimum cost routing for static network problems," Networks, Vol. 1, pp.139-172, 1971.

YOSNIMURA 78 YOSNIMURA T., "An Algorithm for Designing Multidrop Teleprocessing Networks," travail 2433 in ICCC 1978.

Appendice 1

Exemples de travaux relatifs au problème de l'analyse de fiabilité

(références principales : [TANENBAUM 81], [HÄNSLER 72] et [BUZACOTT 80])

1) [TANENBAUM 81] procède à une analyse de connectivité, l'idée étant qu'un réseau fiable devrait rester connexe même si certains noeuds ou arcs sont enlevés. Considérons un graphe donné; l'auteur se pose la question de savoir s'il peut supporter la perte de k noeuds tout en restant connexe [1]. Il décrit deux algorithmes (cfr [KLEITMAN 69] et [EVEN 75]) répondant à la question [2]. Il fait remarquer que les deux algorithmes sont utiles essentiellement dans l'analyse du "worst-case behavior" d'un réseau.

2) [HÄNSLER 72] propose un algorithme d'analyse de fiabilité qui se base sur l'énumération de "cuts".

Considérons un réseau de n noeuds et m lignes. Supposons que la probabilité de panne d'une ligne (resp. d'un noeud) quelconque vaille p (resp. q). Rappelons qu'un "(prime) **s-t link** (resp. node) **cut**" (où s et t sont deux noeuds quelconques) est un ensemble (minimal) d'arcs (resp. de noeuds) dont la suppression disconnecterait s de t (i.e. détruirait tous les chemins entre s et t). Notons P_c [s, t], la probabilité d'une communication réussie entre deux noeuds s et t, en fonctionnement; notons également P_f [s, t] = 1 - P_c [s, t].

---

(1) On peut démontrer qu'un graphe, qui peut supporter la perte de k noeuds tout en restant connexe, peut aussi supporter la perte de de k arcs et rester connexe.

Les auteurs donnent l'expression approchée suivante : (1)

$$P_c\,[s,\,t] = \sum_{i=0}^{m} A^{1}_{s,\,t}\,(i)\,(1-p)^{i}\;p^{m-i}$$

(si $p \gg q$), où $A^{1}_{s,\,t}\,(i)$ désigne le nombre de combinaisons de i arcs telles que, dans le cas où seuls ces arcs sont en état de fonctionnement, il existe au moins un chemin entre s et t.
De même,

$$P_f\,[s,\,t] = \sum_{i=0}^{m} C^{1}_{s,\,t}\,(i)\,p^{i}\,(1-p)^{m-i}$$

(si $p \gg q$), où $C^{1}_{s,\,t}\,(i)$ désigne le nombre de "s - t link cuts" de taille i (dans le cas $q \gg p$, il suffit de remplacer $C^{1}_{s,\,t}\,(i)$ par $C^{n}_{s,\,t}\,(i)$, le nombre de "s-t node cuts" de taille i).

Le problème consiste donc à énumérer tous les chemins (il y en a de l'ordre de 2^{m-n+2}) ou tous les cuts (il y en a de l'ordre de 2^{n-2}) entre deux noeuds quelconques !

---

(1) Ils utilisent $P_f = \mathrm{Max}_{(s,\,t)}\;P_f\,[s,\,t]$, comme indication de la probabilité d'interruption de service du réseau; à p et q fixés, P_f ne dépend que de la topologie du réseau.

Les auteurs font remarquer que :

(i) si le degré des noeuds du réseau est supérieur à 4, alors

$$2^{m-n+2} > 2^{n-2};$$

(ii) le nombre de "prime s-t cuts" est habituellement bien inférieur au nombre de "s-t cuts".

Ils proposent, dès lors, l'expression alternative suivante :

$$P_f[s, t] = P\left[\bigcup_{i=1}^{N} \underline{C}^i_{s,t}\right]$$

où N est le nombre de "prime s-t cuts" et $\underline{C}^i_{s,t}$ = toutes les lignes du $i^{\underline{me}}$ "prime s-t cut" tombent en panne .
Les événements $\underline{C}^i_{s,t}$ ne sont pas mutuellement exclusifs, ainsi :

$$P_f[s, t] \simeq \sum_{i=0}^{N} P[\underline{C}^i_{s,t}]$$

(expression approchée).

Il existe des procédures qui évaluent cette expression approchée par énumération des "prime s-t cuts" (cfr [JENSEN 69]). Les auteurs en proposent une évaluant (sans trop de calculs supplémentaires) exactement $P_f[s, t]$. Les exigences de stockage de leur procédure croissent linéairement avec le nombre d'arcs, mais le temps CPU, quant à lui, croît toujours exponentiellement avec la taille du réseau ! (cfr également [BALL 75]).

Il existe également des méthodes basées sur l'énumération des chemins (cfr [KIM 72] , [SLOANE 72] , [BUZACOTT 70] ... etc).

3) [BUZACOTT 80] étudie le problème suivant :

étant donné un réseau non orienté dans lequel seuls les arcs peuvent tomber en panne (indépendamment l'un de l'autre),

calculer la probabilité Q qu'une combinaison d'arcs tombe en panne telle que le réseau ne soit plus connexe. (1)

Il propose un algorithme récursif calculant Q, dont la complexité est d'ordre O (3^{n-1}), où n est le nombre de noeuds du réseau, celui-ci étant supposé complet ("fully-connected"). Cet algorithme peut calculer également d'autres mesures de fiabilité relatives à la connection des noeuds (sans beaucoup de calculs supplémentaires) :

(i) la probabilité qu'un sous-ensemble de noeuds donné soit disconnecté;

(ii) la probabilité qu'une paire de noeuds soit connectée;

(iii) le nombre moyen de noeuds connectés à un noeud donné;

(iv) le nombre moyen de paires de noeuds connectés;

(v) le facteur d'importance d'un arc (2).

---

(1) En d'autres termes, l'ensemble des arcs en état de fonctionnement ne peut inclure un "spanning tree" (cfr définition en I.5.4) ou encore, l'ensemble des arcs en panne doit contenir un "cut".

(2) Si M est la mesure de fiabilité du réseau, alors le facteur d'importance I (ij) de l'arc ij est donné par

$$I(ij) = \frac{M}{q(ij)}$$

où q (ij) est la probabilité de panne de l'arc ij.

L'algorithme peut aussi calculer (avec une complexité de l'ordre de $O(n^3)$ à $O(n^4)$ au plus) le nombre moyen de noeuds en fonctionnement connectés lorsqu'on suppose que ces derniers peuvent également tomber en panne.

BUZACOTT décrit **divers types d'algorithmes** précédemment proposés pour résoudre ce problème. L'intérêt de cette description est qu'elle met bien en évidence la complexité de ces algorithmes et, plus fondamentalement, l'irréductible complexité de ce problème.

Appendice 2

Le modèle de Kleinrock pour les réseaux à commutation de messages

Nous décrivons, dans cet appendice, le modèle de réseau ouvert de files d'attente pour réseaux à commutation de messages, proposé et développé par L. Kleinrock (cfr [KLEINROCK 64]).

Considérons un réseau de M lignes et N noeuds. Chacune des M lignes est représentée par une file d'attente à serveur unique. La discipline à chaque file est FCFS. On suppose que toutes les lignes sont sans erreur et que les capacités de stockage aux noeuds sont illimitées. Les arrivées de messages (en provenance des stations) sont supposées poissonniennes de moyenne γ_{jk} (en messages par seconde) pour les messages de classe (j, k) ($j, k = 1, 2, \ldots, N$) (1). Les longueurs de messages ont même distribution exponentielle (quelle que soit la classe) de moyenne $\frac{1}{\mu}$ (en bits). La capacité de la ligne i est notée C_i (en bps). Ainsi, la distribution du temps de service (de tous les messages) sur la ligne i est exponentielle de moyenne $1/\mu C_i$ (2). Le routage est supposé fixe (cfr I.5.1) : ainsi, il existe un unique chemin au travers du réseau pour les messages d'une classe donnée. (3)

---

(1) Le taux total d'arrivées de messages dans le réseau (à partir des stations) est noté $\gamma = \sum_{j=1}^{N} \sum_{k=1}^{N} \gamma_{jk}$ (en messages par seconde). Il s'agit du "throughput" du réseau.

(2) μC_i (en messages par seconde) est le taux de service sur la ligne i.

(3) On pourrait aussi supposer le routage aléatoire (cfr p. 61), ce qui introduirait un degré de liberté utile dans certains cas.

Le trafic sur la ligne i est noté λ_i (nombre moyen de messages entrant sur la ligne i par seconde). Le taux total d'arrivée des messages sur les lignes du réseau (trafic interne total) est noté $\lambda = \sum_{i=1}^{M} \lambda_i$ (en messages par seconde). (1) Kleinrock a été amené à formuler également l'hypothèse suivante (dite "Independence Assumption") (cfr [KLEINROCK 64]) : chaque fois qu'un message entre dans un noeud, on lui choisit une nouvelle longueur à partir d'une distribution exponentielle de moyenne $1/\mu$ (2). En effet, lorsqu'un message est routé d'un noeud à l'autre, sa longueur demeure inchangée. Cela implique que les temps de service de ce message aux lignes successives sont statistiquement dépendants; l'analyse de files d'attente en devenait alors complètement intraitable. Cette hypothèse permet de considérer la ligne i comme étant une file M/M/1 de taux d'arrivée λ_i et de taux de service μC_i (i = 1,..., M), indépendamment des autres lignes.

---

(1) La difficulté principale de l'approche par les files d'attente est le calcul des λ_i. Dans le cas d'un réseau en arbre, ce n'est pas un problème puisqu'il n'y a qu'un chemin unique entre chaque paire de points. Cependant, pour un design distribué, l'analyse de file d'attente peut seulement être appliquée en conjonction avec la statégie de routage.

(2) Kleinrock a montré, par simulation, (cfr [KLEINROCK 64]), que l'effet de cette hypothèse sur le délai moyen est négligeable dans le cas de réseaux de connectivité suffisante.

Adoptons les notations suivantes :

(i) T = délai moyen d'un message;

(ii) Z_{jk} = délai moyen d'un message de classe (j,k)
$(j,k = 1,\ldots, N)$;

(iii) π_{jk} = chemin pris par les messages de classe (j,k)
$(j,k = 1,\ldots, N)$;

(iv) $C_i \in \pi_{jk}$ signifie que la ligne i est incluse dans le chemin π_{jk} $(i = 1,\ldots, M$ et $j,k = 1,\ldots, N)$

(v) T_i,... cfr page 72

Il est évident que $T = \sum_{j=1}^{N} \sum_{k=1}^{N} \gamma_{jk}/\gamma \cdot Z_{jk}$ et que $Z_{jk} = \sum_{i\,:\,C_i \in \pi_{jk}} T_i$. Dès lors, (1) $T = \sum_{i=1}^{M} \frac{T_i}{\gamma} \sum_{j,k\,:\,C_i \in \pi_{jk}} \gamma_{jk}$.

Or $\lambda_i = \sum_{j,k\,:\,C_i \in \pi_{jk}} \gamma_{jk}$, d'où :

$$T = \sum_{i=1}^{M} \frac{\lambda_i}{\gamma} \cdot T_i .$$

---

(1) Permutation de signes sommatoires.

Appendice 3

Quelques heuristiques relatives à la conception des réseaux centralisés

Nous décrivons ci-dessous le principe des heuristiques suivantes :

(i) la procédure d'Esau-Williams (cfr [ESAU 66]),

(ii) l'heuristique de Frank, Frisch, Chou et Van Seyke (cfr [FRANK 71]),

(iii) la méthode d'approximation de Vogel (VAM)

et (iv) l'heuristique de Yosnimura (cfr [YOSNIMURA 78]).

(i) L'idée générale de la **procédure d'Esau-Williams** est la suivante. Elle commence avec un réseau en étoile, chaque terminal étant directement connecté au centre (éventuellement par plusieurs lignes !). (1) Elle met à part les lignes initiales chargées au maximum. A chaque étape, une ligne directement reliée au centre est supprimée et une nouvelle connection ajoutée (2) de façon à réduire le coût tout en veillant à toujours satisfaire les exigences de trafic et à maintenir la connectivité du réseau (il y a donc création de liaisons multipoints).

---

(1) Les auteurs supposent qu'une seule capacité de ligne est disponible.

(2) Exemple de reconnection :

Diverses expériences (cfr [WHITNEY 72] et [CHANDY 72]) ont prouvé le bon comportement de cet algorithme (1) (2).

(ii) **L'heuristique de Frank, Frisch, Chou et Van Slyke** (3) utilise la transformation topologique locale suivante (cfr schéma décrit ci-dessous). Pour un arbre donné,

(i) choisir un noeud i et trouver le noeud i_1 le plus proche de i qui n'est pas encore connecté à i;

(ii) ajouter la ligne (i_1, i) à l'arbre et identifier le circuit ainsi formé;

(iii) supprimer à tour de rôle chaque ligne de ce circuit;

(iv) chaque fois qu'une ligne est supprimée, appliquer un algorithme de sélection des capacités (4)

(v) chaque noeud i est ainsi traité;

(vi) on peut considérer des ajouts de lignes de i à ses d plus proches voisins;

(vii) on peut passer au noeud i suivant, quand on a généré L arbres de coût inférieur.

---

(1) Il fournit des résultats proches de 5 à 10 % de l'optimum.

(2) Une version programmée a été commercialisée (cfr IBM 70).

(3) Bon comportement sur de petits problèmes (dont on peut trouver la solution optimale) ainsi que par rapport au MST lorsqu'on remplace le coût des lignes par leur longueur.

(4) Les mêmes auteurs proposent un algorithme optimal de solution au problème de l'allocation de capacités dans le cas d'un réseau en arbre (cfr I.4.)

(iii) Le principe de l'**algorithme VAM** [1] est le suivant (cfr [CHANDY 72]). Le "compromis d'un terminal" est défini comme étant la différence entre les coûts des deux lignes les moins chères reliant ce terminal à un autre. Le terminal ayant le compromis le plus élevé est relié à son plus proche voisin; on forme alors ce qu'on appelle "un segment". Un segment est traité comme un terminal, le coût d'une ligne entre deux segments étant, par définition, le coût de la ligne la moins chère entre les terminaux d'un segment et les terminaux de l'autre. Lorsque la connection de deux terminaux (segments) viole une contrainte de capacité ou de fiabilité, le coût de cette ligne est supposé infini. Une itération du VAM consiste à déterminer le segment ayant le compromis le plus élevé et à réaliser la connection appropriée. Quand tous les terminaux sont reliés au centre, l'algorithme se termine.

(iv) **La technique utilisée par [YOSHIMURA 78]** est classique :

(i) sélection d'un réseau initial;

(ii) changements locaux afin de créer des améliorations;

(iii) obtention d'optima locaux;

(iv) répétition de la procédure jusqu'à épuisement du budget;

(v) la solution globale est alors la meilleure des solutions optimales locales.

L'auteur suggère de choisir des solutions initiales dans lesquelles les terminaux, proches l'un de l'autre initialement, sont groupés ensembles.

---

(1) [CHANDY 72] compare la procédure d'Esau-Williams, l'algorithme de Martin et le VAM entre eux et par rapport aux solutions optimales.

La complexité de la recherche d'un optimum local est de l'ordre de $O(n^{1,7})$, où n désigne le nombre de terminaux. L'auteur prétend améliorer la procédure d'Esau-Williams (ofr [ESAU 66]) de 5 à 10 % (on sait que celle-ci fournit des résultats proches de 5 à 10 % de l'optimum). De plus, la procédure proposée permet d'éviter les cas dégénérés (bien connus) dans lesquels la procédure d'Esau-Williams peut se tromper jusqu'à 50 % par rapport à l'optimum.

Appendice 4

Une approche suivie à propos du design d'ARPANET

(référence principale : [FRANK 70])

Nous décrivons ci-dessous l'approche suivie par [FRANK 70] lors du design d'ARPANET. [FRANK 70] propose une approche de solution au **problème de conception** suivant :

étant donné une configuration de réseau (noeuds et lignes) existante, une estimation du trafic entre noeuds, une borne supérieure sur le délai moyen des messages et une borne inférieure sur le degré de connectivité du réseau,

trouver une topologie de coût minimum, satisfaisant aux contraintes de délai et de connectivité.

Le **but** des auteurs est de développer une méthode permettant de traiter des problèmes de taille réaliste en un temps de calcul raisonnable et fournissant des "solutions faisables" (i.e. satisfaisant les contraintes) dont le coût serait proche de l'optimum.

La **philosophie générale** de la méthode est la suivante. Une **procédure de départ** (PD) génère des solutions faisables. Une **procédure d'optimisation** (PO) essaye de modifier, par des transformations topologiques locales, une solution faisable qu'on lui présente en une nouvelle solution faisable dont le coût serait moindre.

A chaque solution faisable générée par PD, on applique PO jusqu'à ce qu'on ne parvienne plus à réduire le coût (la solution faisable ainsi obtenue est appelée "optimum local").

A chaque solution faisable de départ correspond donc un optimum local. On suppose que certains optima locaux sont proches de l'optimum global. Une **transformation topologique locale** consiste à identifier un ensemble de lignes, à les enlever et à ajouter au réseau un nouvel ensemble de lignes. La qualité de la méthode dépend fortement de celle des transformations locales choisies.

En reprenant les notations de Kleinrock (cfr I.5.3), voici l'expression du **délai moyen** de transmission d'un message retenue par les auteurs :

$$T = \sum_{i=1}^{M} \frac{\lambda_i}{\gamma} T_i$$

et $T_i = \frac{1}{\mu C_i} \left[1 + \lambda_i \frac{(1 + \mu^2 \sigma^2)}{2 (\mu C_i - \lambda_i)} \right]$

où $\frac{1}{\mu}$ est la moyenne des longueurs des messages et σ^2 la variance (formule de Pollaczek-Khinchin).

Les auteurs désirent que le réseau soit conçu de telle manière qu'il puisse supporter un trafic égal entre chaque paire de noeuds. Ainsi, ils supposent que $\gamma_{jk} = 500 \cdot n$ bps, pour tout (j, k) ($j \neq k$ et $j, k = 1, 2, \ldots, N$), où $n \in \mathbb{R}$. Ils ajoutent 500 n bps (trafic de base) à certains γ_{jk} (noeuds dont les exigences en trafic sont plus fortes). Le trafic de base est utilisé afin de déterminer les flux sur chaque ligne et les capacités des lignes (cfr ci-après). Le nombre n est alors augmenté jusqu'à ce que T dépasse une valeur fixée à l'avance (p.ex. 0,2 sec. pour ARPANET).

Les auteurs proposent un algorithme qui détermine une **procédure de routage fixe**, à partir d'une topologie et d'une matrice de trafic données, en se fondant sur l'hypothèse que le chemin le plus favorable entre deux noeuds quelconques est celui qui contient le moins de noeuds intermédiaires (l'algorithme utilise essentiellement une procédure d'étiquetage classique, cfr [FRANK 71^b] ou encore [TANENBAUM 81]).

Ils choisissent d'allouer les **capacités** aux lignes après avoir déterminé le routage du trafic (en prenant systématiquement, pour chaque ligne, l'option la moins chère satisfaisant l'exigence de flux sur cette ligne).

Appendice 5

Une approche de solution au problème de la conception d'un réseau distribué

(référence principale : [TANENBAUM 81])

[TANENBAUM 81] suit l'approche utilisée lors du design d'ARPANET (cfr appendice 4).

Les éléments clés de cette approche sont :

(i) le choix des topologies initiales (faisables);

(ii) la génération de réseaux légèrement modifiés à partir d'un réseau donné (au moyen d'heuristiques dites "de perturbation");

(iii) l'allocation de flux (routage) et de capacités.

L'auteur suggère d'exécuter le programme d'optimisation pour diverses valeurs du délai maximum admissible, afin de dresser une courbe coût-délai (compromis). Il suggère de dresser également une courbe coût-throughput. L'auteur rappelle qu'une amélioration de quelques pourcents peut se traduire par un gain de plusieurs milliers de dollars en coût de location de lignes. L'auteur décrit une heuristique, pour le choix des topologies initiales, proposée par [STEIGLITZ 69] : cette heuristique produit des topologies dont le degré de chaque noeud est supérieur ou égal à k (condition **nécessaire** de k connectivité) (1).

---

(1) Le concours des heuristiques de perturbation sera donc nécessaire afin d'assurer la k connectivité du réseau (contrainte de fiabilité choisie).

Cette heuristique ajoute des lignes (en se basant sur une numérotation aléatoire des noeuds) [1] jusqu'à ce que(k - degré (n)) ≤ 0 , pour tout noeud n.

[MARUYAMA 78] étudie le design des réseaux à commutation de paquets, en insistant plus particulièrement sur l'aspect fiabilité. L'auteur retient les variables de design suivantes : la topologie, le routage, les capacités des lignes, les capacités des noeuds, la discipline de priorité; le but est de minimiser le coût du réseau, sous les contraintes de throughput, de délai **et** de fiabilité. Il propose un algorithme heuristique de solution à ce problème. Il prétend avoir observé que l'utilisation, comme topologie initiale, d'un réseau arborescent répondant à un ensemble de contraintes de design, conduit souvent à un réseau moins coûteux que celui obtenu à partir d'un MST (cfr I.4.) , alors même que la même séquence de procédures d'optimisation locale leur est appliquée. Malheureusement, il ne définit pas clairement cette topologie initiale en arbre.

TANENBAUM précise le problème du routage : il ne peut être dynamique lors du design, alors qu'il le sera sûrement lors du fonctionnement du réseau ! On le supposera donc statique lors du design : on pourra ainsi déduire λ_i (et donc T) de γ_{ij} (cfr notations de Kleinrock, I.5.3)). L'hypothèse qui justifie cette démarche est que le comportement d'un routage dynamique, à l'équilibre, peut être approché suffisamment finement par celui d'un routage statique. Une façon directe d'allouer le flux est d'utiliser le routage dit "du chemin le plus court". Une bonne procédure de routage doit (cfr [FRANK 72[a]]) :

(i) utiliser pleinement les capacités des lignes;

(ii) être efficace au niveau calcul (utilisée de nombreuses fois lors de la phase de design);

(iii) avoir les mêmes caractéristiques que celle qui sera utilisée lors du fonctionnement du réseau.

---

(1) Cela permet de générer plusieurs topologies initiales différentes.

Il est possible de déterminer une procédure de routage qui minimise le délai moyen d'un message (ou maximise le throughput); cette approche n'est cependant applicable qu'à des réseaux de très petite taille (vu (ii) ci-dessus) (cfr références dans [FRANK 72[a]]). Les mêmes auteurs décrivent une procédure heuristique qui dirige le flux de messages sur les chemins les moins utilisés et ayant le nombre minimum de noeuds intermédiaires. Cette approche est rapide et donne de bons résultats (déviations de 5 à 20 % par rapport à l'optimum). Ils décrivent une troisième procédure de routage (appelée "cut saturation", cfr références dans [FRANK 72[a]]) qui généralise la précédente. En fait, on ne connaît pas de procédures générales allouant les capacités tout en optimisant simultanément le routage. En général, on fait l'un puis l'autre (ou vice-versa).

TANENBAUM alloue les capacités en utilisant les résultats de Kleinrock (cfr [KLEINROCK 76]) relatifs au cas continu, et en suggérant l'étude de [GERLA 77] pour le cas discret.

[FRANK 72[a]] décrit deux heuristiques relatives au CCAP, et met en évidence, sur un exemple simple, les limitations de la méthode qui consiste à sélectionner des capacités, parmi un ensemble fini d'options, en se basant sur les approximation continues (cfr [KLEINROCK 76]).

Tanenbaum décrit diverses méthodes de perturbation. Il prétend que certaines études ont montré que le schéma itératif de base n'est pas très sensible à l'heuristique précisément utilisée. Il cite la méthode "branch exchange" dont le principe est le suivant :

(i) choisir deux lignes [1] (quatre noeuds sont ainsi concernés);
(ii) les enlever du réseau;
(iii) ajouter deux nouvelles lignes en utilisant une autre combinaison des quatre noeuds.

[GERLA 74] décrit une heuristique dont il prétend qu'elle donne de meilleurs résultats (tout en exigeant moins de temps CPU) que le "branch exchange", dans le cas de grands réseaux.

---

(1) Le critère de choix peut être un faible taux d'utilisation, un trafic élevé entre paires de noeuds non connectés ... etc.

Cette heuristique enlève ou ajoute des lignes suivant que la contrainte de délai est ou n'est pas satisfaite. Le choix d'une ligne à enlever se base sur le taux d'utilisation et le coût de la ligne (enlever la ligne "gaspillant" le plus d'argent !). On détermine un "cut" (cfr I.5.2) du réseau (qui divise celui-ci en deux composants) et l'on n'ajoute des lignes qu'entre noeuds adjacents aux noeuds adjaçant au cut et n'appartenant pas au même composant. Ces deux heuristiques ne garantissent pas la k-connectivité du réseau.

[LAVIA 75] propose un algorithme qui préserve la connectivité et le diamètre (1) du réseau : les réseaux perturbés sont donc toujours faisables.

[FRANK 72^a] décrit la méthode de perturbation "branch exchange" et signale que le test de k-connectivité peut être réalisé de façon très efficace (en réduisant fortement les calculs) (cfr références dans [FRANK 72^a]).

---

(1) Le diamètre d'un graphe est la longueur de la plus longue géodésique, une géodésique étant le plus court chemin entre deux noeuds donnés. Le diamètre est donc un moyen de mesurer le délai (cfr la notion de longueur moyenne d'un chemin, en I.5.3 (iii)).

Appendice 6

Recherches parallèles dans les bases de données distribuées (BDD)

(références principales [GOSH 76], [SRINIVASAN 80])

(i) Position du problème

Les auteurs s'intéressent à la distribution, sur un réseau informatique, d'une BD dont les "types de segments" (i.e. les types d'enregistrements logiques) présentent des associations logiques entre eux, et plus particulièrement, à l'exploitation, lors de cette distribution, des possibilités de recherche en parallèle qu'offre le réseau lui-même, afin de **réduire le temps de réponse** aux "queries" caractérisés par plusieurs types de segments-cibles. Une telle recherche soulève un **"problème topologique"** (comment distribuer les segments de la BD afin de rendre possible la recherche parallèle) et un "problème d'adressabilité" (le noeud origine d'un "query" doit être capable d'identifier ou de localiser les segments requis par celui-ci). Les auteurs se sont orientés vers le développement de procédures de solution au "problème topologique", le second point étant traité avec plus de détails dans [GRAPA 76].

(ii) "Complete Parallel Searchability" (CPS)

Un query est dit "completely parallel searchable" (CPS) par rapport à une BDD donnée, si tous les types de segments-cibles (de ce query) peuvent être recherchés en parallèle. Une BD est "CPS distributed" (sans redondance) par rapport à un ensemble de "queries" si chacun d'eux est CPS par rapport à la BD, (chaque occurrence de chaque type de segment n'étant stockée qu'une et une seule fois dans le réseau).

Considérons une BD de n types de segments ($S = \{ s_1, s_2, \ldots, s_n \}$) et m queries ($Q = \{ q_1, q_2, \ldots, q_m \}$), la relation entre les deux étant décrite par une matrice $M = (m_{ij})$ telle que $m_{ij} = 1$ si s_j est un type de segment-cible de q_i (= 0, sinon). Supposons que les occurrences d'un type de segment donné ne sont pas réparties parmi plusieurs noeuds (H). Les auteurs proposent un algorithme (A_1) en $O(n^2 . m)$ qui construit une distribution CPS sans redondance et sans contrainte sur le nombre de noeuds du réseau dont on dispose (cfr [GOSH 76] p.108 et [SRINIVASAN 80] p.158). Il s'agit en fait de partitionner S en groupes tels que chaque groupe contienne au plus un type de segment -cible de chaque "query". Le problème peut consister à trouver une distribution CPS sans redondance exigeant un nombre minimum de noeuds. A ce propos, [SRINIVASAN 80] fait les remarques suivantes :

(i) A_1 peut être utilisé pour vérifier si une partition quelconque de S correspond à une distribution CPS;

(ii) le nombre total de partitions distinctes croît très vite avec n (cfr [GOSH 76] p.109);

(iii) On peut déterminer certaines limites sur le nombre de noeuds nécessaires (cfr [GOSH 76] p.109); il est évident, par exemple, que le nombre de noeuds nécessaires pour une distribution CPS non redondante relative à un ensemble de "queries" donné ne peut être inférieur au nombre maximum de types de segments-cibles d'un même "query".

(iii) Contraintes sur le nombre de noeuds dans une distribution

Habituellement, le nombre de noeuds dans un réseau n'est pas un paramètre sur lequel peut jouer le concepteur de la BDD. Ainsi, il peut être impossible de trouver une distribution CPS non redondante de la BD pour un ensemble de "queries" donné.

Gosh (cfr [GOSH 76] p 110) introduit alors la notion de distribution CPS redondante, l'hypothèse (H) étant maintenue, afin de réduire le nombre de noeuds nécessaires pour assurer la propriété CPS. Il décrit un algorithme (A2) (en $O(n^3 . m)$) qui permet de réduire le nombre de noeuds d'une distribution CPS et fournit une distribution CPS redondante. Il démontre que la limite inférieure citée ci-dessus (dans le cas non redondant) est toujours atteinte, quel que soit l'ensemble de "queries" considéré, si la redondance est permise dans la distribution CPS.

Srinivasan (cfr [SRINIVASAN 80] p 158), quant à lui, repose le problème autrement : "étant donné un ensemble de "queries" Q, dont chacun a besoin de différents types de segments d'un ensemble S, trouver une distribution de S en k noeuds, telle que le nombre de recherches nécessaires pour répondre une fois à tous les "queries" soit minimum".

Il parle de problème PPS ("Partial Parallel Searchability"). Il démontre que ce dernier est NP-complet (en le réduisant au problème de coloration d'un graphe) et en propose deux approches de solution : un algorithme "branch-and-bound" (A3) qui fournit l'optimum global mais s'avère peu applicable au monde réel des problèmes relatifs aux DDDs (NP-complet !) et un algorithme basé sur la technique de "clustering" (A4), plus rapide que le "branch-and-bound", mais fournissant des solutions sous-optimales. Le "branch-and-bound" est basé sur la notion de "mesure d'exclusivité" d'un ensemble de segments, qui représente grossièrement le nombre de recherches supplémentaires qu'occasionnerait le regroupement de ces segments sur un seul noeud par rapport à une situation sans conflit de parallélisme.

Srinivasan présente divers résultats expérimentaux dans lesquels il a fait varier n, m et le nombre d'éléments non nuls de M (sa densité) (cfr (ii) ci-dessus). Il apparaît que le temps d'exécution croît exponentiellement avec n, qu'il reste petit pour des densités $\leqslant 0,4$ ou $\geqslant 0,8$ (les BDs classiques dépassent rarement 0,4 !)... etc. (cfr SRINIVASAN 80 pp 161-163).

Le principe de l'algorithme de clustering (A4) consiste à séparer, l'un de l'autre, les types de segments-cibles d'un même "query" par l'intermédiaire d'une "matrice de lien" $B = (b_{ij})$ où $b_{ij} = p_{ij} / (\text{card} \{ k : p_{ik} \text{ ou } p_{jk} \geqslant p_{ij} \} + 1)$ et p_{ij} = probabilité que les types de segment s_i et s_j ne soient pas réclamés ensemble par un quelconque "query" : s_i et s_j peuvent être groupés dans un même noeud si $b_{ij} > t$, où t est une valeur seuil dont la variation peut modifier le nombre et la nature des groupements de segments. L'auteur suggère de faire varier t de façon à satisfaire un critère prédéfini (par exemple "le nombre de groupes doit être inférieur ou égal à k et la fonction de coût suivante doit être minimisée : (coût d'une copie redondante d'un type de segment) . (nombre de copies redondantes) + (coût d'une recherche supplémentaire requise pour répondre à un query).. (nombre de recherches supplémentaires)"). Sur des exemples où $n \leqslant 14$ et $m \leqslant 20$, l'auteur obtient les résultats suivants :

$$\begin{cases} \dfrac{\text{temps d'exécution de A4}}{\text{temps d'exécution de A3}} \simeq 0{,}1 \\ \\ \dfrac{\text{nombre de recherches supplémentaires pour A4}}{\text{nombre de recherches supplémentaires pour A3}} \simeq 1{,}4 \end{cases}$$

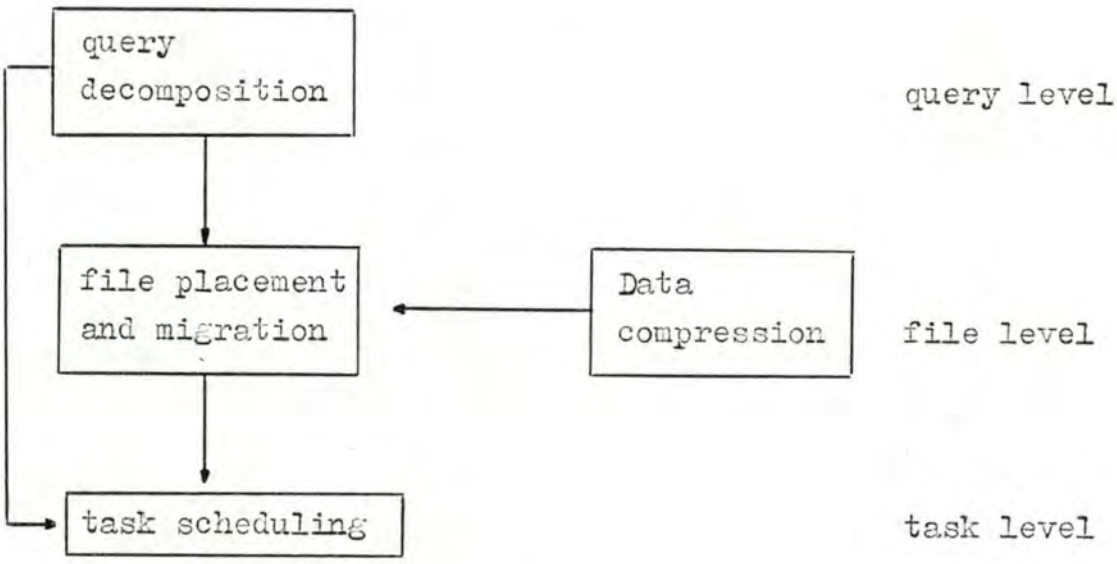
Appendice 7

Une présentation de différents problèmes relatifs à la gestion des données dans un système à BDD

(référence principale : [RAMAMOORTHY 79[a]])

Ramamoorthy et Wah (cfr [RAMAMOORTHY 79[a]]) proposent une classification des questions relatives aux systèmes à BDDs (cfr [RAMAMOORTHY 79[a]]p.668).

La figure suivante montre les relations qui existent entre les différentes questions relatives à la gestion des données. La relation x --> y signifie que la solution de x n'est pas affectée par celle de y, mais non vice versa : la solution de x peut donc être développée indépendamment de celle de y.



-. Relationships among various data management issues .-

Rappelons qu'un "**query**" est une demande d'accès à un ou plusieurs fichiers émanant d'un utilisateur ou d'un programme (les résultats, ou "query return traffic", étant renvoyés, après le traitement du "query", au noeud demandeur).

Le but du "**query decomposition**" est de maximiser le parallélisme dans le traitement et de minimiser la quantité d'informations circulant dans le système.

L'idée est de **décomposer** un "query" accédant à plusieurs fichiers en "sous-queries" n'accédant qu'à un seul fichier, afin d'éviter les transferts préalables des fichiers en un site commun (1), et afin de permettre le traitement en parallèle (de ces "sous-queries") (technique utilisée dans SDD-1, cfr [WONG 77]; cfr également appendice 6). Lorsque le "query" n'est pas décomposable, une solution (proposée par les auteurs) pour éviter des transferts de fichiers consiste à ajouter à ces derniers des informations redondantes.

Les auteurs adoptent le point de vue relationnel. Un "query" défini sur plusieurs relations n'est pas décomposable (en "sous-queries" relatifs à une seule relation) lorsqu'il existe une relation logique (dans le prédicat définissant le "query") définie sur un domaine commun à celles-ci. Il faudrait dès lors que toutes les relations auxquelles il accède soient réunies en un même endroit. Les auteurs proposent une solution alternative consistant à compiler dans ces relations des informations relatives à leurs domaines communs afin d'exécuter le "query" sans transfert préalable de relations.

---

(1) Généralement, le "query return traffic" est bien inférieur au trafic résultant du transfert des fichiers à un emplacement commun.

Cette technique pose plusieurs problèmes :

(i) quantité de stockage supplémentaire;

(ii) mises à jour nécessaires (des informations redondantes) lors de la modification d'un domaine commun d'une relation;

(iii) le concepteur de la BD doit être capable d'estimer la quantité d'informations redondantes à compiler dans les relations.

Cette technique devrait être avantageuse pour les prédicats les plus fréquemment utilisés.

Alors que la quantité d'information utile contenue dans les données des grandes BDs alphanumériques est habituellement assez faible, et que le traitement devient de plus en plus distribué, la **compression des données** apparaît comme étant une solution naturelle permettant de réduire la quantité de données stockées et transférées (sur les lignes de communication). Cependant, l'utilisation des codes de compression semble être en conflit avec celle du codage redondant (par exemple vérification de la parité) qui accroît la fiabilité. Il s'agit donc d'examiner les relations entre taux d'erreur et compression. Les auteurs présentent quelques propriétés désirables des codes de compression, diverses techniques de compression et les directions futures de la recherche dans ce domaine (cfr [RAMAMOORTHY 79^{a}]).

Après avoir décomposé un "query", il faut **ordonnancer le traitement** de ses différents "sub-queries" sur la BDD, pour une distribution de fichiers donnée (définie par le FAP). On parle alors du "Query Scheduling Problem" (QSP). On décompose tout d'abord les "queries" en tâches (une tâche est une simple requête qui utilise une ressource pendant une durée finie). Les tâches sont ordonnancées via un graphe de précédence. Il faut ensuite ordonnancer les "queries" entre eux. On distingue alors le "Sequential QSP" (SQSP) et le "Parallel QSP" (PQSP).

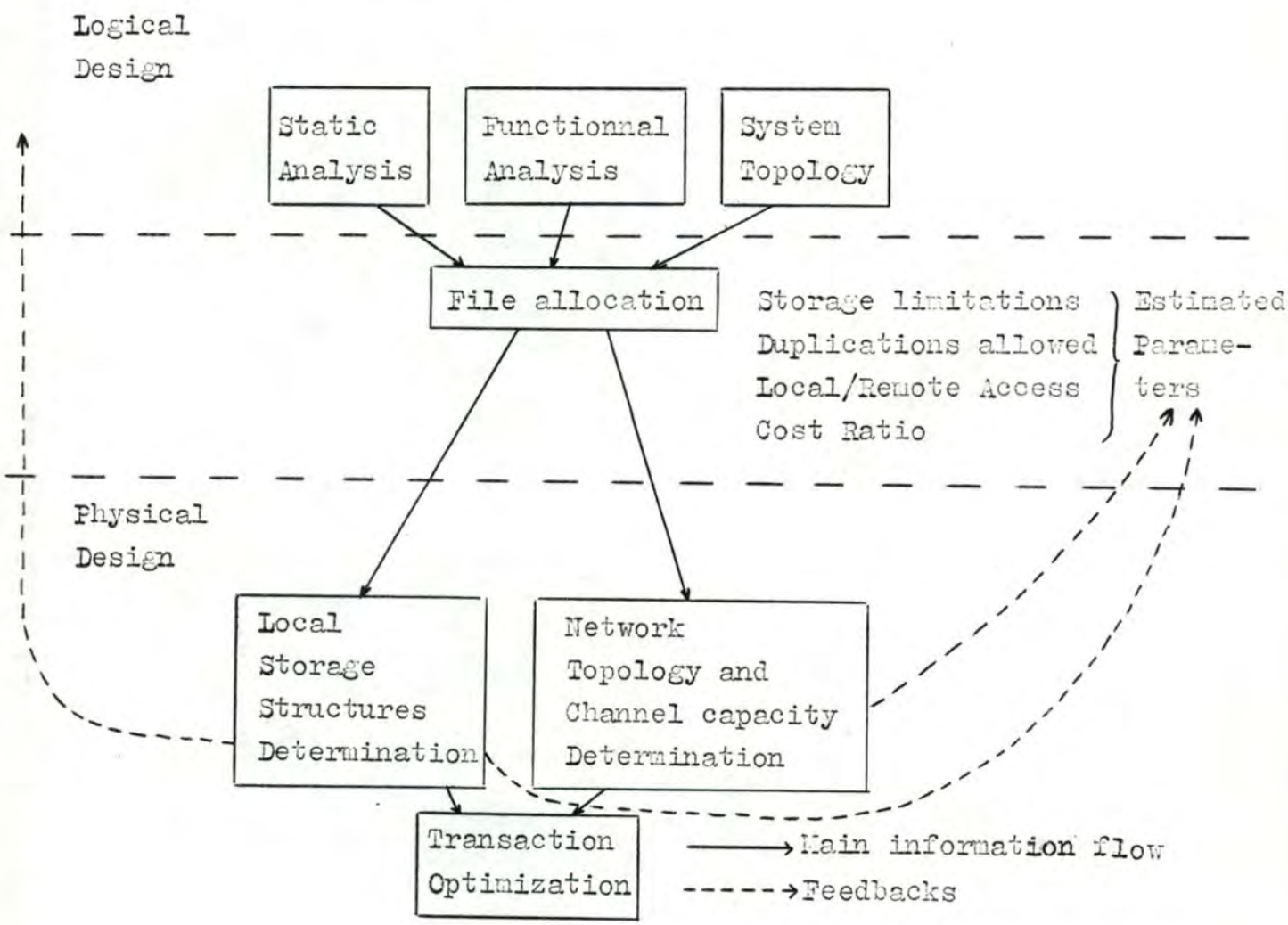
Dans le SQSP, les "sub-queries" sont traités dans un ordre séquentiel; dans le PQSP, ils sont traités en parallèle. Afin de réaliser un compromis entre la quantité de communications et le temps de réponse, une combinaison du SQ processing (qui favorise la première) et du PQ processing (qui favorise le second) peut être réalisée. Le QSP sur une BDD est NP-complet; la recherche devrait donc se diriger vers le développement d'algorithmes d'approximation.

Appendice 8

Deux approches au problème de la conception des systèmes informatiques distribués

(références principales : [CERI 82^b] et [YEH 79])

(i) [CERI 82^b] propose une méthodologie générale de conception d'un système informatique distribué, résumée dans la figure ci-dessous :



-. Phases du design d'une BDD .-

L'analyse statique détermine la structure des fichiers (et leur taille), et l'**analyse fonctionnelle** la logique des transactions, la fréquence de "query" et d'"update" des transactions en fonction des fichiers, ainsi que la fréquence avec laquelle telle transaction est issue de tel noeud (allocation à priori des transactions). Déterminer la **topologie du système** consiste à préciser quels noeuds existeront ensemble (et quelles sont leurs capacités de traitement et de stockage). Les auteurs distinguent deux cas de problèmes d'allocation correspondant à des complexités différentes du software de la BDD, selon que la redondance des copies de fichiers est ou n'est pas permise. L'optimisation de l'allocation des fichiers (en fonction des transactions pré-définies) est basée sur un modèle de minimisation des coûts d'accès dont les valeurs des données d'entrée sont fournies par les trois étapes du design logique et par l'estimation (raffinée par feedback) de paramètres (dépendant de phases postérieures) de stockage et de coûts d'accès.

Remarquons que la phase d'allocation de fichiers sera exécutée pour diverses configurations des paramètres d'entrée (via le feedback) et que les caractéristiques du réseau ne sont pas définies à priori, ni déterminées simultanément avec l'allocation des fichiers. La phase d'**optimisation des transactions** (par exemple analyser et optimiser les "queries" occasionnels) utilise des techniques développées par ([HEVNER 79], [CERI 82[a]] ... etc).

Le **point de vue adopté** par les auteurs est celui de l'optimisation des performances du système considéré comme un tout (par opposition au point de vue de l'utilisateur adopté par Morgan et Levin, cfr p. 32); aussi le **stockage** est-il considéré comme une contrainte et non comme un coût (ajouter une copie de fichier à un noeud ne coûte rien si la capacité de stockage de ce noeud n'est pas dépassée).

L'extrême **simplicité des paramètres de coûts d'accès du modèle** (un coût unitaire pour les accès locaux et un coût unitaire pour les accès éloignés) n'est justifiée que par la méthodologie dans laquelle il s'insère (cfr [CERI 82[b]] p. 348).

Les auteurs démontrent que le **problème d'allocation de fichiers dans le cas non redondant** est isomorphe à un problème classique de recherche opérationnelle (un problème de Knapsack) à propos duquel de nombreux algorithmes spécialisés ont été publiés (techniques spéciales d'énumération, programmation dynamique ... etc, cfr réf. in [CERI 82^{b}]).

Ils proposent un algorithme "branch-and-bound" ("b & b") pour le **cas redondant.** Cet algorithme consiste essentiellement en la décomposition du problème redondant complexe en problèmes de Knapsack simples (avec un nombre limité de variables) que l'on peut résoudre de façon exhaustive ou par une "greedy heuristic" (cfr p.51), le"b & b" étant alors, respectivement, optimal ou heuristique. L'efficacité présumée du "b & b" devrait résider dans le fait que chaque sous-problème est très simple et profite des résultats des sous-problèmes antérieurs, ainsi que dans la stratégie de recherche adoptée (basée sur des critères d'ordre) qui permet d'obtenir de bonnes solutions assez tôt, et ainsi d'améliorer les tests sur les bornes supérieures.

Les auteurs présentent un exemple numérique à 5 noeuds et 8 fichiers (ce qui n'est pas beaucoup !) qui est résolu pour 4 valeurs différentes des capacités de stockage aux noeuds du réseau. Ils soulignent la nette supériorité de leur "b & b" par rapport à l'algorithme de Balas (énumération à but général pour problèmes 0-1, cfr [SALKIN 75]).

(ii) **Yeh et Chandy** (cfr [YEH 79]) se restreignent au problème de conception de systèmes à BDD tels que :

. il n'y a pas de copies multiples de fichiers;

. il y a un haut degré de localisation géographique des transactions;

. la topologie du réseau est un arbre (un centre et des filiales).

Ils justifient ce choix en faisant remarquer que :

. ce problème particulier est rencontré suffisamment souvent en pratique pour mériter qu'on développe à son propos des techniques spécifiques;

. la complexité du problème général interdit le développement d'algorithmes optimaux raisonnablement rapides;

. il est possible, dans ce cas particulier, de prendre en considération simultanément la sélection des processeurs, (pour des sites distribués géographiquement), le choix de la topologie du réseau, la sélection des capacités des lignes de communication et l'allocation des fonctions et des BDs aux sites.

Le but des auteurs est de développer des aides (programmes) à la conception, qui sélectionnent un petit nombre de designs, à partir d'un grand nombre d'alternatives, pour une analyse intensive. Ces programmes aident le concepteur (sans le remplacer !). L'algorithme présenté par les auteurs ne prétend pas fournir le design "optimal" (les facteurs de coût et de performance n'étant pas les seuls à intervenir).

L'équipe de conception spécifie l'ensemble des capacités disponibles pour les lignes (capacités et coûts); elle produit des designs alternatifs pour chaque famille d'équipements éloignés (exemple de famille : I.B.M. 3.790) et spécifie l'ensemble des configurations éloignées alternatives (coût, taille de la mémoire secondaire, puissance de traitement (en transactions traitées par unité de temps, taux de traitement par type de transaction)).

La charge (par type de transaction), présentée au réseau à toutes les filiales, est spécifiée par l'équipe de conception (en unités de transactions par unité de temps). Plusieurs types d'estimations de charge seront pris en considération (charge moyenne journalière aux heures de pointe ...).

Pour chaque ensemble de configurations du système et d'estimations de charge, une courbe "coût (de location)-temps de réponse" est calculée au moyen d'algorithmes proposés par les auteurs. L'équipe de conception sélectionnera un ensemble de points de cette courbe pour une étude plus intensive.

On évalue le petit ensemble de designs sélectionnés par les algorithmes au moyen de simulations et d'autres méthodes afin d'étudier la sensibilité des mesures de performance aux fluctuations de charge, de fiabilité des sites et des lignes ... etc. Il faut ici penser à la croissance future du système. Les concepteurs peuvent apporter leur connaissance de facteurs dont ne tient pas compte l'algorithme employé.

Appendice 9

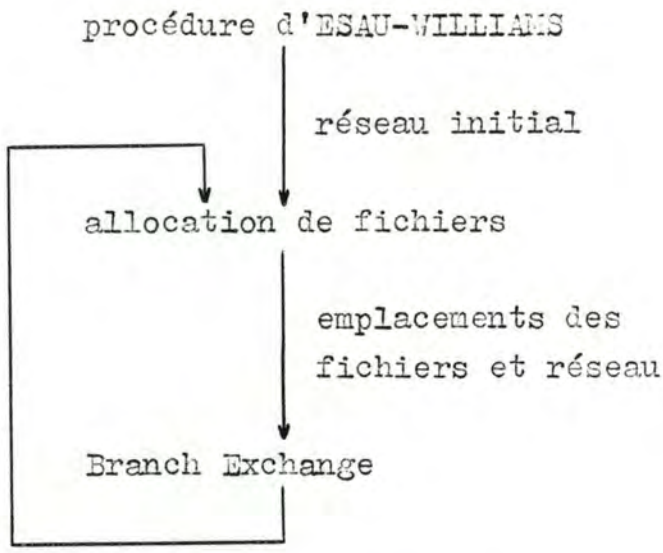
Une description de quelques travaux relatifs à l'étude simultanée du FAP et du CCNDP

Nous présentons brièvement les travaux suivants, relatifs à l'étude simultanée du FAP (cfr p. 1) et du CCNDP (cfr p. 57) : [CASEY 73], [IRANI 82] et [CHEN 80].

(i) **Casey** étudie (cfr [CASEY 73]) simultanément le FAP et le CCNDP. On considère une base de données (BD) partitionnée en fichiers qui peuvent être stockés indépendamment dans le réseau. On suppose que l'on crée un nouveau système à BD distribuée. L'auteur fait remarquer qu'il serait plus réaliste de tenir compte de l'existant, en favorisant plutôt une transition graduelle vers un nouveau système. Il souligne également l'importance des facteurs humains (les compétences, les ambitions, les traditions ... etc) sur les caractéristiques de réseau. Le comportement des types d'accès est supposé statique (cfr I.3.2.) et le réseau de communication de type "store-and-forward"

Etant donné une liste des noeuds utilisateurs, une liste des noeuds de stockage et de traitement potentiels, une liste des fichiers, les taux d'accès (en mode "query" et "update") de chaque utilisateur pour chaque fichier, une liste des capacités de lignes disponibles, les coûts de stockage de chaque fichier en chaque noeud et les coûts de location d'une ligne pour chaque paire de noeuds reliés et pour chaque capacité, **le problème consiste à trouver** une allocation des fichiers aux noeuds de stockage, une topologie de réseau et une allocation de capacités de lignes associée, réalisant le service demandé tout en minimisant le coût total de location des lignes et de stockage des fichiers.

Le modèle proposé est un programme mixte non linéaire. Les techniques de la programmation linéaire n'étant pas directement applicables, l'auteur propose une procédure de recherche heuristique ayant la forme indiquée dans la figure suivante :



-. processus de design .-

Remarquons que #(l'espace des solutions) $\geq 2^{kn} \cdot c^{C_n^2}$, où n (resp k, c) désigne le nombre de noeuds (resp de fichiers, de capacités); ainsi l'auteur s'est-il limité aux topologies de type **arbre**, dont on peut citer les **avantages** suivants (cfr I.5.4):

(i) les décisions de **routage** se réduisent au choix d'une copie de fichier parmi plusieurs;

(ii) l'effet d'un **changement local** dans une structure d'arbre est facile à calculer;

(iii) les arbres sont **intéressants en pratique** (le réseau à coût minimum pour communiquer avec une BD centralisée est un arbre) (cfr I.5.4).

Il suggère d'utiliser les arbres comme point de départ d'une procédure de design plus générale (cfr [FRIEDMAN 73]).

Casey utilise l'algorithme heuristique d'Esau-Williams (cfr I.5.4 et appendice 3) comme étape initiale, afin de générer un réseau centralisé candidat pour de futures améliorations. (1)

Cet algorithme est efficace et rapide; de plus, cette étape initiale permettra de comparer entre eux les coûts du design centralisé et du design distribué.

L'étape d'allocation des fichiers, sans modifier la topologie du réseau, utilise le modèle développé dans [CASEY 72] pour chaque fichier à tour de rôle (cfr I.1); de nouvelles capacités de lignes sont alors calculées et on tente d'améliorer le coût en essayant diverses perturbations locales dans l'allocation des fichiers. La configuration de moindre coût est retenue.

La procédure "Branch exchange", sans toucher à l'allocation des fichiers, modifie la topologie du réseau, via des transformations locales consistant en des ajouts et suppressions de lignes, jusqu'à ce qu'on ne trouve plus aucune amélioration (cfr [FRANK 71^{a}] ou encore appendice 3).

Casey a décrit complètement un exemple à 18 noeuds, 2 fichiers et 6 capacités.

Le but de cette étude est de montrer qu'il est possible de réduire le coût du réseau en alternant le FAP et le CCNDP, et de servir de point de départ à des recherches futures dans ce domaine.

---

(1) On choisit un noeud auquel on stocke une copie de chaque fichier.

(ii) **Irani et Khabbaz** (cfr [IRANI 82]) étudient le problème de la distribution d'une BD intégrée sur un ensemble de sites géographiquement distants, et de la conception d'un réseau de communication afin d'effectuer les opérations de la BD (i.e. FAP et OCNDP).

Ils font remarquer que les arbres sont très peu fiables; de plus, un nombre minimum de lignes ne correspond pas nécessairement à un coût minimum de communication dans le réseau, car ce coût est aussi fonction des capacités des lignes.

Ils se limitent aux topologies G (n, k, d) introduites dans [WILKOV 70] (n est le nombre de noeuds, k la connectivité et d le diamètre du graphe sous-jacent) qui sont hautement fiables et de diamètre d minimal pour (n, k) donné (ce qui réduit coût et délai de communication).

Lors de la conception d'une BD distribuée, il faut prendre en compte :

. la fiabilité du réseau;

. la disponibilité des fichiers;

. le coût de communication;

. le délai (moyen) de communication pour accéder à un fichier (noté T).

La mesure de disponibilité d'un fichier i (notée A_i) retenue, est la probabilité qu'au moins une copie du fichier se trouve en un noeud accessible à tous les autres noeuds; **la mesure de fiabilité du réseau** (notée R) est la probabilité que le réseau soit connecté (i.e. qu'il existe un chemin entre chaque paire de noeuds).

Le **modèle** consiste à minimiser (sur l'allocation des fichiers et des capacités et la topologie du réseau) les coûts de stockage des fichiers et de location des lignes sous les contraintes $R \geqslant R_o$, $A_i \geqslant A_{i,\,o}$ (pour tout fichier) et $T \leqslant T_o$, où R_o, $A_{i,\,o}$ et T_o sont donnés; via une expression de LB (R) ("Lower Bound" sur R) tirée des propriétés des G (n, k, d), la **contrainte de fiabilité** du réseau peut s'exprimer sous la forme d'une contrainte sur k : $k \geqslant k_m$.

Les **contraintes de disponibilité** sont exprimées par l'intermédiaire de limites sur A_i (trouver une expression exacte de A_i est un problème extrêmement complexe) traduites en limites inférieures sur le nombre de copies (n_i) de chaque fichier : $n_i \geqslant n_{i,\,m}$.

La **contrainte de délai** est exprimée, via les résultats de Kleinrock (cfr [KLEINROCK 76]), en termes des variables d'allocation de fichiers et sous forme d'une contrainte sur chaque ligne.

Le **processus de conception** est le suivant (itération sur k) :

1. Contrainte de fiabilité du réseau : $k_m \leqslant k \leqslant n - 1$;

2. Contraintes de disponibilité des fichiers : $n_{j,\,m} \leqslant n_j \leqslant n$;

3. Choix d'une valeur initiale de k;

4. NTDP (cfr I.5.1);

5. FAP et CCAP; si critère d'optimalité OK, alors FIN, sinon, incrémenter k, modifier les coûts de pose des lignes ainsi que les coûts unitaires de communication entre paires de noeuds et aller en 4.

L'étape 4 consiste à trouver le réseau de coût minimum dont le graphe sous-jacent est un G (n, k, d).

Etant donné la topologie générée en 4, l'étape 5 consiste à trouver l'allocation des fichiers et des capacités de lignes qui minimisent les coûts de communication et de stockage tout en respectant les contraintes de délai sur chaque ligne.

Les auteurs ont développé une heuristique. Ils proposent trois procédures résolvant respectivement le NTDP, le FAP et le CCAP. Le CCAP doit être précédé par NTDP (les lignes doivent être allouées) et le FAP (il faut connaître les flux sur chaque ligne). Le FAP doit être précédé du NTDP (la configuration du réseau doit être connue).

Etant donné n, k et les coûts de position des lignes, la **procédure de NTDP** doit minimiser le coût relatif aux lignes posées (la topologie résultante devant être G (n, k, d)). Elle est constituée d'une procédure qui trouve l'ensemble des topologies G (n, k, d) correspondant à (n, k), d'une "greedy heuristic" (cfr [FISHER 80] p.51) qui génère une allocation de lignes, et d'une procédure qui tente d'améliorer, par changements d'allocations, la solution courante.

La **procédure de FAP** alloue un fichier à la fois. Pour chaque fichier, la procédure itère sur n_i (à partir d'une borne inférieure sur n_i) en utilisant une "greedy heuristic" et une "interchange heuristic" (améliorer la solution par réallocation de copies du fichier, cfr [FISHER 80] p. 51).

La **procédure de CCAP** trouve une allocation optimale de capacités.

Après avoir incrémenté k, on évalue les flux attendus sur les différentes lignes et on obtient, via la procédure de CCAP, de nouvelles valeurs des coûts cités à l'étape 5 ci-dessus.

Les auteurs ont réalisé des tests (satisfaisants) d'optimalité sur les heuristiques citées ci-dessus (uniquement sur des exemples à dimensions réduites !). Ils ont étudié l'influence de la connectivité sur le coût à partir de 4 exemples de 1 à 50 fichiers et de 6 à 10 noeuds. Il se peut qu'on obtienne une meilleure solution en augmentant k ! (plus de lignes utilisées, mais réduction de la capacité moyenne).

(iii) **Chen et Akoka** développent (cfr CHEN 80) un modèle d'optimisation (sous forme d'un programme entier non linéaire) et une procédure de solution optimale (basée sur la méthode de programmation **entière appelée** "bounded branch-and-bound", ainsi que sur les structures **spéciales** du modèle) pour le problème du design des systèmes informatiques distribués. Ils proposent une application **réelle** du modèle au cas d'une grande banque.

Le modèle détermine simultanément l'allocation des lignes de communication (point-à-point et full-duplex), leurs capacités, la capacité des ordinateurs à chaque noeud et l'allocation des programmes et des BDs. La dépendance entre programmes et BDs, introduite par Levin et Morgan (cfr I.3.2), est généralisée (l'idée est que tous les programmes ne peuvent travailler sur toutes les BDs, il existe des relations entre eux). Le flux d'information en retour (d'un "query" ou d'un "update") est pris en compte (en supposant qu'il suit la même route qu'à l'aller !). Le but du modèle est de minimiser le coût total du système (1) tout en respectant certaines contraintes (nécessaires à la faisabilité du système).

La technique de solution utilisée est basée sur la méthode "bounded branch-and-bound" (2) (cfr [ABADIE 69a]).

Les auteurs font remarquer que peu de modèles antérieurs ont été appliqués à des problèmes de design réels, ce qui rend très ardue la tâche d'évaluer leur utilité et leur correction. (3)

---

(1) Coût des ordinateurs, coût des softwares de BDs, coût des lignes de communication, coûts de stockage des BDs et des programmes, coûts de communication (aller et retour) des "queries" et des "updates" (des utilisateurs aux programmes et des programmes aux BDs).

(2) Cfr [AKOKA 80] p.55

(3) Morgan et Levin, par exemple, testent leur modèle sur les données de Casey alors même qu'il est plus général que celui de Casey.

Ils proposent dès lors une application de leur modèle à un cas réaliste (une grande banque) dont cependant les dimensions ne semblent pas excessives (quatre noeuds, une BD et un programme !); de plus, les capacités des lignes de communication sont connues à priori et ne doivent donc pas être déterminées par le "BBB" ! Ils proposent également une analyse de la sensibilité de la solution aux variations de facteurs importants (coûts de communication, taux de "queries" ... etc) afin d'en étudier la robustesse (ils appliquent leur algorithme à quatre jeux de données différents). Ils signalent que la solution optimale peut apporter des économies mensuelles de l'ordre de quelques milliers de dollars, alors qu'une exécution de l'algorithme prend sept secondes (sur IBM 370/168) et coûte six dollars ! Ils font remarquer que leur modèle ne prend pas en considération, entre autre chose, les délais de file d'attente et l'optimisation des "queries".

Appendice 10

Spécifications des procédures et fonctions du programme du chapitre II

La liste des procédures et fonctions est dressée ci-dessous; dans la première colonne, figurent les procédures et fonctions déclarées au niveau global; en regard de celles-ci figurent les procédures et fonctions qui y sont déclarées.

Par exemple :

```
A   B   C
        D
    E   F
G   H
```

Dans cet exemple :

- .A et G sont déclarés au niveau global;
- .B et E sont déclarés dans A;
- .C et D sont déclarés dans B;
- .F est déclaré dans E;
- .H est déclaré dans G.

Remarque : l'ordre dans lequel les procédures et fonctions sont citées, est celui dans lequel elles sont spécifiées ci-après. Cet ordre est également celui dans lequel elles sont déclarées, sauf :

- .FORMAT_REEL et TRI sont déclarées avant PRE_STAT_NOEUD_RECC;
- .RECH_HAMILT est déclaré avant ENR_TABLE_ROUTAGE;
- .AFF_SOL_FINAL est déclaré avant CALC_DIST_STAT_STAT;
- .les procédures relatives aux contraintes d'emplacement de fichiers sont déclarées après les procédures relatives à la fiabilité.

PRE_STAT_NOEUD_RECC
PRE_ENR_STATION
PRE_ENR_NOEUD
EX_STATNOEUD
VAL_STATNOEUD
AFF_STATION
AFF_NOEUD
TABL_STATNOEUD
NUM_STATION
STATNOEUD_RECCURR

ENR_STATNOEUD
PRE_ENR_LIGNE PRE_LIGNE_RECC
EX_LIGNE
VAL_LIGNE
AFF_LIGNE
TABL_LIGNE
NUM_LIGNE
LIGNE_RECCURR
ENR_LIGNE
ENR_TABLE_ROUTAGE
RECH_HAMILT
PRE_ENR_FICHIER
PRE_FICHIER_RECC
EX_FICHIER
VAL_FICHIER
AFF_FICHIER
TABL_FICHIER
NUM_FICHIER
FICH_RECCURR
ENR_FICHIER
EX_LOUE_CAP
VAL_LOUE_CAP
AFF_LOUE_CAP
TABL_LOUE_CAP
LOUE_CAP_RECCURR
ENR_LOUE_CAP
EX_NON_LOUE_CAP
VAL_NON_LOUE_CAP
AFF_NON_LOUE_CAP
TABL_NON_LOUE_CAP
NON_LOUE_CAP_RECCURR
ENR_NON_LOUE_CAP
ENR_CAPACITE
ENR_OPTION
AFF_OPTION
CORR_OPTION
AFF_CONTRAINTE_EMPL_FICH
CONTR_EMPL_FICH_TRAITEM

```
ENR_CONTRAINTE_EMPL_FICH
TABL_CONTRAINTE_EMPL_FICH
PRE_ENR_TAUX_LONG
ENR_LONG
ENR_TAUX_LONG
AFF_TAUX_LONG
CORR_TAUX
TAUX_LONG
PRE_ENR_FIABILITE
ENR_FIABILITE
AFF_FIABILITE
CORR_FIABILITE
SAISIE
INIT_MEMOIRE
E_MEMOIRE       EGALITE
M_MEMOIRE
INF
SUP
TRI
FORMAT_REEL
ENTIER_SUP
DISPO
DISPO_GLOB
FLUX            INTERM_FLUX
INDICEDANSCAP
CAP_COUT
DCALC
TCALC
CALC_DIST_STAT_STAT
CFAISABILITE
TFAISABILITE
INIT_CF_CAPSOL
INIT_TF_CAPSOL
CAPAC_U          CAPAC_U_APPROCHEE
CAPAC_B          CAPAC_B_APPROCHEE
INIT_QJTILDE
OPTIMIZING_ROUTINE     TROUVE_MIEUX    GENE_TAB   TRAITEMENT
SOLHEURISTIQUE     GENERATRICE
```

INCR_QJTILDE
AFF_SOL_FINAL

Une procédure (ou une fonction) est spécifiée selon le schéma suivant :

-. arguments : données reçues par la procédure (ou fonction);

-. pré-condition : propriétés que doivent posséder les arguments et/ou les variables globales pour que cette procédure (ou fonction) puisse être appelée;

-. résultats : résultats fournis par la procédure (ou fonction);

-. post-condition : propriétés que possèdent les arguments et/ou les variables globales après l'appel de la procédure (ou fonction).

Les constantes, types et variables globales utilisés dans ces spécifications sont explicités dans le code du programme : déclarations des constantes, types et variables globaux.

procédure PRE-STATNOEUD-RECC.

arguments : aux : tpointstatnoeud ;
i : integer ;
choix-stat : boolean ;
pre-condition : / ;
résultats : / ;
post-condition :
Si choix-stat = true
alors
début-alors
Si ((i > =1) et (i < =s))
alors la station décrite dans statnoeud [i]
est ajoutée à la liste chaînée de stations,
relative à pointstat ;
appel à PRE-STATNOEUD-RECC avec comme arguments :
aux^. premier ;
i+1 ;
choix-stat ;
fin-alors ;
Si choix-stat = false
alors
début-alors
Si ((i > =s+1) et (i < =s+n))
alors le noeud décrit dans statnoeud [i]
est ajouté à la liste chaînée de noeuds,
relative à pointnoeud ;
appel à PRE-STATNOEUD-RECC avec comme arguments :
aux^. premier ;
i+1 ;
choix-stat ;
fin-alors

procédure PRE-ENR-STATION.

arguments : choix-réseau : integer ;
pré-condition : choix-réseau $\in \{1,2,3\}$;

résultats : / ;
post-condition : Affectation à la variable s du nombre de stations du réseau
de numéro choix-réseau.
Les stations du réseau de numéro choix-réseau sont affectées
aux s premières composantes du tableau statnoeud.

appel à la procédure
PRE-STATNOEUD-RECC (pointstat, 1, true).

procédure PRE-ENR-NOEUD.

arguments : choix-réseau : integer ;
pré-condition : choix-réseau $\in \{1,2,3\}$;
résultats : / ;
post-condition : Affectation à la variable n du nombre de noeuds du réseau
de numéro choix-réseau.
Les noeuds du réseau de numéro choix-réseau sont affectés aux
composantes, allant de s+1 à s+n, du tableau statnoeud.

appel à la procédure
PRE-STATNOEUD-RECC (pointnoeud, 1, false).

fonction EX-STATNOEUD.

arguments : nom-int : string10 ;
choix-stat : boolean ;
pré-condition : / ;
résultats : EX-STATNOEUD : boolean ;
post-condition :
EX-STATNOEUD = true
ssi
(choix-stat = true
et la station de nom nom-int appartient à
la liste chaînée relative à pointstat)
ou
(choix-stat = false
et le noeud de nom nom-int appartient à
la liste chaînée relative à pointnoeud)

procédure VAL-STATNOEUD.

arguments : int-statnoeud : trecstatnoeud ;
choix-stat : boolean ;
pré-condition : / ;
résultats : / ;
post-condition : Affectation de valeurs aux différents champs de la variable int-statnoeud.

procédure AFF-STATION.

arguments : choix-réseau : integer ;
pré-condition : choix-réseau $\in \{0,1,2,3\}$;
résultats : / ;
post-condition :
Affichage d'un titre au tableau signalé ci-dessous .
Le contenu de ce titre dépend de la valeur de choix-réseau.
Les stations, appartenant à la liste chaînée relative à pointstat,
sont affichées à l'écran sous forme d'un tableau dont les rubriques sont :
NUM-STATION, NOM, STOCKAGE, PRIX-BIT, FLUX-ENTR, FLUX-SORT et PROB-DISC.

procédure AFF-NOEUD.

arguments : choix-réseau : integer ;
pré-condition : choix-réseau $\in \{0,1,2,3\}$;
résultats : / ;
post-condition :
Affichage d'un titre au tableau signalé ci-dessous.
Le contenu de ce titre dépend de la valeur de choix-réseau.
Les noeuds, appartenant à la liste chaînée relative à pointnoeud,
sont affichés à l'écran sous forme d'un tableau dont les rubriques sont :
NUM-NOEUD, NOM, FLUX-ENTR, FLUX-SORT et PROB-DISC.

procédure TABL-STATNOEUD.

arguments : / ;
pré-condition : / ;
résultats : / ;

post-condition :

Affecter à s (resp.n), le nombre de stations (resp.noeuds) du réseau. Affecter aux différentes composantes du tableau statnoeud, les stations de la liste chaînée relative à pointstat ainsi que les noeuds de la liste chaînée relative à pointnoeud.

fonction NUM-STATION.

arguments : stat : string10 ;
pré-condition : / ;
résultats : NUM-STATION : integer ;
post-condition :

Si la station stat est affectée à l'une des composantes du tableau statnoeud
alors NUM-STATION = indice de cette composante
sinon NUM-STATION = 0 .

procédure STATNOEUD-RECCURR.

arguments : aux : tpointstatnoeud ;
noeudst : string10 ;
ch : char ;
choix-stat : boolean ;
pré-condition : / ;
résultats : / ;
post-condition :

Les opérations sur les stations (resp.noeuds), signalées ci-dessous, seront effectuées sur la liste chaînée de stations (resp.noeuds), relative à pointstat (resp.pointnoeud) si choix-stat = true (resp.false).

Table de décision :

aux = nil	F	F	F	F	F	F	F	F	F	F	T	T	T	T
aux ^. stnoeud.nom = noeudst	T	T	T	T	T	T	T	T	T	F				
ch = 'A'	T										T			
ch = 'S'		T										T		
ch = 'M'			T										T	
ch = 'C'				T	T	T	T	T	T					T
choix-stat					T	T	F	F	F					
noeudst = noeudst1				T	F	F	F	F	F					
EX-STATNOEUD (noeudst1, true)					F	T	F	F	T					
EX-STATNOEUD (noeudst1, false)							F	T						
message : 'Cette station (resp. noeud) existe déjà'.	X													
supprimer la station (resp.noeud) noeudst.		X												
modifier les caractéristiques relatives à noeudst.			X											
introduire le nouveau nom (variable locale noeudst1).				X	X	X								
remplacer noeudst par noeudst1.					X		X							
message :'Ce nom est celui d'une autre station'.						X			X					
message :'Ce nom est celui d'un autre noeud'.								X						
appel à STATNOEUD-RECCURR (aux^. premier, noeudst, ch, choix-stat).										X				
ajouter la station (resp.noeud) noeudst et introduire ces caractéristiques.											X			
message : 'Cette station (resp. noeud) n'existe pas'.												X	X	X

procédure ENR-STATNOEUD.

arguments : choix-réseau : integer ;
choix-stat : boolean ;
pré-condition : choix-réseau $\in \{0,1,2,3\}$;
résultats : / ;
post-condition :

Afficher à l'écran un titre signalant l'enregistrement des stations (resp.noeuds) si choix-stat = true (resp.false).

Répéter

début-répéter

Afficher à l'écran le menu proposé à l'utilisateur pour cet enregistrement :

-soit ajouter une station (resp.noeud) : noté 'A' ;

-soit supprimer une station (resp.noeud) : noté 'S' ;

-soit changer le nom d'une station (resp.noeud) : noté 'C' ;

-soit modifier les caractéristiques d'une station (resp.noeud) : noté 'M' ;

-soit afficher les stations (resp.noeuds) déjà introduites : noté 'V' ;

-soit arrêter l'enregistrement des stations (resp.noeuds) : noté 'N' ;

Ces opérations seront effectuées sur la liste chaînée de stations (resp.noeuds), relative à pointstat (resp.pointnoeud) si choix-stat = true (resp.false).

En fait, le menu diffère suivant que :

-le réseau est personnel : menu = $\{$ 'A','S','C','M','V','N' $\}$;

-le réseau est préexistant :

-pour les stations : menu = $\{$ 'M','V','N' $\}$: cette restriction est due au fait que les matrices de trafic sont préintroduites : le nombre de stations et leur nom sont donc déterminés.

-pour les noeuds : menu = $\{$ 'A','M','V','N' $\}$: cette restriction est due au fait que certaines lignes sont préintroduites et qu'elles relient des noeuds (ou stations) préintroduits.

L'utilisateur est invité à choisir dans le menu qui lui est proposé : la lettre qu'il introduit est affectée à la variable ch.

Se référer à la table de décision qui suit.

fin-répéter
jusqu'à ce que ch = 'N'.

ch ∈ menu	T	T	T			F
ch = 'V'	F	F	F	T	T	
ch = 'N'	F	F	F			
choix-stat	T	F	F	T	F	
EX-STATNOEUD (noeudst, true)		T	F			
introduire le nom de la station (resp.noeud) (variable locale noeudst).	X	X	X			
appel à STATNOEUD-RECCURR (pointstat, noeudst, ch, true).	X					
appel à STATNOEUD-RECCURR (pointnoeud, noeudst, ch, false).			X			
message : 'Ce nom est celui d'une station et non d'un noeud'.		X				
AFF-STATION (choix-réseau).				X		
AFF-NOEUD (choix-réseau).					X	
message : 'Opération non significative'.						X

procédure PRE-LIGNE-RECC.

arguments : aux : tpointligne ;
i : integer ;
pré-condition : / ;
résultats : / ;
post-condition :
Si ((i > =1) et (i < =nbl))
alors la ligne décrite dans ligne [i] est ajoutée à la liste chaînée de lignes, relative à pointligne ;
appel à PRE-LIGNE-RECC avec comme arguments :
aux^. preml ;
i+1 .

procédure PRE-ENR-LIGNE.

arguments : choix-réseau : integer ;
pré-condition : choix-réseau $\in \{1,2,3\}$;
résultats : / ;
post-condition :
Affectation à la variable nbl du nombre de lignes du réseau de numéro choix-réseau.
Les lignes du réseau de numéro choix-réseau sont affectées aux nbl premières composantes du tableau ligne.
Appel à la procédure PRE-LIGNE-RECC (pointligne, 1).

fonction EX-LIGNE.

arguments : origine : string10 ;
extrémité : string10 ;
pré-condition : / ;
résultats: EX-LIGNE : boolean ;
post-condition :
EX-LIGNE = true
ssi la ligne, reliant la station (ou noeud) de nom origine et la station (ou noeud) de nom extrémité, appartient à la liste chaînée relative à pointligne.

procédure VAL-LIGNE.

arguments : int-ligne : trecligne ;
pré-condition : / ;
résultats : / ;
post-condition :
Affectation de valeurs aux différents champs de la variable int-ligne.

procédure AFF-LIGNE.

arguments : choix-réseau : integer ;
pré-condition : choix-réseau $\in \{0,1,2,3\}$;
résultats : / ;
post-condition :
Affichage, à l'écran, d'un titre au tableau signalé ci-dessous. Le contenu de ce titre dépend de la valeur de choix-réseau.
Les lignes, appartenant à la liste chaînée relative à pointligne, sont affichées à l'écran sous forme d'un tableau dont les rubriques sont :
NUM-LIGNE, INIT, FIN, LONG, PRIX-LONG, COUT-JOUR-MIN, COUT-NUIT-MIN.

procédure TABL-LIGNE.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
Affecter à nbl, le nombre de lignes du réseau .
Affecter aux différentes composantes du tableau ligne, les lignes de la liste chaînée relative à pointligne.
Affectation de valeurs aux variables globales mat-dist et mat-ligne.

fonction NUM-LIGNE.

arguments : origine : string10 ;
extrémité : string10 ;
pré-condition : / ;
résultats : NUM-LIGNE : integer ;

post-condition :

Si la ligne, reliant la station (ou noeud) de nom origine et la station (ou noeud) de nom extrémité, est affectée à l'une des composantes du tableau ligne
alors NUM-LIGNE = indice de cette composante
sinon NUM-LIGNE = 0 .

procédure LIGNE-RECCURR.

arguments : aux : tpointligne ;
origine : string10 ;
extrémité : string10 ;
ch : char ;
pré-condition : / ;
résultats : / ;
post-condition :

Les opérations sur les lignes, signalées ci-dessous, seront effectuées sur la liste chaînée de lignes relative à pointligne.
Table de décision :
la condition C utilisée dans cette table est :

```
(((aux^. lligne.init-statnoeud = origine)
   and (aux^. lligne.fin-statnoeud = extrémité))
or ((aux^. lligne.init-statnoeud = extrémité)
     and (aux^. lligne.fin-statnoeud = origine))) .
```

aux = nil	F	F	F	F	T	T	T
condition C	T	T	T	F			
ch = 'A'	T				T		
ch = 'S'		T				T	
ch = 'M'			T				T
message : 'Cette ligne existe déjà'.	X						
supprimer la ligne reliant origine à extrémité, ainsi que la ligne réciproque.		X					
modifier les caractéristiques de la ligne reliant origine à extrémité, ainsi que celles de la ligne réciproque .			X				
appel à LIGNE-RECCURR (aux^. preml, origine,extrémité, ch).				X			
ajouter la ligne reliant origine à extrémité, ainsi que la ligne réciproque et les caractéristiques de chacune de ces deux lignes.					X		
message : 'Cette ligne n'existe pas' .						X	X

procédure ENR-LIGNE.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :

Afficher à l'écran un titre signalant l'enregistrement des lignes.
Répéter
début-répéter
Afficher à l'écran le menu proposé à l'utilisateur pour cet enregistrement :

-soit ajouter une ligne : noté 'A' ;
-soit supprimer une ligne : noté 'S' ;
-soit modifier les caractéristiques d'une ligne : noté 'M' ;
-soit afficher les lignes déjà introduites : noté 'V' ;
-soit arrêter l'enregistrement des lignes : noté 'N'.

Ces opérations seront effectuées sur la liste chaînée de lignes relatives à pointligne.

Ici, menu = {'A','S','M','V','N'} .

L'utilisateur est invité à choisir dans le menu qui lui est proposé: la lettre qu'il introduit est affectée à la variable ch.

Se référer à la table de décision qui suit.

fin-répéter

jusqu'à ce que ch = 'N'.

ch ∈ menu	T	T	T	T		F
ch = 'V'	F	F	F	F	T	
ch = 'N'	F	F	F	F		
EX-STATNOEUD (origine, true)	F	T	T	T		
or EX-STATNOEUD (origine, false)						
EX-STATNOEUD (extrémité, true)			F	T		
or EX-STATNOEUD (extrémité, false)						
origine = extrémité		T	F	F		
introduire le nom de la station (ou noeud) origine de la ligne (variable locale origine).	X	X	X	X		
introduire le nom de la station (ou noeud) extrémité de la ligne (variable locale extrémité).		X	X	X		
message : 'Ce nom n'est ni celui d'une station, ni celui d'un noeud'.	X		X			
message : 'L'extrémité de la ligne est la même que l'origine'.		X				
appel à LIGNE-RECCURR (pointligne, origine, extrémité, ch).				X		
appel à AFF-LIGNE (choix-réseau).					X	
message : 'Opération non significative'.						X

procédure ENR-TABLE-ROUTAGE.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
Appel à RECH-HAMILT (le but de cet appel est de déterminer la valeur de res-connexe).
Si res-connexe
alors
Si l'utilisateur possède une table de routage
alors
début-alors
le programme invite l'utilisateur à donner pour tout couple de stations (les couples identiques et réciproques d'un couple considéré ne sont pas considérés), le chemin utilisé pour la communication entre ces stations. Le chemin pour un couple de stations, réciproque d'un couple considéré, est celui relatif à ce couple mais, évidemment, parcouru en sens inverse.
Les tests, quant à l'existence des stations (et noeuds) et des lignes données par l'utilisateur, sont effectués.
Affectation de valeurs à la variable globale routing.
fin-alors
sinon afficher le message :

"Certains couples de stations ne sont reliés par aucun chemin".

procédure RECH-HAMILT.

arguments : / ;
pré-condition : / ;
résultats : rés-connexe : boolean ;
post-condition :
Cette procédure donne le chemin le plus court pour relier deux stations quelconques du réseau.
Si ce chemin existe pour tout couple de stations,
alors rés-connexe = true ;
affectation de valeurs à la variable globale routing ;
sinon rés-connexe = false.

procédure PRE-FICHIER-RECC.

arguments : aux : tpointfichier ;
i : integer ;
pré-condition : / ;
résultats : / ;
post-condition :
Si ((i > =1) et (i < =nbf))
alors le fichier décrit dans fichier [i] est ajouté à la liste chaînée de fichiers, relative à pointfichier ;
appel à PRE-FICHIER-RECC avec comme arguments :
aux^. premf ;
i+1.

procédure PRE-ENR-FICHIER.

arguments : choix-réseau : integer ;
pré-condition : choix-réseau $\in \{1,2,3\}$;
résultats : / ;
post-condition :
Affectation à la variable nbf du nombre de fichiers du réseau de numéro choix-réseau.
Les fichiers du réseau de numéro choix-réseau sont affectés aux nbf premières composantes du tableau fichier.
Appel à la procédure PRE-FICHIER-RECC (pointfichier, 1).

fonction EX-FICHIER.

arguments : nom-int : string10 ;
pré-condition : / ;
résultats : EX-FICHIER : boolean ;
post-condition :
EX-FICHIER = true
ssi le fichier, de nom nom-int, appartient à la liste chaînée relative à pointfichier.

procédure VAL-FICHIER.

arguments : int-fichier : trecfichier ;
pré-condition : / ;
résultats : / ;
post-condition :
Affectation de valeurs aux différents champs de la variable int-fichier.

procédure AFF-FICHIER.

arguments : choix-réseau : integer ;
pré-condition : choix -réseau $\in \{0,1,2,3\}$;
résultats : / ;
post-condition :
Affichage, à l'écran, d'un titre au tableau signalé ci-dessous. Le contenu de ce titre dépend de la valeur de choix-réseau.
Les fichiers, appartenant à la liste chaînée relative à pointfichier, sont affichés à l'écran sous forme d'un tableau dont les rubriques sont :
NUM-FICHIER, NOM, LONG et DISPO-SOUH.

procédure TABL-FICHIER.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
Affecter à nbf, le nombre de fichiers du réseau.
Affecter aux différentes composantes du tableau fichier, les fichiers de la liste chaînée relative à pointfichier.

fonction NUM-FICHIER.

arguments : fich : string10 ;
pré-condition : / ;
résultats : NUM-FICHIER : integer ;
post-condition :
Si le fichier fich est affecté à l'une des composantes du tableau fichier
alors NUM-FICHIER = indice de cette composante.
sinon NUM-FICHIER = 0

procédure FICH-RECCURR.

arguments : aux : tpointfichier ;
fich : string[10] ;
ch : char ;
pré-condition : / ;
résultats : / ;
post-condition :

Les opérations sur les fichiers, signalées ci-dessous, seront effectuées sur la liste chaînée de fichiers, relative à pointfichier.

Table de décision :

aux = nil	F	F	F	F	F	F	F	T	T	T	T
aux^. ffich.nom = fich	T	T	T	T	T	T	F				
ch = 'A'	T							T			
ch = 'S'		T							T		
ch = 'M'			T							T	
ch = 'C'				T							T
fich = fich 1				T	F	F					
EX-FICHIER (fich1)					T	F					
message : 'Ce fichier existe déjà'.	X										
supprimer le fichier fich .		X									
modifier les valeurs relatives à fich .			X								
introduire le nouveau nom (variable locale fich1) .				X	X	X					
remplacer fich par fich1 .						X					
message : 'Ce nom est celui d'un autre fichier!					X						
ajouter le fichier fich et introduire ses caractéristiques .								X			
message : 'Ce fichier n'existe pas'.									X	X	X
appel à FICH-RECCURR (aux^. premf, fich, ch) .							X				

procédure ENR-FICHIER.

arguments : choix-réseau : integer ;
pré-condition : choix-réseau $\in \{0,1,2,3\}$;
résultats : / ;
post-condition :

Afficher à l'écran un message signalant l'enregistrement des fichiers.
Répéter
début-répéter

Afficher à l'écran le menu proposé à l'utilisateur pour cet enregistrement :
-soit ajouter un fichier : noté 'A' ;
-soit supprimer un fichier : noté 'S' ;
-soit changer le nom d'un fichier : noté 'C' ;
-soit modifier les caractéristiques d'un fichier : noté 'M' ;
-soit afficher les fichiers déjà introduits : noté 'V' ;
-soit arrêter l'enregistrement des fichiers : noté 'N'.

Ces opérations seront effectuées sur la liste chaînée de fichiers relatives à pointfichier.

En fait, le menu diffère suivant que :
-le réseau est personnel : menu = $\{$'A','S','C','M','V','N'$\}$;
-le réseau est préexistant : menu = $\{$'M','V','N'$\}$:
cette restriction est due au fait que les matrices de trafic sont préintroduites : le nombre de fichiers et leur nom sont donc déterminés.

L'utilisateur est invité à choisir dans le menu qui lui est proposé : la lettre qu'il introduit est affectée à la variable ch.

Se référer à la table de décision qui suit.

fin-répéter
jusqu'à ce que ch = 'N'.

ch ∈ menu	T		F
ch = 'V'	F	T	
ch = 'N'	F		
introduire le nom du fichier (variable locale fich) .	X		
appel à FICH-RECCURR (pointfichier, fich, ch).	X		
appel à AFF-FICHIER (choix-réseau).		X	
message : 'Opération non significative'.			X

fonction EX-LOUE-CAP.

arguments : val-int : real ;
pré-condition : / ;
résultats : EX-LOUE-CAP : boolean ;
post-condition :
EX-LOUE-CAP = true
ssi la capacité louée, de valeur val-int, appartient à la liste chaînée relative à pointlouecap.

procédure VAL-LOUE-CAP.

arguments : int-cap-loué : treccap-loué ;
pré-condition : / ;
résultats : / ;
post-condition :
Affectation de valeurs aux différents champs de la variable int-cap-loué.

procédure AFF-LOUE-CAP.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
Affichage, à l'écran, d'un titre au tableau signalé ci-dessous.
Les capacités, appartenant à la liste chaînée relative à pointlouecap, sont affichées à l'écran sous forme d'un tableau dont les rubriques sont : NUM-CAPAC.LOUEE, VALEUR, CONST.A et CONST.B.

procédure TABL-LOUE-CAP.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
Affecter à nbcap-loué, le nombre de capacités louées du réseau.
Affecter aux différentes composantes du tableau cap-loué, les capacités de la liste chaînée relative à pointlouécap.

procédure LOUE-CAP-RECCURR.

arguments : aux : tpointlouecap ;
ccaploue : real ;
ch : char ;
pré-condition : / ;
résultats : / ;
post-condition :

Les opérations sur les capacités louées, signalées ci-dessous, seront effectuées sur la liste chaînée de capacités louées, relative à pointlouécap.

Table de décision :

aux = nil	F	F	F	F	T	T	T
aux ^. louecap.valeur = ccaploué	T	T	T	F			
ch = 'A'	T				T		
ch = 'S'		T				T	
ch = 'M'			T				T
message : 'Cette capacité louée existe déjà'.	X						
supprimer la capacité de valeur ccaploué .		X					
modifier les caractéristiques de la capacité de valeur ccaploué .			X				
appel à LOUE-CAP-RECCURR (aux ^. premlc, ccaploué, ch).				X			
ajouter la capacité de valeur ccaploué et introduire ses caractéristiques.					X		
message : 'Cette capacité louée n'existe pas'.						X	X

procédure ENR-LOUE-CAP.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :

Afficher à l'écran un titre signalant l'enregistrement des capacités louées.
Répéter
début-répéter
Afficher à l'écran le menu proposé à l'utilisateur pour cet enregistrement :
-soit ajouter une capacité louée : noté 'A' ;
-soit supprimer une capacité louée : noté 'S' ;
-soit modifier les caractéristiques d'une capacité louée : noté 'M' ;
-soit afficher les capacités louées déjà introduites : noté 'V' ;
-soit arrêter l'enregistrement des capacités louées : noté 'N'.

Ces opérations seront effectuées sur la liste chaînée de capacités louées, relative à tpointcaploué.

Ici; menu = { 'A','S','M','V','N' } .

L'utilisateur est invité à choisir dans le menu qui lui est proposé : la lettre qu'il introduit est affectée à la variable ch.

Se référer à la table de décision qui suit.
fin-répéter
jusqu'à ce que ch = 'N'.

ch ∈ menu	T		F
ch = 'V'	F	T	
ch = 'N'	F		
introduire la valeur de la capacité louée (variable locale ccaploué).	X		
appel à LOUE-CAP-RECCURR (pointlouecap, ccaploué, ch).	X		
appel à AFF-LOUE-CAP .		X	
message : 'Opération non significative'.			X

fonction EX-NON-LOUE-CAP.

arguments : val-int : real ;
pré-condition : / ;
résultats : EX-NON-LOUE-CAP : boolean ;
post-condition :
EX-NON-LOUE-CAP = true
ssi la capacité non louée, de valeur val-int, appartient à la liste chaînée relative à pointnonlouecap.

procédure VAL-NON-LOUE-CAP.

arguments : int-cap-non-loue : treccap-non-loué ;
pré-condition : / ;
résultats : / ;
post-condition :
Affectation de valeurs aux différents champs de la variable int-cap-non-loué.

procédure AFF-NON-LOUE-CAP.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
Affichage, à l'écran, d'un titre au tableau signalé ci-dessous.
Les capacités, appartenant à la liste chaînée relative à pointnonlouecap, sont affichées à l'écran sous forme d'un tableau dont les rubriques sont : NUM-CAPAC-NON-LOUEE, VALEUR, CONST.C.

procédure TABL-NON-LOUE-CAP.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
Affecter à nbcap-non-loué, le nombre de capacités non louées du réseau.
Affecter aux différentes composantes du tableau cap-non-loue, les capacités de la liste chaînée relative à pointnonlouecap.

procédure NON-LOUE-CAP-RECCURR.

arguments : aux : tpointnonlouecap ;
ccapnonloué : real ;
ch : char ;
pré-condition : / ;
résultats : / ;
post-condition :

Les opérations sur les capacités non louées, signalées ci-dessous, seront effectuées sur la liste chaînée de capacités non louées, relative à pointnonlouecap.

Table de décision.

aux = nil	F	F	F	F	T	T	T
aux^. nonlouecap.valeur = ccapnonloué	T	T	T	F			
ch = 'A'	T				T		
ch = 'S'		T				T	
ch = 'M'			T				T
message : 'Cette capacité non louée existe déjà'.	X						
supprimer la capacité de valeur ccapnonloué.		X					
modifier les caractéristiques de la capacité de valeur ccapnonloué.			X				
appel à NON-LOUE-CAP-RECCURR (aux^. premmlc, ccapnonloué, ch).				X			
ajouter la capacité de valeur ccapnonloué et introduire ses caractéristiques.					X		
message : 'Cette capacité non louée n'existe pas'.						X	X

procédure ENR-NON-LOUE-CAP.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :

Afficher à l'écran un titre signalant l'enregistrement des capacités non louées.
Répéter
début-répéter
Afficher à l'écran le menu proposé à l'utilisateur pour cet enregistrement :
-soit ajouter une capacité non louée : noté 'A' ;
-soit supprimer une capacité non louée : noté 'S' ;
-soit modifier les caractéristiques d'une capacité non louée : noté 'M' ;
-soit afficher les capacités non louées déjà introduites : noté 'V' ;
-soit arrêter l'enregistrement des capacités non louées : noté 'N'.

Ces opérations seront effectuées sur la liste chaînée de capacités non louées, relative à tpointnonlouecap.

Ici, menu = { 'A','S','M','V','N' } .

L'utilisateur est invité à choisir dans le menu qui lui est proposé : la lettre qu'il introduit est affectée à la variable ch.

Se référer à la table de décision qui suit.
fin-répéter
jusqu'à ce que ch = 'N'.

ch ∈ menu	T		F
ch = 'V'	F	T	
ch = 'N'	F		
introduire la valeur de la capacité non louée (variable locale ccapnonloué).	X		
appel à NON-LOUE-CAP-RECCURR (pointnonlouecap, ccapnonloué, ch).	X		
appel à AFF-NON-LOUE-CAP.		X	
message : 'Opération non significative'.			X

procédure ENR-CAPACITE.

arguements : / ;
pré-condition : / ;
résultats : / ;
post-condition :
Affectation de valeurs aux variables globales prix-fx-ligne, cap et nbcap.

procédure ENR-OPTION.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
Affectation de valeurs aux variables globales relatives aux options.

procédure AFF-OPTION.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
Affichage à l'écran des différentes options choisies par l'utilisateur.

procédure CORR-OPTION.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
L'utilisateur du programme introduit le numéro de l'option erronée et celle-ci est corrigée.

procédure AFF-CONTRAINTE-EMPL-FICH.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
Affichage des différentes contraintes d'emplacement des fichiers, s'il en existe.
Sinon, affichage du message : "Il n'y a pas de contrainte d'emplacement des fichiers".

procédure CONTR-EMPL-FICH-TRAITEM.

arguments : stat : string10 ;
fich : string10 ;
ch : char ;
pré-condition : EX-STATNOEUD (stat, true) = true ;
EX-FICHIER (fich) = true ;
résultats : / ;
post-condition :
Posons num1 = NUM-STATION (stat) et num2 = NUM-FICHIER (fich).

Table de décision :

ch = 'A'	T	T		
ch = 'S'			T	T
matrice-masque [num1, num2] = 2	T	F	T	F
enregistrer la contrainte .	X			
message : "Il existe déjà une contrainte concernant cette station et ce fichier".		X		
message : "Il n'existe pas de contrainte concernant cette station et ce fichier".			X	
matrice-masque [num1, num2] = 2				X

procédure ENR-CONTRAINTE-EMPL-FICH.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
Afficher à l'écran un titre signalant l'enregistrement des contraintes d'emplacement des fichiers.

Initialisation à 2 des éléments de matrice-masque.

Répéter

début-répéter

Afficher à l'écran le menu proposé à l'utilisateur pour cet enregistrement :

-soit ajouter une contrainte : noté 'A' ;

-soit supprimer une contrainte : noté 'S' ;

-soit afficher les contraintes déjà introduites : noté 'V' ;

-soit arrêter l'enregistrement des contraintes : noté 'N'.

Ici, menu = $\{$'A','S','V','N'$\}$.

L'utilisateur est invité à choisir dans le menu qui lui est proposé : la lettre qu'il introduit est affectée à la variable ch.

Se référer à la table de décision qui suit.

fin-répéter

jusqu'à ce que ch = 'N'.

ch ∈ menu	T	T	T		F
ch = 'V'	F	F	F	T	
ch = 'N'	F	F	F		
EX-STATNOEUD (stat, true)	F	T	T		
EX-FICHIER (fich)		F	T		
introduire le nom de la station concernée (variable locale stat).	X	X	X		
message : 'Ce nom n'est pas celui d'une station'.	X				
introduire le nom du fichier concerné (variable locale fich).		X	X		
message : 'Ce nom n'est pas celui d'un fichier'.		X			
appel à CONTR-EMPL-FICH-TRAITEM (stat, fich, ch).			X		
appel à AFF-CONTR-EMPL-FICH.				X	
message : 'Opération non significative'.					X

procédure TABL-CONTRAINTE-EMPL-FICH.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
Affectation d'une valeur à la variable globale masque à partir de la valeur de la variable globale matrice-masque.

procédure PRE-ENR-TAUX-LONG.

arguments : choix-réseau : integer ;
pré-condition : choix-réseau $\in \{1,2,3\}$;
résultats : / ;
post-condition : affectations aux variables globales citées ci-dessous de valeurs relatives au réseau de numéro choix-réseau ;

mu-query-perm
mu-query-jour
mu-query-nuit
mu-ret-query-perm
mu-ret-query-jour
mu-ret-query-nuit
mu-upd-perm
mu-upd-jour
mu-upd-nuit
mu-ret-upd-perm
mu-ret-upd-jour
mu-ret-upd-nuit
u-perm
uprime-perm
v-perm
vprime-perm
u-jour
uprime-jour
v-jour

v-prime-jour
u-nuit
uprime-nuit
v-nuit
vprime-nuit

procédure ENR-LONG.

arguments : traff-occurr : ttraff-occurr ;
nature-flux : tnature-flux;
pré-condition : / ;
résultats : / ;
post-condition : enregistrement de la longueur moyenne d'un message pour le trafic relatif à traff-occurr et nature-flux.

procédure ENR-TAUX-LONG.

arguments : traff-occurr : ttraff-occurr ;
nature-flux : tnature-flux ;
pré-condition : / ;
résultats : / ;
post-condition : enregistrement des taux et de la longueur moyenne d'un message pour le trafic relatif à traff-occurr et nature-flux.

procédure AFF-TAUX-LONG.

arguments : traff-occurr : ttraff-occurr ;
nature-flux : tnature-flux ;
pré-condition : / ;
résultats : / ;
post-condition : affichage à l'écran des taux et de la longueur moyenne d'un message pour le trafic relatif à traff-occurr et nature-flux.

procédure CORR-TAUX.

arguments : traff-occurr : ttraff-occurr ;

```
            nature-flux : tnature-flux ;
pré-condition : / ;
résultats : données-correctes : boolean ;
post-condition : introduction, par l'utilisateur, de la station et
                 du fichier pour lesquels le taux, relatif à traff-
                 occurr et nature-flux, est erroné;
                 si ces données introduites par l'utilisateur sont
                    correctes
                 alors
                      début-alors
                           données-correctes := true;
                           correction du taux, relatif à traff-occurr
                              et nature-flux, concernant les station
                              et fichier introduits
                      fin-alors
                 sinon données-correctes := false.
```

procédure TAUX-LONG.

```
arguments : traff-occurr : ttraff-occurr ;
            nature-flux : tnature-flux ;
            choix-réseau : integer ;
pré-condition : choix-réseau ∈ { 0,1,2,3 } ;
résultats : / ;
post-condition : cette procédure assure l'enregistrement et la vali-
                 dation des taux relatifs à traff-occurr et nature-flux
                 ainsi que de la longueur moyenne des messages corres-
                 pondants; pour ce faire, elle fait appel aux procédures
                 pré-citées relatives aux taux et à la longueur des
                 messages.
```

procédure PRE-ENR-FIABILITE.

```
arguments : choix-réseau : integer ;
pré-condition : choix-réseau ∈ {1,2,3} ;
résultats : / ;
post-condition : affectation, à la variable globale r, d'une valeur
                 relative au réseau de numéro choix-réseau.
```

procédure ENR-FIABILITE.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition : affectation d'une valeur à la variable globale r.

procédure AFF-FIABILITE.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition : affichage à l'écran de la valeur de la variable
globale r.

Procédure CORR-FIABILITE.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition : introduction, par l'utilisateur, du couple de stations
pour lequel la composante de r est erronée;
si le couple introduit est effectivement un couple de
stations
alors si le couple est un couple identique
alors message : "la fiabilité vaut nécessairement
1"
sinon correction du taux de fiabilité relatif à
ces deux stations
sinon message : "ce que vous avez introduit n'est pas
un couple de stations".

procédure SAISIE.

arguments : / ;
pré-condition : / ;
résultats : / ;

post-condition : assure une saisie correcte des données, grâce aux différentes procédures (et fonctions) spécifiées ci-avant.

procédure INIT-MEMOIRE.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-conditions :

Les variables globales :

- add-mémoire;
- sous-mémoire;
- sommet-add-mémoire;
- sommet-sous-mémoire;
- nbre-accès-mémoire;
- nbre-accès-mémoire-réussi;
- nbre-rempli-add-mémoire;
- nbre-rempli-sous-mémoire;

sont initialisées à zéro.

fonction E-MEMOIRE.

arguments : x : imaxsmaxf;
pré-condition : / ;
résultats : E-MEMOIRE : boolean;
post-conditions :

- E-MEMOIRE est true si x est mémorisée soit dans add-mémoire, soit dans sous-mémoire. Le terme"mémorisé" est à prendre au sens large c'-à-d. au sens décrit dans la mise en oeuvre du programme (paragraphe II.2), dans la partie relative à la mémorisation des allocations de fichiers déjà étudiées;

- nbre-accès-mémoire : = nbre-accès-mémoire + 1;
- nbre-accès-mémoire-réussi : = nbre-accès-mémoire-réussi + 1
  si E-MEMOIRE = true.

procédure M-MEMOIRE.

arguments : x : imaxsmaxf ;
add : boolean;
q : pairemaxf;
pré-condition : / ;
résultats : / ;
post-conditions :

- si add = true
  alors
  début-alors
  sommet-add-mémoire : = sommet-add-mémoire + 1;
  si sommet-add-mémoire > DIM-ADD-MEMOIRE
  alors
  début-alors
  sommet-add-mémoire : = 1;
  nbre-rempli-add-mémoire : = nbre-rempli-add-mémoire + 1
  fin-alors
  fin-alors;
- si add = false
  alors
  début-alors
  sommet-sous-mémoire : = sommet-sous-mémoire + 1;
  si sommet-sous-mémoire > DIM-SOUS-MEMOIRE
  alors
  début-alors
  sommet-sous-mémoire : = 1;
  nbre-rempli-sous-mémoire : = nbre-rempli-sous-mémoire + 1
  fin-alors
  fin-alors;
- La matrice x est mémorisée dans :
  - add-mémoire [sommet-add-mémoire] si add = true;
  - sous-mémoire [sommet-sous-mémoire] si add = false.

  De nouveau, le terme "mémorisé" est à prendre au sens large c'-à-d. au sens décrit dans la mise en oeuvre du programme (paragraphe II.2), dans la partie relative à la mémorisation des allocations de fichiers déjà étudiées.

procédure INF.

arguments : tab : rmax;
pteur : integer;
pré-condition : pteur ≤ MAX;
résultats : min : real;
ind : integer;
post-condition :
Si pteur ≤ 0
alors
début-alors
min : = 0
ind : = 0
fin-alors
sinon
début-sinon
min = minimum de { tab [1],..., tab [pteur] } ;
ind = indice de ce minimum dans le tableau tab
fin-sinon.

fonction SUP.

arguments : tab : rmax;
pteur : integer;
pré-condition : pteur ≤ MAX;
résultats : SUP : boolean;
post-condition :
Si pteur ≤ 0
alors SUP = 0
sinon SUP = maximum de { tab [1],..., tab [pteur] } .

procédure TRI.

arguments : tab : rmax;
pteur : integer;
pré-condition : pteur ≤ MAX;
résultats : tab : rmax;
indice : imax;

post-condition :

Le tableau tab est trié par ordre croissant pour ses indices allant de 1 jusqu'à pteur;

Pour i $\in [1, \text{pteur}]$: indice [i] = indice dans le tableau tab non trié (celui reçu en argument) du $i^{ème}$ élément du tableau tab trié.

procédure FORMAT-REEL.

arguments : r : real ;
part-ent : integer ;
part-dec : integer ;
pré-condition : / ;
résultats : / ;
post-condition : ajoût de blancs lié à la présentation des données en tableaux .

fonction ENTIER-SUP.

arguments : a : real ;
pré-condition : / ;
résultats : ENTIER-SUP : integer ;
post-condition : ENTIER-SUP est le plus petit entier supérieur à a.

fonction DISPO.

arguments : allocfich : maxs ;
indfich : integer ;
traff-occurr : ttraff-occurr ;
pré-condition : / ;
résultats : DISPO : real :
post-condition :
DISPO = disponibilité du fichier, de numéro indfich et dont un exemplaire
et est stocké dans la station i ssi allocfich [i] = true
(pour i ∈ [1,s]), calculée pour le trafic de messages relatif à
traff-occurr.

fonction DISPO-GLOB.

arguments : x : imaxsmaxf ;
traff-occurr : ttraff-occurr.
pré-condition : / ;
résultats : DISPO-GLOB : boolean ;
post-condition :
DISPO-GLOB = true
ssi l'allocation de fichier x vérifie la contrainte de disponibilité pour le
trafic de messages relatif à traff-occurr.

procédure FLUX.

arguments : x : imaxsmaxf ;
pré-condition : / ;
résultats : / ;
post-condition :
Si reg-permanent = true
alors des valeurs sont affectées aux variables globales

lambda-perm ;
lambda-query-perm ;
lambda-ret-query-perm ;
lambda-upd-perm ;
lambda-ret-upd-perm ;
gamma-perm ;
mu-perm ;

conformément aux formules de calcul de flux considérées dans le paragraphe II.1.1.6) ;

Si reg-permanent = false
alors des valeurs sont affectées aux variables globales

lambda-jour ;
lambda-query-jour ;
lambda-ret-query-jour ;
lambda-upd-jour ;
lambda-ret-upd-jour ;
gamma-jour ;
mu-jour ;
lambda-nuit ;
lambda-query-nuit ;
lambda-ret-query-nuit ;
lambda-upd-nuit ;
lambda-ret-upd-nuit ;
gamma-nuit ;
mu-nuit ;

conformément aux formules de calcul de flux considérées dans le paragraphe II.1.1.6).

procédure INDICEDANSCAP.

arguments : capacité : tcap-ligne ;
pré-condition : / ;
résultats : indcapacité : imaxl ;
post-condition :
pour l ∈ [1,nbl] :
indcapacité [l] = indice dans cap [l] de capacité [l] ..

procédure CAP-COUT.

arguments : x : imaxsmaxf ;
pré-condition : / ;
résultats : / ;

post-condition :

pour l ∈ [1,nbl] et j ∈ [1,nbcap] :

Si (cap [l] [j]. loue = false) et (ligne-bidir = 0)

alors affectation d'une valeur à la variable globale cap [l] [j]. cout (cette valeur est le tarif mensuel de la ligne l si elle était munie de la capacité cap [l] [j]. valeur) ;

Si (cap [l] [j]. loue = false) et (ligne-bidir = 1)

alors affectation d'une valeur à la variable globale cap [l] [j]. cout (cette valeur est le tarif mensuel de la ligne bidirectionnelle physique à laquelle la ligne l est associée si cette ligne bidirectionnelle était munie de la capacité cap [l] [j]. valeur).

fonction DCALC.

arguments : x : imaxsmaxf ;
capacité : tcap-ligne ;
pré-condition : / ;
résultats : DCALC : real ;
post-condition :

DCALC = tarif mensuel, de l'allocation de fichiers x munie de l'allocation de capacités capacité, calculé conformément à la formule de cout considérée dans le paragraphe II.1.3.

fonction TCALC.

arguments : x : imaxsmaxf ;
capacité : tcap-ligne ;
pré-condition : appel à FLUX (x) ;
résultats : TCALC : real ;
post-condition :

TCALC est le délai moyen pour le transfert d'un message calculé conformément à la formule considérée dans le paragraphe II.1.2.

procédure CALC-DIST-STAT-STAT.

arguments : / ;
pré-condition : / ;
résultats : / ;

post-condition :

Affecter une valeur à la matrice mat-rout-dist (variable globale).

fonction CFAISABILITE.

arguments : x : imaxsmaxf ;
pré-condition : les tableaux cap-loué et cap-non-loué sont triés par ordre croissant du champ valeur de leurs composantes ;
résultats : CFAISABILITE : boolean ;
post-condition :

CFAISABILITE = true
ssi x est cfaisable (terme défini dans la mise en oeuvre du programme : paragraphe II.2) :
-pour le trafic de messages relatif au régime permanent si reg-permanent = 1 ;
-pour les trafics de messages relatifs aux périodes de jour et de nuit si reg-permanent = 0.

fonction TFAISABILITE.

arguments : x : imaxsmaxf ;
pré-condition : / ;
résultats : TFAISABILITE : boolean ;
post-condition :

TFAISABILITE = true
ssi x est tfaisable (terme défini dans la mise en oeuvre du programme : paragraphe II.2) :
-pour le trafic de messages relatif au régime permanent si reg-permanent = 1 ;
-pour le trafic de messages relatif à la période jour si reg-permanent = 0.

procédure INIT-CF-CAPSOL.

arguments : x : imaxsmaxf ;
pré-conditions : x est cfaisable ;
appel à CAP-COUT (x) (voir spécification de CAP-COUT) ;
cap [1] trié par ordre croissant du champ coût de ses composantes (pour 1 ∈ [1,nbl]) ;

résultats : capacité : tcap-ligne ;
post-condition :
capacité est l'allocation de capacités qui, parmi l'ensemble des allocations de capacités qui, associées à x, vérifient la contrainte de capacité, est celle qui, associée à x, donne le coût minimal.

procédure INIT-TF-CAPSOL.

arguments : x : imaxsmaxf ;
pré-condition : x est tfaisable ;
résultats : capacité : tcap-ligne ;
post-condition :
capacité est l'allocation de capacités qui, parmi l'ensemble des allocations de capacités qui, associées à x, vérifient la contrainte de délai, est celle qui, associée à x, donne le coût minimal.

procédure CAPAC-U.

arguments : x : imaxsmaxf ;
pré-condition : x est tfaisable ;
ligne-bidir = false ;
résultats : capsol : tcap-ligne ;
post-condition :
Si ((nbcap-loué = 0) ou (nbcap-non-loué = 0))
alors capsol est l'allocation de capacités qui, associée à x,
minimise le coût.
sinon capsol n'est qu'une solution approchée de cette minimisation.
Dans les deux cas une solution approchée est donnée par un appel à la procédure CAPAC-U-APPROCHEE (voir mise en oeuvre du programme, paragraphe II.2.2).

procédure CAPAC-B.

arguments : x : imaxsmaxf ;
pré-condition : x est tfaisable ;
ligne-bidir = true ;
résultats : capsol : tcap-ligne ;

post-condition :

Si ((nbcap-loué = 0) ou (nbcap-non-loué = 0))
alors capsol est l'allocation de capacités qui, associée à x,
minimise le coût
sinon capsol n'est qu'une solution approchée de cette minimisation.
Dans les deux cas une solution approchée est donnée par un appel à la procédure CAPAC-B-APPROCHEE. (voir mise en oeuvre du programme, paragraphe II.2.2).

procédure INIT-QJTILDE.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :

le vecteur qtilde est initialisé :
qtilde [j] = nombre minimum d'exemplaires du fichier j
(pour j ∈ [1,nbf]),
tel qu'il existe au moins une allocation de ce fichier j
qui soit disponible, si ce nombre existe ;
= s+1 sinon ;
Pour j ∈ [1,nbf] tel que qtilde [j] = s+1 :
afficher le message "La contrainte de disponibilité pour le fichier j est impossible à satisfaire".
s'il existe j ∈ [1,nbf] tel que qtilde [j] = s+1
alors la variable globale pbfais-dispo est mise à false
sinon pbfais-dispo est mise à true.

procédure OPTIMIZING-ROUTINE.

arguments : x : imaxsmaxf ;
pré-condition : x est cfaisable, dfaisable et efaisable ;
résultats : z : imaxsmaxf ;
capsol : tcap-ligne ;
d : real ;
t : real ;
optim-fournit-sol : boolean ;

post-condition :

z = allocation de fichiers tfaisable produite par l'algorithme lié à la procédure OPTIMIZING-ROUTINE (voir mise en oeuvre du programme au paragraphe II.2.3) si cette allocation existe ;
= matrice nulle sinon.

capsol = allocation de capacités capsol donnée par
CAPAC-U (z,capsol) (resp. CAPAC-B (z,capsol)) si
ligne-bidir = 0 (resp.=1) et si z ≠ matrice nulle ;
= allocation nulle (c'-à-d capsol [1] . capac = 0
et capsol [1] . loue = false
pour 1 ∈ [1,nbl]) sinon.

d = coût associé à z et capsol si z ≠ matrice nulle ;
= infini sinon.
t = délai moyen d'un message associé à z et capsol si z ≠ matrice nulle ;
= infini sinon.

la variable globale optim-fournit = sol
= true ssi z ≠ matrice nulle.

fonction TROUVE-MIEUX.

arguments : add : boolean ;
post-condition : / ;
résultats : TROUVE-MIEUX : boolean ;
post-condition :

cette fonction correspond - à la procédure d'ajuut (mise en oeuvre du programme paragraphe II.2.3) si add = true ;
- à la procédure de soustraction (mise en oeuvre du programme paragraphe II.2.3) si add = false.

procédure GENE-TAB.

arguments : l : integer ;
pré-condition : l ≤ r où r est une variable locale à TROUVE-MIEUX ;
résultats : / ;
post-condition :

cette procédure génère toutes les combinaisons de l éléments parmi r ;

en fait r est le nombre de stations élues dans l'algorithme d'ajout (ou de soustraction) évoqué dans la mise en oeuvre du programme (paragraphe II.2.3), rappelons que r $\leq$ nbf.

procédure TRAITEMENT.

arguments : tab : ttab ;
pré-condition : / ;
résultats : / ;
post-condition :
étude de l'allocation de fichiers obtenue à partir du tableau : tab (ce tableau mémorise les stations élues évoquées dans GENE-TAB).

procédure SOLHEURISTIQUE.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
génère les germes relatifs à un moule déterminé (voir mise en oeuvre du programme paragraphe II.1.4) par l'intermédiaire d'un appel à la procédure recursive GENERATRICE.

procédure INCR-QJTILDE.

arguments : / ;
pré-condition : / ;
résultats : / ;
post-condition :
incrémentation de la variable globale qtilde (voir mise en oeuvre du programme paragraphe II.1.4) par l'intermédiaire des procédures RECCURR et COMBINAISON et pour chaque valeur de qtilde, appel à SOLHEURISTIQUE.

procédure AFF-SOL-FINAL.

arguments : / ;
pré-condition : le problème admet une solution ;
résultats : / ;
post-condition : affichage à l'écran :

-. de l'allocation de fichiers solution ;
-. de l'allocation de capacités solution ;
-. du flux résultant sur chaque ligne :
il sera décomposé en flux relatifs aux "query traffic", "query-return traffic", "update traffic" et "update-return traffic" (si reg-permanent = 0, alors ces flux seront donnés pour le jour et pour la nuit);
-. pour chaque station s et chaque fichier f, de la station où est stocké l'exemplaire du fichier f à utiliser par les abonnés de la station s, si util-uniform = 0;
-. du délai moyen des messages ;
-. des nombres de bits par seconde entrant et sortant de chaque station et de chaque noeud ;
-. du nombre de bits stockés à chaque station ;
-. de la table de routage, si elle est fournie par le programme et non par l'utilisateur.

Prolongée linéaire.

Le problème est le suivant :

Trouver le M-uple $(x_1,\ldots,x_M)$ de R^M

qui minimise $D = \sum_{i=1}^{M} d_i\, x_i$

sous la contrainte

$$\sum_{i=1}^{M} (\lambda_i / \gamma) \quad (1 / (\mu x_i - \lambda_i)) \leq T_{max}$$

où d_i, λ_i, γ, μ et T_{max} sont des constantes positives (pour $i \in [1,M]$)

Solution de ce problème :

La fonction considérée dans la contrainte étant décroissante par rapport à x_i ($i \in [1,M]$), l'inégalité dans la contrainte peut être remplacée par une égalité.

Considérons le Lagrangien :

$$G = \sum_{i=1}^{M} d_i\, x_i + \beta\left(\left\{ \sum_{i=1}^{M} (\lambda_i / \gamma) \quad (1 / (\mu x_i - \lambda_i)) \right\} - T_{max} \right)$$

Les M équations

$$\frac{\partial G}{\partial x_i} = 0 \text{ pour } i \in [1,M]$$

donnent

$$0 = d_i - \beta(\lambda_i / \gamma) * (\mu / (\mu x_i - \lambda_i)^2)$$

$$(\mu x_i - \lambda_i)^2 = \lambda_i \mu \beta / \gamma\, d_i$$

$$x_i = (1/\mu) * \sqrt{\lambda_i \mu \beta / \gamma\, d_i} + \lambda_i / \mu$$

Il reste à calculer la valeur de la constante β :

$$\sum_{i=1}^{M} (\lambda_i / \gamma) \quad (1 / (\mu x_i - \lambda_i)) = T_{max}$$

$$\sum_{i=1}^{M} (\lambda_i / \gamma) \quad (1 / \sqrt{\lambda_i \mu \beta / \gamma\, d_i}\,) = T_{max}$$

$$\sqrt{\mu\beta/\gamma} = (1 / \gamma\, T_{max}) * \sum_{i=1}^{M} \sqrt{\lambda_i\, d_i}$$

D'où :

$$x_i = \frac{\lambda_i}{\mu} + \left(\frac{\lambda_i}{\mu\, \gamma\, T_{max}}\right) \frac{\sum_{j=1}^{M} \sqrt{\lambda_j\, d_j}}{\sqrt{\lambda_i\, d_i}}$$

Prolongée exponentielle.

Le problème est le suivant :

Trouver le M-uple $(x_1,\ldots,x_M)$ de R^M

qui minimise $D = \sum_{i=1}^{M} d_i \, x_i^{\alpha}$

sous la contrainte

$$\sum_{i=1}^{M} (\lambda_i / \gamma) \quad (1 / (\mu x_i - \lambda_i)) \leq T_{max}$$

où d_i, α, λ_i, γ, μ et T_{max} sont des constantes positives (pour $i \in [1,M]$)

Solution de ce problème :

La fonction considérée dans la contrainte étant décroissante par rapport à x_i ($i \in [1,M]$), l'inégalité dans la contrainte peut être remplacée par une égalité.

Considérons le Lagrangien :

$$G = \sum_{i=1}^{M} d_i \, x_i^{\alpha} + \beta \left(\left\{ \sum_{i=1}^{M} (\lambda_i / \gamma) \quad (1 / (\mu x_i - \lambda_i)) \right\} - T_{max} \right)$$

Les M équations

$$\frac{\partial G}{\partial x_i} = 0 \text{ pour } i \in [1,M]$$

donnent

$$0 = d_i \alpha \, x_i^{\alpha-1} - \beta (\lambda_i / \gamma) \quad (\mu / (\mu x_i - \lambda_i)^2)$$

$$(\mu x_i - \lambda_i)^2 = (\lambda_i \mu \beta / \gamma \, d_i) \quad (x_i^{1-\alpha} / \alpha)$$

$$x_i - \frac{\lambda_i}{\mu} - \sqrt{\frac{\beta \lambda_i}{\gamma \mu \alpha \, d_i}} * x_i^{(1-\alpha)/2} = 0$$

Calculons la valeur de la constante β :

$$\sum_{i=1}^{M} (\lambda_i / \gamma) \quad (1 / (\mu x_i - \lambda_i)) = T_{max}$$

$$\sum_{i=1}^{M} (\lambda_i / \gamma) \sqrt{\gamma \alpha \, d_i / \beta \lambda_i \mu} \quad x_i^{(\alpha-1)/2} = T_{max}$$

$$\sqrt{\beta/\gamma\mu^{\alpha}} = (1 / \gamma\, T_{max}\mu) * \sum_{i=1}^{M} \sqrt{\lambda_i\, d_i}\; x_i^{(\alpha-1)/2}$$

D'où :

$$x_i = \frac{\lambda_i}{\mu} + \frac{\lambda_i}{\mu\gamma\, T_{max}} \frac{\sum_{j=1}^{M} \sqrt{\lambda_j\, d_j}\; x_j^{(\alpha-1)/2}}{\sqrt{\lambda_i\, d_i}}\; x_i^{(1-\alpha)/2}$$

Voici une ébauche de la résolution de cette équation :
Considérons les suites x_{in} ($i \in [1,M]$) définies par

$$x_{i1} = \lambda_i / \mu$$

$$x_{i(n+1)} = \frac{\lambda_i}{\mu} + \frac{\lambda_i}{\mu\gamma\, T_{max}} \frac{\sum_{j=1}^{M} \sqrt{\lambda_j\, d_j}\; x_{jn}^{(\alpha-1)/2}}{\sqrt{\lambda_i\, d_i}}\; x_{in}^{(1-\alpha)/2} \quad (1)$$

Si les suites x_{in} (pour $i \in [1,M]$) convergent vers une valeur finie x_{io} alors $(x_{10},\ldots,x_{M0})$ est solution de l'équation étudiée. En effet l'égalité (1) étant vraie pour tout n, elle reste vraie à la limite, lorsque n tend vers $+\infty$. Il suffit donc de calculer les différentes valeurs de x_{in} (pour $i \in [1,M]$ et pour n suffisamment grand) (c'est possible si les constantes du problème sont numériques) et de tester si ces valeurs convergent vers une limite finie.