Université des Sciences et de la Technologie
HOUARI BOUMEDIENNE (U.S.T.H.B.) ALGER

——«o0o»——

INSTITUT D'INFORMATIQUE

——«o0o»——

THESE

PRESENTE A L'USTHB POUR L'OBTENTION DU GRADE DE MAGISTER EN INFORMATIQUE

Par : Mr BOUKALA M. CHERIF

PARALLELISATION DES ALGORITHMES BRANCH AN D BOUND

Soutenue publiquement le 13 Février 1991 Devant le JURY Commposé de:

Mr BENYELLES C.B. Professeur USTHB ..... Président

Mr AINOUCHE A. Profeseur USTHB ..... Rapporteur

Mr KHELLADI A. Professeur USTHB ..... Examinateur

Mme ABDERRAHIM A. Chargé de Cours USTHB .... Examinateur

Mr BADACHE N. Chargé de Cours USTHB ..... Examinateur

SOMMAIRE

certains paramètres, relatifs au système distribué (topologie) et aux critères sur lesquels est basée la méthode B&B (fonction d'évaluation).

Notre thèse est composée de six chapitres :

Dans le **chapitre** I, nous introduisons d'abord le calcul parallèle ainsi que le modèle théorique standard : la PRAM. Nous donnons ensuite quelques définitions et concepts utilisés. Nous avons spécialement introduit le concept de processus, défini de manière générale les réseaux et donné les principales structures des algorithmes parallèles (structure synchrone, asynchrone et maitre-esclave). Nous donnons aussi un bref aperçu sur le problème de la détection de la terminaison dans les algorithmes distribués et enfin, nous nous intéressons à l'apport des machines parallèles en optimisation combinatoire.

Le **chapitre** II est consacré à la description des méthodes de recherche arborescente. Celles-ci se basent sur une exploration d'une arborescence. Dans ce même chapitre, sera présentée de façon détaillée la méthode B&B, cas particulier de ces méthodes.

L'implémentation séquentielle des méthodes B&B est décrite dans le **chapitre** III. Leur complexité en temps et en espace sont également définies, mettant en évidence l'influence de la fonction d'évaluation et de la stratégie de recherche. Nous présentons aussi dans ce chapitre un modèle théorique : le modèle probabiliste de Wah-Yu, qui permet la recherche d'une complexité moyenne en temps et en espace.

Dans le **chapitre** IV, nous exposons des schémas généraux de construction des algorithmes B&B distribués. Dans le premier, dit synchrone (maitre esclave), chaque processus (esclave) explore un seul sommet de l'arborescence chaque fois qu'il lui est confié par un processus particulier (maitre) gérant le développement de l'arborescence. Tandis que dans le deuxième, dit asynchrone, les processus jouent le même rôle et développent chacun une partie de l'arborescence. Ce dernier schéma est le mieux adapté à l'implémentation sur un réseau, car le nombre de messages y est réduit. Nous formalisons ce schéma, d'une façon générale, permettant ainsi de l'adapter à tout problème d'optimisation combinatoire, à l'aide d'un langage algorithmique, et donnons les mécanismes permettant la terminaison de ces algorithmes. Nous proposons ensuite une optimisation du nombre de messages, vue son influence sur le temps d'exécution.

Quant aux performances des algorithmes B&B parallèles, elles sont étudiées au **chapitre V**, où une nouvelle notion propre aux algorithmes parallèles, l'accélération, est introduite. On pourrait penser que l'augmentation du nombre de processeurs accélérerait la recherche de la solution optimale, mais il n'en n'est pas toujours ainsi. Des anomalies peuvent survenir : on peut trouver la solution dix fois plus vite avec seulement trois processeurs qu'avec un seul processeur (anomalie favorable), comme on peut la trouver en un temps plus grand en utilisant plusieurs processeurs (anomalie défavorable). Des conditions d'existence de telles anomalies sont exposées dans ce même chapitre.

Enfin, les résultats et les tests expérimentaux de l'implémentation d'algorithmes B&B distribués pour le problème du voyageur de commerce sont exposés et commentés au **chapitre VI**.

CHAPITRE I

LE CALCUL PARALLELE

I.1 INTRODUCTION

L'apparition de machines parallèles, si elles permettent dans la plupart des cas, de substanciels gains de performance, posent un problème majeur : le séquencement des opérations dans le temps n'est plus respecté. Il devient alors impératif de développer une nouvelle algorithmique adaptée à cette réalité. Cette nouvelle manière de conception introduit à son tour de nouveaux concepts. Dans ce qui suit, il sera donné un aperçu sur ces différents concepts.

I.2 CALCUL PARALLELE ET CALCUL DISTRIBUE

Le calcul parallèle peut être considérée comme la coopération d'un ensemble de processus exécutant chacun un algorithme séquentiel qui lui est propre (éventuellement le même) et se communiquant des informations, par échange de messages ou par accés à une mémoire commune.
Motivée par l'existence de réseaux d'ordinateurs et l'apparition de machines composées de processeurs inter-communicants, la démarche algorithmique nécessaire est nouvelle et ne peut s'inspirer que partiellement de l'important savoir faire accumulé jusqu'à présent qui repose essentiellement sur la séquentialité des opérations d'un algorithme et l'existence d'une mémoire dans laquelle sont centralisées les donneés du problème à traiter [RAY 85].
Le problème majeur va être de pouvoir identifier des processus de traitements élémentaires qui vont coopérer à la réalisation d'une tâche commune mais ne pouvant à un instant donné, obtenir d'informations que sur l'évolution antérieure de leurs partenaires et non sur leur situation à cet instant. En effet, le caractère asynchrone des systèmes concernés, et l'absence de

mémoire partagée pour les réseaux distribués fait qu'il est impossible à un processus d'observer un état global et instantané des autres processus .
Une autre caractéristique de ces informations est qu'elles sont obtenues par échange de messages. Un algorithme distribué est donc défini comme un ensemble de processus qui, communiquant exlusivement par envoi de messages, coopèrent à la réalisation d'un but commun.
Le modèle théorique standard pour le calcul parallèle est la PRAM (parallel random access machine [AHO 74]). Brièvement, elle peut être définie comme une machine, avec un potentiel infini de processeurs identiques et une mémoire partagée de taille infinie. Une variante généralement utilisée de ce modèle est la CREW PRAM (concurrent read exclusive write) où les lectures simultanées sont permises, mais les écritures simultanées sont interdites.
Le calcul dans ce type de machine synchrone commence dès qu'un processeur est activé : à chaque pas, un processeur actif effectue une opération standard ou active un autre processeur. Le calcul s'arrête lorsque le processeur initial s'arrête. Les ralentissements dûs aux conflits d'accès mémoire sont négligés. Mais ce modèle reste tout de même pas très réaliste [LAV 86].

I.3 LE CONCEPT DE PROCESSUS

Un processus est l'activité résultante de l'exécution d'un programme et de ses données par un processeur séquentiel (machine de Turing par exemple), il peut éventuellement être muni de primitives de communications lui permettant d'émettre et de recevoir des messages. Dans un contexte parallèle, informellement, on considère un processus comme étant l'unité d'exécution élémentaire d'un algorithme parallèle.

I.4 LE RESEAU

Le medium de communications entre processus est le réseau, entité physique permettant de connecter des processus entre eux et capable de véhiculer des informations échangées par ces derniers.
On peut associer au réseau le concept de graphe de communications, graphe représentant les potentialités de communication de processus à processus. Si on note P, l'ensemble

des processus mis en oeuvre dans un algorithme distribué, on notera alors :

$$G = (P,L),\ L \subseteq P \times P,$$

le graphe de communications. Ainsi $(p_1,p_2) \in L$, signifie que le processus p_1 peut communiquer avec le processus p_2.
Une arête (p_1,p_2) (ou éventuellement un arc) de G peut représenter soit une possibilité physique de communications entre les processus exécutant p_1 et p_2. Dans ce cas l'arête (ou éventuellement l'arc) schématise l'existence par exemple d'une ligne bi-directionnelle (ou éventuellement uni-directionnelle) ou simplement une possibilité virtuelle de communication entre les processus p_1 et p_2 indépendamment des moyens physiques mis en oeuvre pour accomplir la transmission afférente.
Parmi les configurations ou topologies particulières de réseaux, on trouve l'anneau (bidirectionnel, unidiretionnel), l'étoile, l'arborescence. Dans ces deux derniers cas, un noeud particulier (centre de l'étoile, racine de l'arborescence) joue un rôle privilégié et des fonctions d'allocation lui sont en général dévolues. Un réseau, dans lequel chaque noeud est relié à tous les autres est appelé réseau complet.

I.5 LES STRUCTURES PARALLELES

La caractéristique la plus importante du calcul parallèle est sa structure représentant la façon avec laquelle les différents composants sont reliés entre eux. Elle représente la structure logique des tâches effectuant le calcul, on peut citer les structures synchrones, asynchrones, maitre-esclaves, pipeline, ...

I.5.a Les structures synchrones

Les processus travaillent de façon synchronisée, c'est à dire qu'ils exécutent leurs tâches en un "lock step" (blocage en attente), en utilisant un mécanisme de synchronisation explicite. Tous les processus ont les mêmes autorités et contrôles. Nous pouvons représenter cette structure par le graphe des tâches suivant :

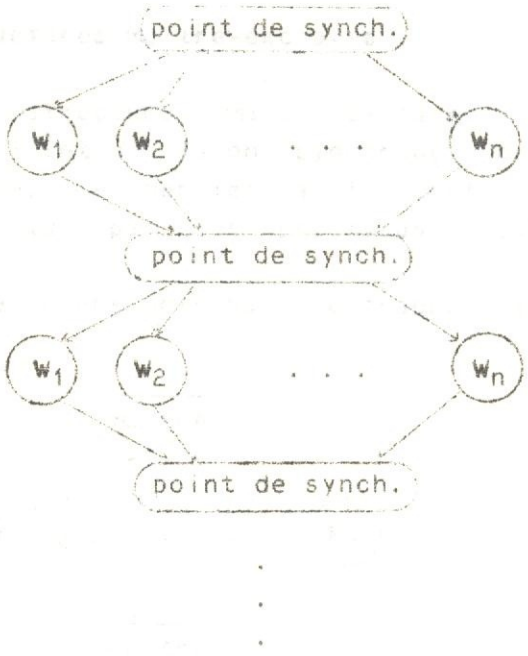


fig.1. La structure synchrone.

où les W_i représentent une séquence de calcul, par le processeur i, entre deux points de synchronisation.
Cette structure est la moins utilisée en pratique, elle entraine une inactivité plus grande des processeurs physiques dûe au fait que ces derniers attendent les points de synchronisation en étant inactifs. De plus si le code exécuté par les processus est le même, le temps de conflits augmente encore leur temps d'inactivité. Cependant cette structure est considérée comme un cas particulier de la structure maitre-esclave.

I.5.b La structure asynchrone

Dans la structure asynchrone, les processus travaillent indépendamment les uns des autres. Tous ces processus ont la même et égale autorité et contrôle. La coopération entre les processus est effectuée par partage de données ou par envoi de messages, cela suppose que de telles communications ne provoquent pas une dépendance entre les tâches, mais seulement des attentes et des conflits pour l'accès aux ressources partagées.
Les tâches exécutées par les processus peuvent être considérées comme étant indépendantes, sans contraintes de précédence.
Généralement, cette structure est la plus performante.

I.5.c Les structures maitre-esclaves

Dans la structure maitre-esclave, plusieurs processus esclaves travaillent de façon synchrone sous le contrôle du processus maitre. Ce dernier a le contrôle absolu sur les traitements effectués par les processus esclaves durant leurs activités.
Ceci peut être représenté par le graphe des tâches suivant :

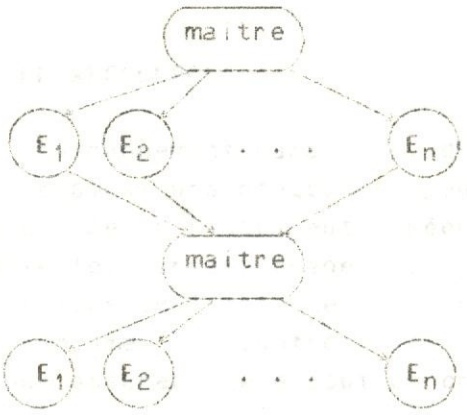


fig.2. Le graphe des tâches de la structure maitre-esclave.

Chaque itération contient une tâche maitre puis les tâches esclaves E_i en parallèle. On remarque alors que lorsque le processus maitre est actif, les processus esclaves sont inactifs, ce qui diminue le degré de parallélisme et par conséquent les performances de cette structure.

I.6 LES COMMUNICATIONS INTER-PROCESSUS

Dans un algorithme parallèle, les processus se communiquent des informations. Dans certains cas, cela pourrait être uniquement l'envoi du résultat final au processus maitre . Dans d'autres, des communications plus fréquentes servent à guider les processus pour atteindre plus vite la solution.

Une instance de communication consiste en l'envoi d'informations d'un processus à un autre, celles-ci peuvent être des données ou des informations de contrôle. L'itinéraire de ces informations peut influencer sur les performances de l'algorithme.
Le passage de messages peut être fait principalement de deux manières différentes: en mode asynchrone, permettant au processus en communication de reprendre le contrôle dès que les primitives de communication sont exécutées et en mode synchrone, où le processus qui envoie un message est bloqué jusqu'à ce que le récepteur ait accusé réception. Ce mode permet non seulement l'envoi de messages mais aussi la synchronisation entre les processus.

I.7 PARTAGE DE RESSOURCES

Dans un environnement parallèle, des ressources peuvent être partagées entre plusieurs processus pour l'accomplissement d'une certaine tâche. Ce partage peut créer des conflits entre les processus (attente, interblocage, ...), ainsi un processus peut rester inactif lors de l'attente d'une ressource utilisée par un autre. Un mécanisme de contrôle d'accès aux ressources sera nécessaire pour assurer une exécution correcte du programme.

I.8 DETECTION DE LA TERMINAISON D'ALGORITHMES DISTRIBUES

En algorithmique distribuée, la détection de la terminaison est une préoccupation majeure. Elle recouvre en fait deux problèmes :

a. Détecter si tous les processus sont à l'arrêt.
b. S'assurer que le calcul demandé a été bien accompli.

Résoudre ceci, consiste à permettre à tout processus de détecter qu'il a "bien" terminé, et que tous les autres ont atteint cet état.
Plusieurs solutions ont été proposées pour la résolution de ces problèmes. Elles se basent toutes sur une structuration de l'ensemble des processus de façon à pouvoir visiter chacun des processus du calcul distribué et vérifier la propriété de terminaison. Les structures de base utilisées pour effectuer le parcours du graphe de communications sont l'anneau unidirectionnel, le circuit hamiltonien ou encore le circuit eulérien ou également l'arbre couvrant sur le graphe associé au réseau. Le mode de parcours est fonction de la topologie de

contrôle. Ainsi, si cette topologie est un arbre couvrant, le mode de parcours utilisé est celui du calcul diffusant dont le principe a été introduit par Dijkstra et Scholten. Lorsque la topologie de contrôle est un anneau (ou un circuit) de processus, on utilise plutôt un jeton circulant.
Il existe beaucoup d'autres modes de parcours que nous ne citerons pas, basés tous pricipalement sur la topologie de contrôle et sur la structure de l'algorithme distribué (structure maitre-esclave, synchrone, asynchrone, ...) [BLA 88].

I.9 OPTIMISATION COMBINATOIRE ET ALGORITHMES PARALLELES

Motivés par l'existence de réseaux d'ordinateurs et l'apparition de machines parallèles, les chercheurs en optimisation combinatoire, ont donné de nombreux algorithmes mettant à profit le mieux possible les caractéristiques de telles machines.
La question que l'on se pose lorsque nous utilisons une machine parallèle ou un système distribué est : "peut-on résoudre des problèmes que l'on ne pouvait résoudre en séquentiel ? - La complexité des problèmes change-t-elle ?"

I.9.a Problèmes d'optimisation combinatoire difficile

Dans les problèmes d'optimisation combinatoire, NP-complets, la plupart des méthodes de résolution n'échappent pas à une énumération de l'ensemble des solutions. Même si celle-ci est judicieusement limitée (méthodes d'énumération implicite telles que B&B, ...), l'idée d'utiliser des machines parallèles s'est vite imposée. En effet, en séquentiel, une telle énumération s'arrête vite faute de place mémoire ou de temps devenant prohibitif, même pour des problèmes de taille raisonnable.
Des machines à parallélisme local pourraient offrir une accélération du calcul de l'évaluation de telles procédures mais pour de nombreuses instances de problèmes, le gain est faible.
L'emploi de machines offrant un parallélisme massif (ou à parallélisme global) s'avère plus intéressant : des parties entières de l'arborescence des solutions pourraient être énumérées en parallèle.

I.9.b Complexité [LAV 86] [ROU 87]

Dans le calcul parallèle, beaucoup de problèmes dans P (classe des problèmes polynômiaux) peuvent être résolus en un temps polylog : $(\log)^{O(1)}$, c'est à dire en un temps borné par un polynôme en logarithme de la taille n du problème. Certains problèmes faciles deviennent très faciles : par exemple, tri en parallèle en $O(\log n)$ avec $O(n^2/\log n)$ processeurs, ...
D'autres problèmes de P n'admettent pas de solution en parallèle en un temps polylog, ils sont dits P-complets. Cette classe regroupe donc les problèmes les plus difficiles de P : toute solution va requérir un temps "super logarithmique", ce qui provient sans doute du fait qu'ils sont de nature intrinsèquement séquentielle.
Mais ces résultats sont en fait obtenus pour des modèles théoriques simplificateurs de machines existantes (CREW PRAM suppose un nombre infini de processeurs et une mémoire partagée de taille infinie).
De nouveaux résultats fondés sur des hypothèses plus réalistes ont été définis. Dans le cas du calcul distribué, il a été prouvé que les problèmes faciles restent faciles (dans la classe P), les problèmes difficiles restent également difficiles (P-complet ou NP-complet) en égard à la complexité en communication (nombre de messages) et à la complexité en temps [LAV 86]. La complexité en espace pour un algorithme donné peut être également prise en compte.
Pour les machines multiprocesseurs commercialisées, on s'intéresse essentiellement à l'accélération obtenue en passant d'une machine à un processeur, à une machine à plusieurs processeurs.

CHAPITRE II

LES METHODES DE RECHERCHE ARBORESCENTE ET LES ALGORITHMES B&B

II.1 INTRODUCTION

Plusieurs problèmes d'optimisation, de décision, ... n'ont pas d'algorithme de résolution directe, ou alors s'il en existe, ils sont inefficaces (du point de vue temps et espace mémoire nécessaires à l'exécution, ainsi que de l'exactitude de la solution). Ces problèmes n'échappent pas à l'énumération de toutes les solutions, méthode toutefois impraticable (coûteuse). Dans ce chapitre, nous décrivons les méthodes de recherche arborescente qui permettent d'éviter l'exploration de toutes les solutions. L'ensemble des solutions est représenté sous forme d'une arborescence, la recherche de la solution optimale revient à parcourir (judicieusement) cette arborescence.

II.2 METHODE DE RECHERCHE ARBORESCENTE

La méthode de recherche arborescente a été découverte et utilisée par plusieurs chercheurs. Son nom (Backtracking) lui a été attribué par D.H.Lehmer de l'Université de Californie à Berkeley [BAU 65]. Cette méthode plutôt universelle peut être décrite de la manière suivante :
On veut déterminer un vecteur où tous les vecteurs $(x_1, x_2, .. x_n)$ de S, (S étant le produit cartésien $X_1 x X_2 ... X_n$) satisfaisant à une fonction objective $\bar{Q}(x_1, x_2, ... x_n)$, chaque X_i est effectivement énumérable.
L'approche brute serait de former tous les vecteurs possibles de S, d'évaluer chacun à l'aide de la fonction objective $\bar{Q}$ et voir celui qui satisfait notre objectif. Mais le nombre de possibilités est immense (dans le problème des huit reines par exemple, il y a C_{64} configurations possibles, quelques 4.4 billions!). La méthode de recherche arborescente a été conçue

pour donner la même solution mais avec beaucoup moins d'essais. L'idée de base consiste en la construction progressive (un composant à la fois) du vecteur et en utilisant une fonction objective modifiée pour tester si le vecteur en construction a une chance ou pas de satisfaire l'objectif du problème. Cela permet d'écarter plusieurs possibilités et réduire ainsi, l'ensemble de recherche. Schématiquement parlant, cela revient à la construction d'une arborescence. A chaque étape, on prend un noeud de l'arborescence (correspondant au vecteur construit jusque-là) et on le relie à ses successeurs qui correspondent au vecteur père augmenté chacun d'une composante. Les noeuds n'ayant plus de chance de satisfaire l'objectif ne seront plus décomposés. Quant aux autres, on continue à les décomposer jusqu'à l'obtention de la solution voulue. Plus de détails sur la méthode et des exemples (problème classique des huit reines, ...) se trouvent dans [BAU 65].
Une autre description de la méthode et ses applications en intelligence artificielle ainsi que dans d'autres domaines (jeux, analyse syntaxique, ...) se trouvent dans [NIL 82], On y trouve aussi la description des arbres de décision ET/OU et leurs utilisations ainsi que les procédures MINMAX, ALPHA-BETA utilisées dans la recherche de stratégie de jeux (tic-tac-toe, ...). Toutes ces procédures se basent sur le même principe, c'est à dire l'exploration d'une arborescence et posent donc les mêmes problèmes d'explosion combinatoire de l'espace et du temps requis pour la recherche de la solution ou du but.

II.3 PROBLEME A RESOUDRE

Un problème d'optimisation combinatoire, de façon générale, peut être formulé de la manière suivante :
Etant donné un ensemble des solutions réalisables S etant effectivement énumérable, f une fonction associant à chaque élément de S une valeur réelle (son coût) .
Il s'agit de déterminer une solution $\hat{u}$ telle que :

$$f(\hat{u}) = \min_{x \in S} f(x)$$

$\hat{u}$ est la solution optimale du problème.
Il est toujours possible de résoudre de tels problèmes en énumérant toutes les solutions (les éléments) de S, et pour effectuer cette énumération de façon systématique, il est

préférable de décomposer progressivement l'ensemble des solutions potentielles S. Cette décomposition est faite sous forme d'arborescence, l'ensemble S correspondra à la racine. Une étape de la décomposition consiste à choisir un sous-ensemble S' de S, correspondant à un sommet pendant de l'arborescence, et à le séparer en sous-ensembles $S'_1, S'_2, ..S'_K$, le sommet correspondant à S' dans l'arborescence est alors joint aux sommets nouvellement crées et qui correspondent aux sous-ensembles $S'_1, S'_2, ..S'_K$. Pratiquement, cette méthode est inefficace, mais elle peut être grandement améliorée si nous avions un moyen nous permettant d'éviter l'examen de toutes les parties en lesquelles l'ensemble des solutions est décomposé.
Dans la méthode B&B, c'est la fonction d'évaluation qui nous permet d'éliminer des branches complètes de l'arborescence et de localiser (de façon que l'on qualifie d'intelligente) la solution optimale.

II.4 DESCRIPTION GENERALE DES METHODES B&B [SAK 84]

Ces algorithmes reposent sur trois notions clés :

. **Séparation :**
l'ensemble des solutions est décomposé successivement en des sous-ensembles de taille de plus en plus réduite (en vue d'aboutir soit à un sous-ensemble ne contenant qu'une seule solution ou à un sous-ensemble vide).

. **Evaluation :**
pour chaque sous-ensemble crée par séparation, l'évaluation permet d'avoir une idée de la valeur des solutions lui appartenant.

. **Stratégie de recherche :**
c'est la règle de sélection du sous-ensemble de solutions que l'on va décomposer (séparer).

L'efficacité des algorithmes bâtis selon ce schéma dépend étroitement de la manière choisie pour procéder à la séparation et surtout à l'évaluation.

II.4.a PRINCIPE DE SEPARATION

L'ensemble des solutions réalisables S est inclus dans un autre ensemble U, U est par exemple l'ensemble des solutions du problème relaxé [SAK 84], S inclus dans U. U et S sont respectivement appelés ensemble des solutions et ensemble des solutions réalisables.
Le principe de séparation consiste à décomposer l'ensemble U des solutions en sous-ensembles de plus en plus réduits, jusqu'à ce que l'on aboutisse à des sous-ensembles qui sont soit vides, soit réduits à une solution réalisable unique ou ne pouvant pas contenir de solutions meilleures que celle déja connue.

La séparation devra vérifier les trois conditions suivantes :

1. La réunion des sous-ensembles obtenus lors d'une séparation doit être égale à l'ensemble séparé.

2. Le nombre total de séparations nécessaires pour atteindre une solution optimale est fini.

3. Lorsqu'un sous-ensemble de solutions ne peut plus être séparé, il doit être alors possible de prouver que cet ensemble :
   - ne contient plus de solution réalisable.
   - est réduit à une solution réalisable.
   - ne peut contenir de solution meilleure que celle déja connue.

Une telle procédure peut être illustrée par une arborescence, comme le montre le schéma suivant, dont la racine correspond à l'ensemble de solutions S, chaque noeud correspond à un sous-ensemble de solutions, les successeurs d'un noeud correspondent aux sous-ensembles crées lors de la séparation de ce dernier.

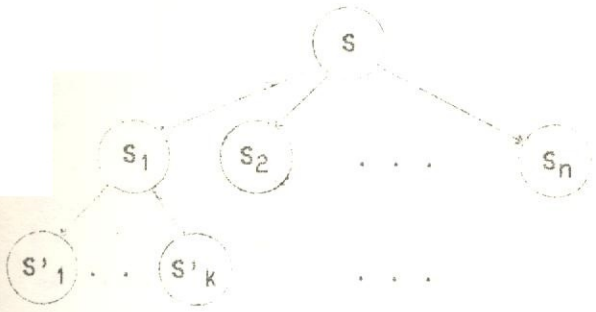


fig. 3. Arborescence de recherche.

Les feuilles représentent donc soit une solution réalisable, soit un sous-ensemble ne contenant pas de solutions réalisables ou un sous-ensemble ne pouvant pas contenir une solution meilleure que celle déja trouvée.

Dans ce qui suit, nous confondrons le sommet de l'arborescence et le sous-ensemble ou le sous-problème qu'il représente.

Remarque

Le principe de séparation n'impose pas de contraintes sur le nombre de successeurs d'un sommet, sauf que celui-ci doit être fini. S'il est égal à deux, la séparation est dite **dichotomique**, et elle est dite **polytomique** lorsque ce nombre est supérieur à deux.

II.4.b FONCTION D'EVALUATION

Cette fonction associée à chaque sous-ensemble (noeud de l'arborescence) construit par séparation une valeur appréciant par défaut la meilleure des solutions appartenant au sous-ensemble considéré; soit v cette fonction, elle doit vérifier les propriétés :

1. Si S_K est un sommet associé à un sous-ensemble de solutions, alors $v(S_K) \leq \min f(s)$, $s \in S_K$

2. Si S_K est un sommet terminal tel que $S_K = \{ s_K \}$ alors $v(S_K) = f(s_K)$

3. Si S_K ne contient pas de solutions réalisables, on posera $v(S_K) = +\infty$

4. Si S_K est séparé en S_{K+1}, S_{K+2}, ... S_{K+l} alors
$v(S_{K+i}) \geq v(S_K) \quad 1 \leq i \leq l$
les valuations croissent de la racine vers les feuilles de l'arborescence .

5. Supposons que l'on connaisse une borne supérieure, BS par exemple, qui corresponde à une solution réalisable du problème, obtenue soit parce qu'un sommet terminal est atteint soit qu'une solution approchée aurait été trouvée par une heuristique. Cette borne nous permettra d'affirmer qu'un ensemble S_K ne peut pas contenir de solution meilleure que celle déja trouvée si son évaluation est supérieure ou égal à BS
$v(S_K) \geq BS$ donc $\forall x \in S_K \quad f(x) \geq BS$
Cette propriété nous permet de ne pas continuer l'exploration de ce sous-ensemble.

II.4.c STRATEGIES DE PARCOURS DE L'ARBORESCENCE

C'est la règle qui permet de sélectionner le prochain sommet à séparer. Les stratégies les plus classiques sont les suivantes:

Meilleur d'abord (best first)
Sélectionner le sommet dont l'évaluation est la plus faible avec l'idée que l'ensemble associé a plus de chance de contenir la solution optimale.

Profondeur d'abord (depth first)
Choisir l'ensemble le plus récemment crée. Dans cette stratégie, on parcourt une branche jusqu'à isoler un sommet terminal puis on remonte au niveau supérieur et on réitère.

Largeur d'abord (breadth first)
Le sommet choisi est celui le plus anciennement crée ou de plus haut niveau dans l'arborescence.

Remarques

- D'autres stratégies peuvent être également utilisées (une combinaison des stratégies meilleur d'abord et profondeur d'abord, stratégie aléatoire, ...).

- Si les noeuds de l'arborescence sont maintenus dans une liste, on peut associer une priorité à chacun. Soit $h(S_i)$ la priorité associée au noeud S_i :
Pour la stratégie meilleur d'abord : $h(S_i) = v(S_i)$.
Pour la stratégie largeur d'abord : $h(S_i) = l(S_i)$, où l est la profondeur de sommet S_i dans l'arborescence.
Pour la stratégie profondeur d'abord : $h(S_i) = -l(S_i)$

CHAPITRE III

IMPLEMENTATION ET PERFORMANCES DES ALGORITHMES B&B (en sequentiel)

Dans ce chapitre, il sera présenté l'implémentation séquentielle des algorithmes B&B, l'influence des critères sur lesquels est basé la méthode B&B sur les performances en temps et en espace et enfin un apercu sur le modèle probabiliste de Wah et Yu qui permet la recherche de l'espérance de mesures de performance.

Définition

Un sommet S est dit actif si le coût v(S) de ce sommet est inférieur à la solution réalisable actuelle. Ce sommet est alors susceptible de contenir la solution optimale.

III.1 ALGORITHME B&B

On gère la liste des sommets actifs par une file F et on maintient également la meilleure solution courante. Une itération de l'algorithme consiste à, choisir un sommet critique, le séparer et évaluer tous ses successeurs, on dira alors que ce sommet a été exploré. Chacun des successeurs sera soit :

. Inséré dans la file, c'est à dire rajouté à l'ensemble des sommets actifs, et ceci, si son coût est inférieur à la solution réalisable trouvée jusque-là, et que lui même n'en soit pas une.

. Eliminé si son coût est supérieur à la meilleure solution réalisable courante .

Si l'un des successeurs représente une solution réalisable alors il faut améliorer éventuellement la solution réalisable courante et mettre à jour la file des sommets actifs F.

L'algorithme se termine lorsque la file des sommets actifs sera vide, ou en terme d'arborescence, lorsque tous les sommets pendants auront une évaluation supérieure ou égale à la meilleure solution courante, celle-ci sera la solution optimale du problème traité.

III.2 SPECIFICATION DE L'ALGORITHME

```
évaluer S   ;                    (* S étant la racine *)
si S est une solution réalisable
   alors fin
   sinon
        insérer S dans F  ;    (* F la file des sommets actifs *)
        meilleure solution courante = +∞;
        tant que F est non vide
        faire
             choisir S' de F   ;          (* exploration de S *)
             F = F \ { S'}     ;
             séparer S' en  S'1, S'2 , ... S'K   ;
             pour chacun des S'i
             faire
                  évaluer S'i   ;
                  si v(S'i) <  meilleure solution
                     alors
                          si S'i est une solution réalisable
                             alors
                                  meilleure solution = v(S'i)
                                  (* améliorer la solution
                                     réalisable  *)

                             sinon
                                  insérer S'i dans F
                                  (* S'i est un sommet actif *)
                          fsi
                     sinon S'i sommet éliminé  ;
                           (* S'i ne peut contenir la sol. opt.*)
                  fsi
             fait
        fait
fsi
fin .
```

Remarques

- Les procédures d'insertion et de choix de sommets actifs dépendent de la stratégie adoptée pour le parcours de l'arborescence.

- La procédure (ou fonction) d'évaluation dépend du problème que l'on est en train de résoudre (par exemple, on utilise le 1-arbre, l'affectation, . . . comme fonction d'évaluation pour le problème du voyageur de commerce) .

Une présentation très intuitive utilisée par Wah et Yu [WAH 82] pour un modèle probabiliste permet de modéliser le fonctionnement de cet algorithme.
Le modèle consiste en deux murs, le mur du haut représente le sommet actif de plus faible coût (stratégie meilleur d'abord), le mur du bas, la meilleure solution connue . Initialement le mur représentant la meilleure solution est à l'infini et monte dès que l'on trouve une solution réalisable meilleure, et dès qu'un sommet est exploré, le mur du haut prend la position correspondante au prochain sommet actif de plus petit coût.

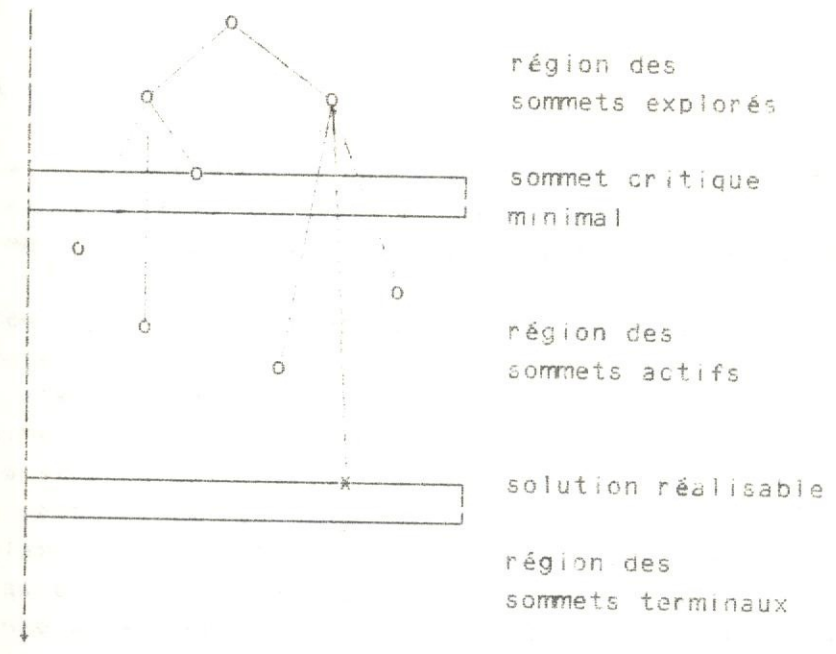


fig. 4. Modèle de Wah et Yu [WAH 82]

L'algorithme sera fini lorsque le mur du haut coincide avec le mur du bas ou ne peut prendre de position qu'après celle du mur du bas.

III.3 PERFORMANCES DES ALGORITHMES B&B SEQUENTIELS

Dans les années 70, quelques auteurs (particulièrement Kohler et Steiglitz [KOH 74], Ibaraki [IBA 76], ...) avaient essayé de définir théoriquement les performances des algorithmes B&B. Depuis, des expériences et des tests sur ces algorithmes ont été menés (Brown et Purdon [BRO 81], ...).

Dans ce qui suit, nous donnons une idée sur la complexité en temps et en espace et voyons que le temps que l'on met pour trouver une solution et prouver qu'elle est optimale et l'espace mémoire requis dépendent des trois notions clés sur lesquelles sont basés les algorithmes B&B : critère de séparation, fonction d'évaluation et la stratégie de sélection. Nous allons donner notamment quelques résultats connus, de façon à mettre en évidence les notions importantes et pouvoir expliquer par la suite les anomalies du comportement des algoritmes B&B parallèles.

III.4 COMPLEXITE EN ESPACE

La complexité en espace d'un algorithme B&B dépend directement de la taille maximale de la liste des sommets actifs. Celle-ci à son tour dépend principalement de la stratégie de sélection adoptée.
Pour la stratégie profondeur d'abord, la taille maximale de la liste des sommets actifs sera linéairement proportionnelle à la profondeur de l'arborescence, cette dernière est elle même une fonction linéaire de la taille du problème.
Pour le cas des stratégies meilleur d'abord et largeur d'abord, la taille de cette liste est une fonction exponentielle de la taille du problème. Cependant la stratégie meilleure d'abord est meilleure puisqu'elle ne génère que l'arborescence critique (qui va être définie ultérieurement), c'est à dire que les seuls sommets explorés sont ceux dont le coût est inférieur à la solution optimale.

Mais les stratégies profondeur ou largeur d'abord ont tendance à générer un nombre plus important de sommets, et ne permettent pas forcément d'obtenir la solution optimale plus rapidement (bien que dans le cas de la stratégie profondeur d'abord, ce sont les solutions réalisables du problème qui sont d'abord cherchées). L'amélioration de la fonction d'évaluation permet de réduire la taille de la liste des sommets actifs, cependant la taille de cette liste reste exponentielle à la taille du problème (voir § VI.1).

III.5 COMPLEXITE EN TEMPS D'UN ALGORITHME B&B [ROU 87]

Une idée de la complexité en temps d'un algorithme B&B peut être donnée à l'aide des deux grandeurs : T le temps total de l'algorithme et T^* le temps total pour trouver une solution optimale. Ces deux grandeurs dépendent à leur tour de :

NS : Le nombre total de sommets explorés .

NB : Le nombre de sommets séparés avant la dernière modification de la meilleure solution réalisable

Le temps total de l'algorithme (trouver une solution et prouver qu'elle est optimale) est estimé par :

$$T = O(t\ NS)$$

où t est le temps moyen requis pour explorer (c'est à dire séparer un sommet et évaluer ses successeurs) un sommet, il dépend beaucoup du problème considéré.
Le temps total pour obtenir une solution optimale (sans prouver qu'elle l'est) est :

$$T^* = O(t\ NB)$$

Cette mesure est assez importante, car l'algorithme peut souvent s'arrêter, faute de temps ou de place mémoire, avant que l'on ait prouvé que la dernière solution réalisable trouvée est optimale. L'expérience [ROU 86] montre que dans un algorithme B&B, pour la plupart des problèmes combinatoires classiques, la solution optimale est trouvée dans le premier quart du temps total, le temps restant étant passé à prouver l'optimalité de cette solution.

Définitions

Pour une instance d'un problème donné, un algorithme B&B de principe de séparation Γ et de fonction d'évaluation v, génère différentes arborescences selon la stratégie h employée. Pour un algorithme B&B (Γ, v, h), définissons les ensembles de sommets suivants :

S = l'ensemble de sommets de l'arborescence $B(\Gamma, v, h)$

$C = \{ S_i \in S \ / \ v(S_i) < f^* \}$ ensemble de sommets d'évaluation inférieure à celle de la solution optimale f^*, sommets dits critiques.

$C' = \{ S_i \in S \ / \ v(S_i) \leq f^* \}$

$R = \{ S_i \in S \ / \ v(Si) = f^* \}$

$Q = \{ S_i \in S \ / \ v(S_i) = f^* \text{ et } \Gamma(S_i) = \emptyset \}$ ensemble de sommets non terminaux d'évaluation égale au coût de la solution optimale.

$O = \{ S_i \in S \ / \ f(S_i) = f^* \}$

Seul C est indépendant de la stratégie h adoptée, tous les autres ensembles sont de cardinalité plus ou moins grande, suivant la stratégie utilisée.

Nous appelons **arborescence critique** C, la sous-arborescence correspondante aux sommets de C.

Une stratégie est dite optimale, si elle construit une arborescence de taille minimale pour trouver et prouver qu'une solution est optimale (elle minimise le nombre total NS de sommets explorés).

III.5.a BORNES INFERIEURES DES INDICATEURS DE COMPLEXITE EN TEMPS

Des bornes inférieures de NS et NB sont facilement obtenues:

Proposition 1

1. $NS \geq |C|$
2. $NB \geq \min \{ l(S_i) / S_i \in R \cap O \}$
   $l(S_i)$ niveau du sommet S_i dans B.

preuve

1. L'algorithme ne prend fin que lorsque les sommets pendants de l'arboresence ont une évaluation supérieure ou égale à f^*.

2. Le chemin jusqu'à une solution réalisable est mesuré par le niveau du sommet correspondant, c'est donc au moins le nombre de sommets qu'il faut explorer jusqu'à une solution optimale.

Trois propriétés peuvent caractériser une fonction d'évaluation :

Définition 1 (P1)

Une fonction d'évaluation est **discriminante**, si et seulement, si pour deux sommets disjoints S_i et S_j appartenant à $S \setminus O$ $v(S_i) = v(S_j)$ (il n'y a jamais égalité, donc ambiguité, sur les évaluations des sommets).

Définition 2 (P2)

Une fonction d'évaluation est dite **faible**, si l'évaluation de tout sommet non optimal est différent du coût de la solution optimale : Pour $S_i \in S \setminus O$, $v(S_i) = f^*$

Définition 3 (P3)

Une fonction d'évaluation v_1 est dite **plus pertinente** qu'une autre fonction v_2 définie pour le même problème, si et seulement si $\forall S_i \in S$; $v_2(S_i) < v_1(S_i) \leq f(S_i)$.

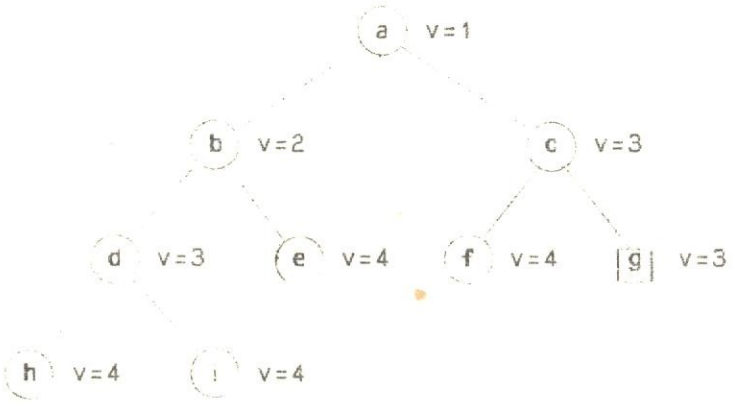
De ces propriétés découlent les résultats suivants :

Proposition 2

La stratégie meilleur d'abord est unique et optimale si la fonction d'évaluation est discriminante, sinon, il existe néanmoins une strategie meilleur d'abord qui soit optimale, et qui examine un nombre de sommets égal à $|C|+|Q|$.

Preuve : Simple d'après les définitions précédentes.

Sur l'exemple suivant, la stratégie meilleur d'abord h_1 qui explore les sommets a, b et c est optimale, alors que la stratégie meilleur d'abord h_2, qui explore a, b, d et c ne l'est pas.



Proposition 3

La stratégie meilleur d'abord ne génère que l'arborescence critique si la fonction d'évaluation est faible.

Preuve : Si v est faible $Q = \emptyset$. La stratégie meilleur d'abord génère par définition tous les sommets S_i tel que $v(S_i) < f^*$, donc uniquement ceux de C.

III.5.b BORNES DE LA COMPLEXITE EN TEMPS D'UN ALGORITHME B&B AVEC LA STRATEGIE MEILLEUR D'ABORD

Proposition 4

Si la fonction d'évaluation est discriminante alors

$$|C| < NS \leq |C'-R| \quad , \quad NB = NS.$$

Si la fonction d'évaluation est faible alors **NS = |C|**

Une démonstration, par Ibaraki est mentionnée dans [ROU 86].

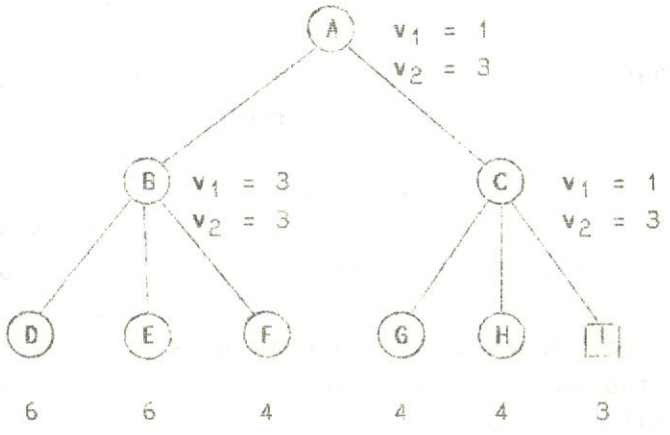
Propriété

Utiliser une fonction d'évaluation plus pertinente dans un algorithme B&B peut conduire à une augmentation du nombre de sommets explorés pour la stratégie meilleur d'abord, et provoquer une diminution de ce nombre pour les stratégies profondeur et largeur d'abord.

Preuve :

Pour la première partie, l'exemple ci-dessous en est une preuve, pour la suite voir [PAP 82].

exemple



v_2 plus pertinente que v_1.
Si la fonction d'évaluation appliquée est v_2, l'ensemble des sommets explorés peut être {A, B, C}, alors que si l'on applique v_1, cet ensemble sera réduit à {A, C} seulement, bien que v_2 soit plus pertinente que v_1.

III.6 MODELES PROBABILISTES

Les estimations faites dans le paragraphe précédent étaient fonction de paramètres qui eux-mêmes ne sont pas maitrisés, (tout ce que l'on peut dire en général, c'est qu'ils sont exponentiels à la taille du problème). Dans les modèles probabilistes, on cherche une complexité moyenne en temps et en espace (espérance mathématique de la valeur de NS).
Nous donnons un bref aperçu du modèle de Wah et Yu [WAH 82]. Il permet de mettre en évidence la liaison entre la pertinence d'une fonction d'évaluation, la distribution de la valeur des solutions réalisables du problème considéré et la stratégie utilisée.

Modèle probabiliste de Wah et Yu [WAH 82].

Les suppositions faites sur l'arborescence sont les suivantes :
. Le nombre de successeurs par sommet est fixé ou connu en moyenne.
. La hauteur h de l'arborescence est connue, tout sommet terminal a donc un niveau au plus égal à h.
. La variable aléatoire est la différence entre les évaluations d'un sommet père et fils.

Le modèle utilisé pour les algorithmes B&B est celui des deux murs, décrit précédemment. Les positions respectives de ces deux murs sont ensuite calculées en supposant que les différences entre les bornes inférieures d'un sommet et celles de ses successeurs crées par séparation sont des variables aléatoires indépendantes, identiquement distribuées suivant une fonction de densité gamma.
Ce modèle a été vérifié par simulation pour des problèmes tels que la programmation linéaire en nombres entiers, le problème du sac à dos et recouvrement. Il a servi également à l'étude de la stratégie meilleur d'abord et à la comparer à la stratégie profondeur d'abord.
L'analyse et les résultats expérimentaux montrent que le nombre moyen d'itérations et l'espace mémoire maximal requis pour trouver une solution avec la stratégie meilleur d'abord est exponentiel avec un exposant sous-linéaire.
L'influence de la distribution des solutions ainsi que celle de la pertinence de la fonction d'évaluation ont également été introduites dans ce modèle.

Les résultats montrent alors que le nombre total de sommets espéré pour l'arborescence construite dépend de la "pertinence" de la fonction d'évaluation de la façon suivante: plus elle est proche de la valeur exacte prise par la fonction coût f sur le sous-ensemble considéré, plus ce nombre décroit. Le taux d'augmentation ou de diminution dépend beaucoup de la distribution du coût des solutions réalisables.
Si beaucoup de solutions réalisables ont un coût voisin de celui de la solution optimale, le pouvoir de la borne inférieure (quel que soit sa pertinence) est faible et le nombre de sommets explorés devient important.
Si ce n'est pas le cas, la fonction d'évaluation apporte une information très intéressante, beaucoup de sommets avec une évaluation de coût élevé vont être éliminés, et la taille de l'arborescence sera petite.
De plus, si l'évaluation n'est pas pertinente, les stratégies meilleur et profondeur d'abord se conduisent également mal : le nombre de sommets développés est élevé.
Lorsqu'elle est très bonne, les deux stratégies se comportent bien: peu de sommets dans les arborescences. Un léger avantage, en nombre de sommets est néanmoins observé pour la stratégie meilleur d'abord.

CHAPITRE IV

IMPLEMENTATION DISTRIBUEE DES ALGORITHMES B&B

IV.1 INTRODUCTION

L'expérience montre en général que même, l'amélioration de la fonction d'évaluation, le choix d'une bonne stratégie de sélection et l'utilisation d'heuristique ne nous permettent pas de résoudre des problèmes relativement grands, faute d'espace ou de temps, qui croissent de façon exponentielle avec la taille du problème. La parallélisation constitue alors un moyen permettant la résolution de problèmes de plus grande taille.
Plusieurs approches de parallélisation peuvent être considérées, parmi elles :

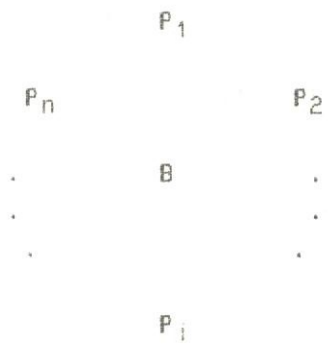
- l'évaluation parallèle.
- exploration en parallèle.
- ... .

Pour ce qui nous concerne, nous nous sommes intéressés à la parallélisation de l'exploration.
L'exploration en parallèle de plusieurs sommets de l'arborescence (au lieu d'un seul à la fois) peut permettre d'élaguer très tôt des branches de l'arborescence et conduire à des gains appréciables en temps et en espace, et permettant ainsi de traiter des problèmes de taille plus grande que ceux résolus en séquentiel.

Nous présentons dans ce chapitre deux méthodes générales pour construire des algorithmes B&B distribués, l'un basé sur la structure maitre-esclave et l'autre sur la structure asynchrone, pouvant être implémentées sur un système que nous supposons tout à fait fiable, c'est à dire qu'il n'y a ni pannes de sites, ni perte, ni altération des messages. Cependant, les retards et les déséquencements de messages sont possibles.

IV.2 B&B DISTRIBUE MAITRE-ESCLAVE [MOH 82]

On délègue un processus maitre qui se charge d'affecter les sommets de l'arborescence aux processus esclaves. Ces derniers effectuent une séparation de ce sommet et évaluent les successeurs. Le fonctionnement d'un tel algorithme peut être schématisé comme suit :



Le processus B se charge de la gestion des bornes et de la liste des sommets actifs, les autres processus (tous identiques) sont "servis" par B, il leur affecte les sous-problèmes à explorer et gère en retour les éventuelles nouvelles bornes ainsi que les éventuels sous-ensembles à séparer, le processus B représente alors un gestionnaire de mémoire.
Ce type d'algorithme est plutôt dédié à un réseau où les communications sont très rapides et où le nombre de processeurs physiques est peu élevé, le problème essentiel va être la gestion des communications afin d'éviter les goulets d'étranglement dans les relations entre le processus B et les autres processus. Une telle structure est donc très peu adaptable à un environnement distribué, plus particulierement là où les differents noeuds sont geographiquement éloignés.
Cette méthode a comme avantage de mieux tenir à jour la valeur de la meilleure solution réalisable, permettant ainsi la réduction des énumérations inutiles. Toutefois, la nécessité d'un nombre important de points de synchronisation, explique ses faibles performances. [MOH 82]

La terminaison du calcul dans cette algorithme est détectée par le processus B lorsque la file des sommets est vide, et que tous les processus P_i sont inactifs.

IV.3 B&B DISTRIBUE ASYNCHRONE [ROU 87]

Le temps de communication entre deux processus est en général beaucoup plus long que le temps nécessaire à une étape élémentaire de calcul d'un processus, ce qui nous fait penser qu'en réduisant les communications entre les processus, on réduira le temps global de calcul. Ceci a alors mené à la définition d'une autre façon de répartir le travail, où tous les processus jouent le même rôle et développent chacun une partie de l'arborescence, méthode que nous avons implémenté.

IV.3.2 CAS D'UN RESEAU COMPLET

IV.3.2.a DESCRIPTION DE L'ALGORITHME

Chaque processus se comporte de la même manière que l'algorithme B&B séquentiel, en développant avec le plus d'autonomie possible une partie de l'arborescence. Il explore les sommets de sa sous-arborescence suivant la stratégie choisie, l'information échangée entre processus est, entre autres, la valeur de la solution réalisable, chaque fois qu'elle est améliorée sur l'un des processus.
Parmi les problèmes qui se posent dans ce type de parallélisation se trouvent :

. La redistribution du travail, lorsqu'un processus termine d'explorer sa sous-arborescence.
. La détection de la fin de l'algorithme.
. ...

On inclue alors à chaque processus, la procédure de communication suivante :

- Si une meilleure solution vient d'être trouvée alors la communiquer aux autres processus.

- Si on reçoit une meilleure solution réalisable alors il faut mettre à jour la solution locale et la file des sommets.

- S'il n'y a plus de sommets à explorer dans la file local alors envoyer une demande de sous-problèmes aux autres processus.

- Si on reçoit une demande de sous-problèmes, alors si la file n'est pas vide envoyer un sous-problème au processus demandeur, sinon envoyer une réponse négative.

Ces deux derniers points lèvent le problème de redistribution du travail lorsqu'un processus termine d'explorer sa sous-arborescence. En effet, celui-ci envoie une demande de sous-problèmes aux autres processus, qui à leur tour lui en envoie s'ils en disposent.

IV.3.2.b OPTIMISATION DU NOMBRE DE MESSAGES.

1. Au début, seul l'un des processus dispose du problème "initial" pour le traiter. Les files des autres processus étant vides, ils font une demande de sous-problèmes, mais peuvent attendre longtemps avant que leur demande ne soit satisfaite. Cette attente est plus au moins importante selon la topologie du reseau.

2. Quand un processus n'a plus de sous-problèmes à traiter (sa file est vide), il fait une demande de sous-problèmes à ses voisins, les cas suivants peuvent alors souvent se présenter :

   2.a. le sous-problème reçu sera rapidement traité et le processus se voit de nouveau obligé d'en demander un autre.

   2.b. le processus qui a envoyé un sous-problème en avait peu dans sa file, il se retrouve alors lui même obligé d'en demander.

Ces problèmes font croitre le nombre de messages. Dans le but de réduire ce nombre et d'établir un équilibre des charges entre les processus, nous introduisons les solutions suivantes :

1. Au début, tous les processus auront le problème initial, qu'ils décomposeront normalement (comme cela est fait en séquentiel) jusqu'à ce que chacun ait au moins N sous-problèmes (N étant le nombre de processus) dans sa file. Le processus i prendra alors le $i^{ième}$ sous-problème et ignorera le reste, le N^{ieme} processus, quant à lui prendra le $N^{ième}$ sous-problème plus le reste des sous-problèmes qui ne sont pas pris

par les autres processus.

2.a A la réception d'une demande de sous-problème, le processus envoie au lieu d'un seul, plusieurs sous-problèmes, suivant le nombre qu'il a dans sa file et ceci en utilisant des heuristiques.

2.b Un processus recevant une demande, n'envoie de sous-problèmes que s'il dispose de plus d'un certain nombre, déterminé également par des heuristiques.

IV.3.2.c SPECIFICATION DE L'ALGORITHME

Chaque processus exécute l'algorithme principal suivant :

```
debut
évaluer(S)   (*  S la racine, représente le problème initial  *)
si S est une solution réalisable
alors fin
sinon
      insérer S dans la file F
      Meilleure solution courante = + ∞    (* peut aussi être
                                  obtenue par des heuristiques *)
      tant que F non vide et |F| < nombre de processus
      faire
            choisir S' de F
            F = F \ { S'}
            séparer S' en S'1 ... S'K
            pour chaque S'i
            faire évaluer S'i
             si v(S'i) < Meilleure solution courante
              alors
                  si S'i est une solution courante
                    alors Meilleure solution courante = v(S'i)
                    sinon insérer (S'i dans F )
                  fsi
              sinon S'i sommet éliminé
             fsi
            fait
      fait
fsi
```

```
si |F| ≥ nombre de processus
alors si p < N              (* p étant le numéro du processus,
                               N le nombre total de processus *)
      alors garder dans F uniquement le sous-problème p
      sinon garder dans F les sous-problèmes N, N+1, ...
      fsi
fsi

recevoir (les messages)     (* sans attente, à la réception d'un
                               message, la procédure de traitement
                               des messages est déclenchée   *)
tant que la file F non vide
faire choisir S' de F
      séparer S' en S'1 ... S'K
       pour chaque S'i
      faire évaluer (S'i)
            si v(S'i) < Meilleure solution réalisable
            alors
                si S'i est une solution réalisable
                  alors Meilleure solution réalisable= v(S'i)
                        diffuser (v(S'i)) aux autres processus
                  sinon S'i sommet éliminé
                fsi
            fsi
      fait

      si il existe un processus Pj inactif et |f| > nombre fixé
      alors envoyer un sous-problème au processus Pj
            diffuser (Pj réactivé)
      fsi

      si la file F vide et il existe des processus actifs
      alors diffuser (demande de sous-problèmes)
            attente jusqu'à
            réception d'un sous-problème ou
            nombre de réponses négatives = N-1 ou
            nombre de processus inactifs = N-1
      fsi

      si la file F vide
      alors diffuser (processus Pi inactif)
      fsi
fait
fsi
fin.
```

Remarques :

recevoir est une primitive de réception de messages sans attente (non bloquante). A la réception d'un message, la procédure de traitement de messages est invoquée.

Six types de messages sont possibles, à la réception d'un message de type donné, on effectue le traitement associé comme le montre la procédure suivante :

Procédure de traitement de messages

si le message reçu est :

. une **solution réalisable** alors :
    amélioration éventuelle de la solution réalisable locale,
    si la solution reçue est meilleure que la meilleure solution courante
    alors meilleure solution courante = solution reçue
        mise à jour éventuelle de la file F.
    fsi

. une **demande de sous-problème** :
    si |F| > Nombre fixé
    alors envoi d'un certain nombre de sous-problèmes au processus demandeur
    sinon envoi d'une réponse négative
    fsi

. un **sous-problème** :
    insérer sous-problème dans la file F

(* traitement de messages de contrôle, permettant essentiellement la détection de la terminaison du calcul *)

. une **réponse négative** :
    le nombre de réponses négatives est incrémenté

. **désactivation d'un processus P_j** :
    si etat(P_j) = 1
    alors le nombre de processus inactifs est incrémenté fsi
    etat(P_j) = etat(P_j) - 1

réactivation d'un processus P_j :

si etat(P_j) = 0

alors le nombre de processus inactifs est décrémenté fsi

etat(P_j) = etat(P_j) - 1

fin Procédure

Chaque processus P_i peut donc être dans l'un des états suivants: actif, en attente, inactif, terminé. La transition d'un état vers un autre est provoqué par un évènement donné, comme le montre le schéma suivant :

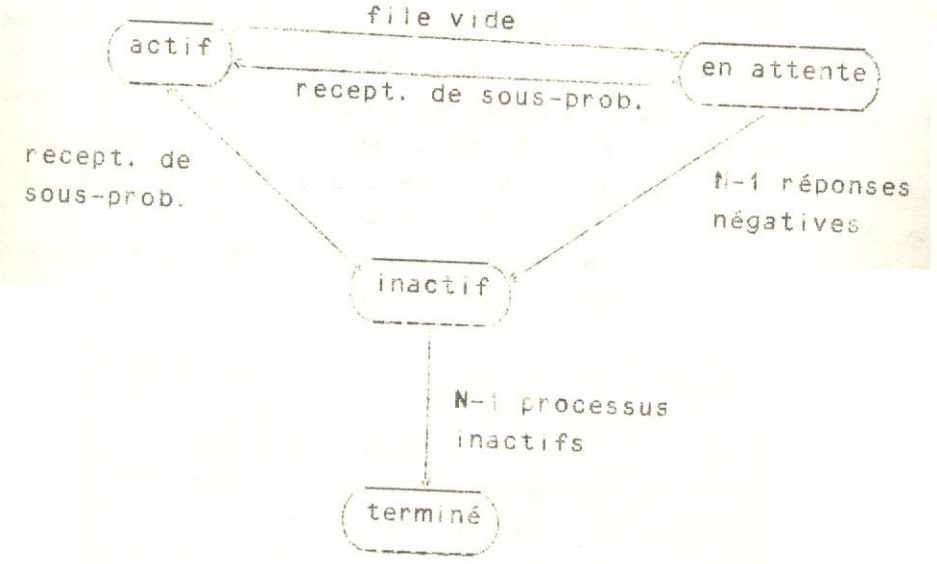


fig. Automate représentant les états possibles d'un processus.

Remarque

Le changement d'état de chaque processus est connu de tous les autres par communication de messages, explicites ou implicite.

IV.3.2.d TERMINAISON

La terminaison est une propriété basée sur l'état global et instantané du calcul distribué, or chaque processus ne peut connaitre à un instant donné que son état local et l'état

antérieur des autres processus, et le fait de terminer son calcul ne signifie pas nécessairement que le calcul global est terminé. en effet, un processus qui n'a plus de sous-problèmes dans sa file ne peut être supposé terminé, mais il fait une demande aux autres processus et se met en état "d'attente". La réception d'un sous-problème de l'un des processus actifs permet alors de le relancer.
Nous avons alors mis en oeuvre un contrôle de terminaison basé sur la connaissance, par chaque processus, de l'état (antérieur) de tous les autres, et sur les réponses aux messages de demande de sous-problèmes. Ainsi lorsqu'un processus fait une demande de sous-problèmes, il peut en recevoir des autres processus actifs, ce qui permet de le relancer, mais s'il n'en reçoit pas (s'il reçoit des réponses négatives de tous les processus actifs) il se met à l'état "inactif". Pendant ce temps, il peut aussi recevoir des sous-problèmes, il se remet alors à l'état "actif", mais si par contre, tous les autres processus sont ou se mettent à l'état "inactif", il se met alors à l'état "terminé". Cet état est atteignable par tous les processus impliqués dans ce calcul, ce qui assure la terminaison de tout l'algorithme.

A chaque processus P_j est associé un compteur **etat**(P_j) permettant de savoir son état, le processus P_j est supposé être actif si la valeur de **etat**(P_j) est à 1, inactif si **etat**(P_j) est à 0. Quand **etat**(P_j) est supérieur à 1 où bien inférieur à 0, on a alors un déséquencement de messages. Dans ce cas, on accorde un certain crédit pour la reception des messages en retard. Ce qui permet d'eviter la détection de fausses terminaison.

IV.3.3 CAS D'UN RESEAU EN ANNEAU

Le principe de l'algorithme est le même pour n'importe quelle topologie, seul le protocole d'échange de messages diffère. Dans le cas d'un réseau en anneau, les processus ne communiquent qu'avec leur successeur et leur prédécesseur. Ainsi, lorsqu'une meilleure solution réalisable est trouvée, elle devra circuler sur tout l'anneau visitant alors tous les processus leur permettant de mettre éventuellement à jour leur solution locale, et leur file des sous-problèmes. Les demandes de sous-problèmes se font au successeur, si celui-ci n'en dispose pas, il envoie un message, qui devra alors visiter tous les autres processus pour les informer que son prédécesseur est "inactif".

De même que dans le cas d'un réseau complet, un processus "inactif" peut être réactivé, mais uniquement par son successeur, qui informe alors les autres processus de cette réactivation.

IV.3.3.a LA TERMINAISON

Le principe de la terminaison dans le cas d'un anneau est semblable à celui précédemment décrit c'est à dire que chaque processus attend jusqu'à ce que tous les autres soit à l'état "inactif" pour se mettre à l'état "terminé".

IV.3.4 DISCUSSION

Les messages arrivent de façon asynchrone (à des instants aléatoires), la procédure de traitement de messages est alors invoquée ce qui peut provoquer, dans certains cas des situations indésirables sur le déroulement de l'execution.

Exemple

Si un processus trouve une nouvelle solution réalisable, mais avant d'effectuer la mise à jour, il en reçoit une meilleure d'un autre processus, la procédure de traitement de messages est alors invoquée, celle-ci mettra à jour la meilleure solution locale. Au retour, le processus affectera à son tour la valeur qu'il avait trouvée à la meilleure solution et perdra la véritable meilleure solution .

```
.
si Si est une sol-real
alors
 si v(Si) < meilleure-sol
 alors
 meilleur solution = v(Si)
 fsi
```

```
réception d'une sol. reali.
branchement vers la proc.
de traitt de messages
.
.
meilleure-sol=v(Si)
.
retour
```

La solution à ce problème est de faire de certaines parties du programme des sections critiques où certains messages ne seront consommés qu'à la fin de celles-ci.

CHAPITRE V

PERFORMANCES D'UN ALGORITHME B&B PARALLELE

V.1 INTRODUCTION

Nous nous intéresserons dans ce chapitre à une nouvelle notion, propre aux algorithmes parallèles, qu'est l'**accélération** ou encore le **speed-up**. Elle représente le gain que l'on obtient en utilisant plusieurs processus parallèles.
Cette mesure n'est pas toujours comme nous l'espérons. On peut penser que l'augmentation du nombre de processeurs va accélérer le temps de résolution, et que cette accélération ne dépasserait pas le nombre de processeurs, mais il n'en est pas toujours ainsi. On peut avoir aussi bien une accélération sur-linéaire (anomalie favorable), comme on peut avoir une décélération (anomalie défavorable).
C'est donc dans ce contexte, que nous donnerons dans ce qui suit les conditions dans lesquelles nous pouvons avoir ces anomalies.

Définitions

L'accélération ou speed-up noté Acc_K est définie comme étant le rapport entre le temps d'exécution T_1 pour une instance d'un problème sur un processeur et le temps d'exécution T_K de la même instance sur K processeurs :

$$Acc_K = \frac{T_1}{T_K}$$

Lorsque Acc_K est proportionnelle à K, on dira que l'accélération est linéaire.

Une **anomalie** est obtenue, si pour un algorithme B&B parallèle et une instance d'un problème, l'accélération Acc_K est: inférieure à 1, très inférieure ou supérieure à K

$Acc_K < 1$ anomalie préjudiciable: on perd du temps en utilisant une machine multiprocesseurs.

$1 < Acc_K << K$ accélération sous-linéaire

$Acc_K > K$ accélération sur-linéaire

Lorsque l'on utilise un nombre de processeurs $K_2 > K_1$ les cas suivants sont des **anomalies de croissance** :

préjudice de croissance si $\dfrac{Acc_{K1}}{Acc_{K2}} < 1$

décélération en croissance si $1 < \dfrac{Acc_{K1}}{Acc_{K2}} << \dfrac{K_1}{K_2}$

accélération en croissance $\dfrac{Acc_{K2}}{Acc_{K1}} > \dfrac{K_1}{K_2}$

Remarque 1 : Il est évident que $\dfrac{Acc_{K2}}{Acc_{K1}} = \dfrac{T_{K1}}{T_{K2}}$

Remarque 2 : Nous pouvons considérer de la même manière le gain en place mémoire, qui peut être défini comme suit :

$$G_K = \frac{N_1}{N_K}$$

où N_1 est le nombre maximal de sommets avec un processeur
N_K est le nombre maximal de sommets avec K processeurs

Les mêmes définitions concernant les anomalies du gain en espace peuvent être aussi établies.

V.2 CALCUL DE L'ACCELERATION

Quelques modèles théoriques (modèles probabilistes) ont été développés dans le but de déterminer des mesures de performances dans le calcul parallèle. Parmi ces mesures, il y a l'accélération en temps, qui représente le gain dû à l'augmentation du nombre de processeurs. La plupart de ces modèles considèrent les systèmes parallèles centralisés (les processus accèdent à une mémoire commune).
Dans [JAN 88], un modèle pour l'algorithme B&B parallèle avec la stratégie heuristique (aléatoire), a été proposé, les résultats de simulation étaient à environ 20% près des valeurs prédites par ce modèle.
Dans [ROU 87], et toujours, pour les mêmes systèmes, on trouve une preuve que l'accélération, pour la stratégie meilleur d'abord est d'autant plus proche de K (K étant le nombre de processeurs utilisés) que la taille du problème augmente.
Mais tous ces modèles sont faits pour des systèmes centralisés et supposent aussi des hypothèses simplificatrices (sur les délais de transfert de l'information, sur la fonction d'évaluation faible, discriminante, ...), car il est très difficile, voire impossible de considérer dans un modèle théorique tous les paramètres qui entrent en jeu en praticue, dans un algorithme parallèle.
pour pallier à ces insuffisances, certains auteurs ont adopté les modèles expérimentaux. Parmi eux J.Mohan [MOH 82] a mené deux expériences sur le problème du voyageur de commerce, réalisées sur une machine Cm* de Carnegie Mellon. La première (TSP1) a été l'implémentation selon la structure maitre-esclave pour laquelle l'accélération décroit dès que l'on dépasse 4 processeurs, la seconde (TSP2) a été une implémentation asynchrone où l'accélération croit en fonction du nombre de processeurs (les expériences ont été réalisées sur un nombre de processeurs compris entre 2 et 16).

V.3 EXISTENCE D'ANOMALIES DANS LES ALGORITHMES B&B PARALLELES

L'existence et les conditions d'existence de ce phénomène dans les algorithmes parallèles ont été peu étudiées, néanmoins, quelques auteurs, tel Wilke en a donné une illustration. Les résultats existants sont relatifs aux algorithmes parallèles synchrones, simplification souvent utilisée lorsque l'on veut étudier un algorithme parallèle [LAI 84], [LAI 85], [LI 84].

Pour les algorithmes B&B parallèles, ce phénomène peut aussi exister et cela dépend en partie des propriétés de la fonction d'évaluation (discriminante, ...) et de la stratégie utilisée. Ainsi, si en séquentiel l'algorithme B&B est optimal (construit l'arborescence minimale), l'accélération en temps obtenue avec K processeurs sera au plus égale à K, car l'arborescence explorée en parallèle contient toujours l'arborescence minimale quelle que soit la stratégie utilisée. Ceci veut dire que l'on ne peut pas avoir des accélérations surlinéaires pour des algorithmes B&B optimaux en séquentiel.

THEOREME [ROU 87]

Si un algorithme B&B utilise la stratégie meilleur d'abord et une fonction d'évaluation discriminante ou faible, l'accélération Acc_K obtenue avec K processeurs (K>1) est au plus égale à K ($Acc_K <= K$).

COROLLAIRE [ROU 87]

Une condition suffisante pour qu'une stratégie meilleure d'abord ne provoque pas des anomalies favorables, est que la fonction d'évaluation soit discriminante (ou faible).
Une condition nécessaire pour qu'elle provoque une accélération sur-linéaire est que des sommets non-optimaux de l'arborescence aient des évaluations identiques et égales à la valeur de la solution optimale.

V.4 EXISTENCE D'ANOMALIES DE CROISSANCE

THEOREME [LAI 86]

Soient d et m respectivement la profondeur et le nombre de noeuds de l'arborescence, soit $K_2 > K_1$, si la stratégie adoptée est meilleur d'abord, la fonction d'évaluation est faible et que :

$$\frac{m - d}{K_2} + d - 1 < \frac{m}{K_1}$$

alors il n'y aura pas d'anomalies préjudiciables si l'on utilise K_2 au lieu de K_1 processeurs.

COROLLAIRE [LAI 86]

Pour les mêmes hypothèses, si $K_1 < (m + d)/2(d-1)$ alors il ne se produit pas d'anomalies préjudiciables si l'on double le nombre de processeurs.

V.5 DISCUSSION

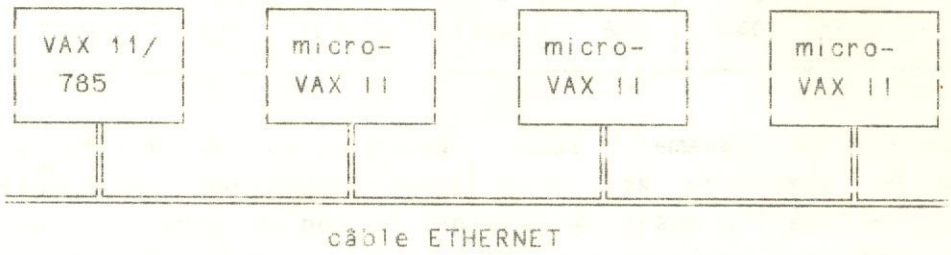
L'étude existante des anomalies dans les algorithmes B&B parallèles dans [LI 86], [LAI 86] a été faite dans le cas de machines multiprocesseurs utilisant une mémoire commune, mais elle ne tient pas compte de beaucoup de problèmes se posant dans de tels systèmes (gestion, contrôle de ressources, ...) et influant énormément sur les paramètres de performances.
Les algorithmes B&B distribués restent encore peu étudiés; certains auteurs ont fait des études expérimentales des performances des algorithmes B&B parallèles.
O.Vornberger [VOR 86] a fait une étude expérimentale, en implémentant le problème du voyageur de commerce, sur un réseau d'ordinateurs personnels ayant une topologie anneau.
Aussi, Kumar et Rao [RAO 88] ont fait une implémentation et une analyse de la recherche en profondeur en parallèle sur différentes architectures (anneau, cube, ...).

Dans le prochain chapitre, seront présentés les résultats obtenus lors de l'implémentation d'un algorithme B&B (pour la résolution du problème du voyageur de commerce) dans un réseau d'ordinateurs avec les topologies anneau et complet.

CHAPITRE VI

TESTS ET RESULTATS

Les tests ont été réalisés sur un réseau de quatre machines, un VAX 11/785 et trois micro-VAX II, connectées entre-elles par un câble de diffusion ETHERNET :



Nous avons opté, et ce dans le but d'obtenir un parallélisme réel, d'associer un seul processus à chaque site (processeur physique).
Le contexte d'application choisi est le problème classique du voyageur de commerce symétrique, pour des tailles variant de 10 à 40 villes. Les différentes instances traitées sont tirés de façon pseudo-aléatoire, la stratégie de parcours adoptée est "meilleur d'abord".

A travers ces tests, nous essayons d'analyser:

- Dans le cas séquentiel, l'influence de la fonction d'évaluation sur, le temps de recherche, le nombre de noeuds explorés et l'espace mémoire utilisé.

- Dans le cas distribué, l'influence, de la fonction d'évaluation, de la topologie et la taille du problème sur l'accélération.

VI.1 INFLUENCE DE LA FONCTION D'EVALUATION

Nous avons effectué des essais en séquentiel en utilisant deux fonctions d'évaluation basées sur le problème d'affectation. Toutefois, dans l'une d'elles, nous effectuons en plus des compressions et des contractions du grapne partiel obtenu aprés la résolution du problème d'affectation à l'aide de l'algorithme du Hongrois améliorant ainsi l'autre fonction (voir annexe).

	affect.			affect.+ comp.+ contr		
taille	NE	TF	T_{cpu}	NE	TF	T_{cpu}
10	1	4	0.07	1	4	0.09
20	26	54	3.96	13	54	4.07
30	23	174	55.50	17	157	46.72
40	2623	12796	11320	650	2400	1899.2

A partir de ce tableau, nous remarquons la réduction considérable, particulièrement pour les problèmes de grande taille, du nombre de noeuds explorés NE grâce à l'amélioration de la fonction d'évaluation, ce qui a pour effet la réduction du temps CPU (T_{cpu} en seconde) et en espace (TF : taille maximale de la file des sous-problèmes) en consentant un effort supplémentaire dans l'évaluation. Néanmoins, le caractère exponentiel du temps et de l'espace mémoire requis pour la recherche de la solution optimale, en utilisant une fonction d'évaluation améliorée, reste toujours.

VI.2 INFLUENCE DE LA FONCTION DEVALUATION SUR L'ACCELERATION

Dans le tableau ci-dessus sera illustrée l'influence de la fonction d'évaluation sur l'accélération dans le cas d'un réseau complet:

taille	acc_1	acc_2
10	0.087	0.126
20	0.935	0.878
30	2.004	1.893
40	2.438	2.238

acc_1 (resp. acc_2) correspond à l'accélération obtenue en utilisant l'affectation comme fonction d'évaluation (resp. à l'accélération obtenue en utilisant en plus de l'affectation la compression et la contraction) (voir annexe).

Nous remarquons que l'amélioration de la fonction d'évaluation diminue l'accélération, bien que le temps de calcul ait baissé considérablement en parallèle. Cette diminution est dûe au fait que le temps de recherche effectif a diminué pour donner la domination aux traitements supplémentaires propres au calcul distribué.

VI.3 INFLUENCE DE LA TOPOLOGIE SUR L'ACCELERATION

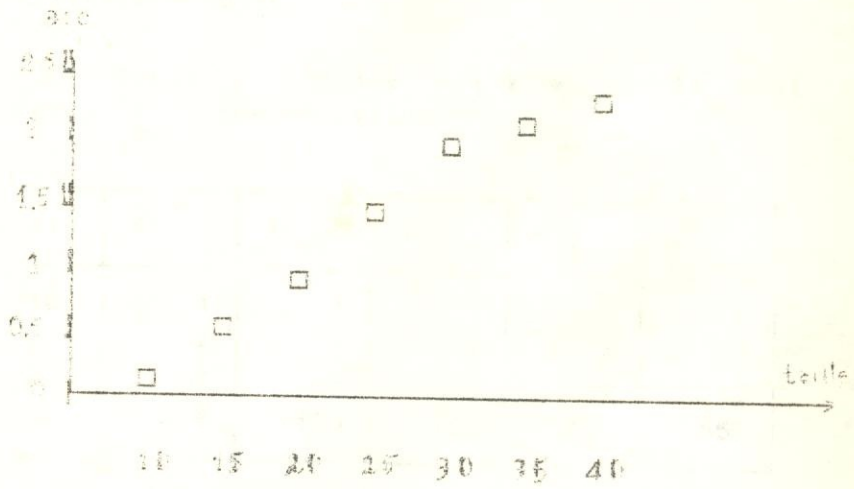
Nous avons aussi effectué des tests sur deux topologies différentes, le réseau complet et le réseau en anneau bidirectionnel, et ce dans le but de voir l'influence de l'itinéraire et le protocole d'échange de messages sur le temps d'exécution.
Les résultats obtenus à partir des tests sont donnés par le tableau suivant :

taille	acc_a	acc_c
10	0.063	0.126
20	0.733	0.878
30	1.752	1.893
40	2.138	2.238

Nous remarquons que les résultats obtenus dans le cas d'un réseau complet sont meilleurs que ceux d'un réseau en anneau. Nous pensons que cela est dû au protocole d'échanges d'informations, plus restreint dans le cas d'un anneau. En effet, dans le réseau complet, la diffusion de messages permet une meilleure communication, alors que dans le réseau en anneau, les messages devant visiter tout le réseau, parcourent un noeud à la fois, ce qui influe sur le temps de calcul.

VI.4 L'ACCELERATION EN FONCTION DE LA TAILLE DU PROBLEME

La courbe ci-dessous montre l'accroissement de l'accélération en fonction de la taille du problème traité:



Nous remarquons que l'accélération pour les problèmes de petite taille (inférieure à environ 20 villes) est inférieure à 1, ce qui veut dire que pour ces tailles, même l'utilisation de plusieurs processus parallèles ne diminue, voire augmente le temps de calcul global. Cependant, cette accélération augmente en fonction de la taille, sans toutefois atteindre un certain seuil, égal théoriquement au nombre de processus parallèles, pour des problèmes de taille 40 par exemple, l'accélération atteint 2.6, cette différence est dûe aux traitements supplémentaires introduits dans l'implémentation distribuée, ainsi qu'au fait que dans la méthode choisie pour le calcul distribué, des noeuds de l'arborescence qui ne sont pas explorés en séquentiel, le sont en distribué, comme le montre le tableau ci-dessous:

taille	NE_{PAR}	NE_{SEQ}
10	16	4
20	78	54
30	181	157
40	3192	2400

VI.5 L'ESPACE MEMOIRE

Nous considérons la taille maximale de la file des noeuds critiques pour chaque processus.

taille	TF_0	TF_1	TF_2	TF_3	TF_{seq}
10	1	1	1	1	1
20	4	6	4	4	13
30	5	6	6	4	17
40	218	183	142	221	650

TF_i : taille maximale de la file des noeuds critiques du processus i

TF_{seq}: taille maximale de la file des noeuds critiques dans le cas séquentiel

D'après ce tableau, nous constatons une répartition de l'espace utilisé dans le cas du calcul distribué, néanmoins l'espace global utilisé est plus grand que celui utilisé en séquentiel, cette augmentation résulte aussi des explorations inutiles.

VI.6 CONCLUSION

Les tests réalisés sur un réseau de 4 noeuds (4 processus parallèles) ont montré que l'accélération pour les problèmes de petite taille, inférieure à environ 20 villes, est inférieure à 1. Cependant, celle-ci augmente en fonction de la taille du problème, sans pour autant dépasser un certain seuil, l'accélération obtenue pour des problèmes de taille 40, par exemple, était de 2.6. On s'attend à ce que ce seuil soit proche de 4 (vu que l'on a utilisé quatre processus parallèles), mais il n'en est pas ainsi, car des traitements supplémentaires, qui ne

sont pas considérés en séquentiel, sont introduits dans l'implémentation distribuée. Ces traitements peuvent être résumés dans les points suivants :

- Etablissement des liaisons entre les différents processus.
- Traitements de contrôle (détection de la terminaison, traitement des messages, ...).
- Exploration de noeuds inutiles, non explorés en séquentiel.
- Attentes inactives (quand la file d'un processus est vide, il attend, tout en étant inactif, la réception d'autres sous-problèmes).

En plus des traitements sus-cités, il en existe d'autres (les initialisations, par exemple), qui ne pouvaient être parallélisés, ils sont alors effectués par tous les processus. Toutefois, nous obtenons un gain qui pourrait être satisfaisant en temps et en espace mémoire (si l'on considère l'espace mémoire utilisé par chaque processus).
Malheureusement, nous ne pouvions faire de tests pour des problèmes de plus grande taille, les temps de calcul devenant prohibitifs.

CONCLUSION

La méthode B&B est l'outil permettant la recherche de la solution optimale pour beaucoup de problèmes réputés d'être "difficiles" (NP-difficiles), mais le temps de résolution devient vite prohibitif en augmentant la taille du problème. Celui-ci augmente de façon exponentielle avec la taille. D'autre part, l'espace mémoire nécessaire au développement de l'arborescence de recherche peut aussi augmenter de façon combinatoire, selon la stratégie choisie pour effectuer la recherche. Nous avons vu que si nous choisissons la stratégie profondeur d'abord, l'espace mémoire nécessaire est une fonction polynômiale de la taille du problème, mais le temps de recherche sera encore plus grand qu'avec d'autres stratégies.
Une façon de réduire le temps de recherche est l'amélioration de la fonction d'évaluation. En effet, si on investit un temps plus important dans la recherche d'une bonne évaluation, cela se traduit par une réduction considérable du temps global de la recherche de la solution optimale. Cependant, les problèmes de grande taille peuvent rester non résolus, faute de temps ou d'espace mémoire, vu le caractère exponentiel de la méthode.
Une façon naturelle de réduire le temps de recherche est alors la parallélisation.
Pour réaliser la recherche de la solution optimale en parallèle, il est plus approprie de choisir une machine multiprocesseurs ou un réseau distribué, c'est à dire des machines à "parallélisme global".
Nous avons donné deux méthodes de parallélisation indépendantes du contexte d'une application particulière et destinées à être implémentées sur des architectures de machines présentant des formes de parallélisme asynchrones, et qui peuvent servir de schémas de base aux algorithmes conçus pour des machines multiprocesseurs à mémoire partagée.
Ces méthodes, correspondant chacune à une structure de parallélisation, ont été proposées selon :

- la structure maitre-esclaves.
- la structure asynchrone.

Nous avons alors choisi cette dernière méthode pour son adéquation à l'implémentation distribuée. Elle offre une certaine

autonomie des processus réduisant ainsi les interactions (communications) entre eux. En effet, chaque processus explore une partie de l'arborescence globale de façon autonome et ne communique avec les autres processus que s'il a trouvé une solution réalisable ou si sa file est vide, il envoie alors un message de demande de sous-problèmes, et aussi quelque messages de contrôle, permettant la détection de la terminaison. Nous avons proposé en outre des méthodes d'implémentation sur deux topologies différentes, l'anneau et le réseau complet.
Nous avons donné un aperçu sur les modèles théoriques étudiant les performances des algorithmes B&B parallèles et présentant les conditions nécessaires d'existence d'anomalies.
Pour ce qui nous concerne, nous avons opté pour l'expérimentation sur un réseau de quatre noeuds. Les résultats obtenus nous ont permis de dégager les conclusions suivantes :

. Il y a un meilleur partage de données dans le cas du réseau complet.

. L'amélioration de la fonction d'évaluation a entrainé une diminution de l'accélération, néanmoins, elle a permis aussi la diminution considérable du temps de calcul en parallèle.

. Des anomalies favorables n'ont pas été obtenues car nous avons utilisé la stratégie meilleur d'abord, par contre des anomalies défavorables ont été obtenues pour des problèmes de petite taille.

. L'accélération croît en fonction de la taille du problème, mais possède un seuil qu'elle ne peut pas dépasser. Ce seuil est inférieur au nombre de processus.

. L'espace mémoire utilisé par chaque processus est réduit grâce à la distribution des sous problèmes entre ces processus (chacun explore une partie de l'arborescence, ces parties sont disjointes).

La nouveauté et la complexité du domaine est un facteur ouvrant de larges perspectives aux investigations, entre autres, nous proposons :

. L'étude des phénomènes de comportement de ces algorithmes, qui peut aussi être poursuivie pour dégager les critères d'exploration adaptés aux parallélisme.

. Les architectures des machines et les topologies des réseaux doivent aussi être mises en jeu pour avoir l'influence sur l'accélération et dégager l'architecture ou la topologie la plus adéquate pour ce type d'algorithmes.

. Enfin, ce travail pourrait être intégré dans un système de résolution (résolveur) de problèmes combinatoires, afin de réduire le temps de recherche, le problème majeur de ces systèmes étant l'explosion temporelle et spaciale.

ANNEXE

LE PROBLEME DU VOYAGEUR DE COMMERCE

Un représentant de commerce doit rendre visite à n clients $x_1, x_2, \ldots x_n$ en partant d'une ville x_0 et revenir à son point de départ. Il connait les distances $d_{i,j}$ séparant deux clients quelconques x_i et x_j (si $d_{i,j} = d_{j,i}$ on parlera de problème du voyageur de commerce symétrique).
Dans quel ordre doit-il rendre visite à ses clients pour que la distance totale parcourue soit minimale ?. Ce problème revient à la recherche d'un cycle hamiltonien dans un graphe complet G (d'une tournée) construit sur l'ensemble des sommets $X = \{ x_0, x_1, \ldots x_n \}$, les arêtes étant munies des longueurs $d_{i,j}$.
Lorsque le point d'arrivée x_{n+1} est différent du point de départ x_0 le problème revient à la recherche d'une chaine hamiltonienne de longueur totale minimale entre x_0 et x_{n+1}. On se ramène aisément au problème de la recherche d'un cycle hamiltonien en transformant le graphe.
Le problème du voyageur de commerce (orienté ou non orienté) se rencontre dans de très nombreuses situations et dans des domaines extrêmement variés: ramassage scolaire, ordonnancement de production, câblage de circuit, synthèse de circuits logiques séquentiels, etc.
On ne connait pas d'algorithme polynômial pour résoudre le problème du voyageur de commerce, les algorithmes les plus efficaces utilisent les méthodes de recherche arborescente B&B.
Cependant, notons qu'il existe des cas particuliers où l'on peut résoudre ce problème par un algorithme polynômial.

A.1 FORMALISATION DU PROBLEME DU VOYAGEUR DE COMMERCE SYMETRIQUE

Le problème du voyageur de commerce symétrique peut être formulé par le programme linéaire en nombres entiers suivant :

$$(1) \qquad \sum_{i<j} x_{ij} + \sum_{1<j<i} x_{ji} = 2$$

$$(2) \qquad \sum_{j=2..n} x_{1j} = 2$$

$$(3) \qquad \sum_{1\le i\le j\le n} x_{ij} = n$$

$$(4) \qquad \sum_{i\in X'} \sum_{j\in X} x_{ij} <= |X'|-1 \quad \text{pour tout } X' \quad \{ 2, \ldots n \}$$

$$\sum_{1\le i<j\le n} c_{ij} x_{ij} = Z(Min)$$

On remarque que les contraintes (3) et (4) sont celles qui définissent les 1-arbres, et les contraintes (1) et (2) assurent que celui-ci a tous ses sommets de degré 2 pour être une tournée. Tandis que le problème réduit aux contraintes (1) et (2), représente le problème de l'affectation, et les contraintes (3) et (4) assurent la connexité pour que le graphe partiel qui en résulte soit une tournée.
Ces deux propriétés nous permettent donc d'utiliser soit le 1-arbre, soit l'affectation comme fonction d'évaluation au problème du voyageur de commerce; on peut dire aussi que les problèmes de recherche du 1-arbre et d'affectation sont des problèmes relaxés du problème du voyageur de commerce.

Définition du 1-arbre

Le graphe A = (X, E) où l'ensemble des sommets est X = {1,2, ... m} est un "1-arbre" si :

- Le sous-graphe de (X, E) construit sur les sommets {2,3, ... m} est un arbre
- (X, E) contient deux arêtes incidentes au sommets 1.

Un 1-arbre de coût minimum peut être obtenu en construisant un arbre de poids minimun dans le sous graphe construit par les sommets {2,3, ... m} auquel on ajoute deux arêtes incidentes au sommet 1 de coût minimum.

A.2 METHODE DE RESOLUTION DU PROBLEME DU VOYAGEUR DE COMMERCE SYMETRIQUE UTILISANT LE 1-ARBRE

Soit S l'ensemble des graphes partiels de $G = (X,E)$ qui sont des tournées. Les sous-ensembles S' de S engendrés dans le cours de l'algorithme sont définis par deux ensembles disjoints E_0 et E_1 de E, par conséquent S' sera désigné par $S(E_0,E_1)$ avec :

$S(E_0,E_1) = \{ (X,E') / (X,E')$ est une tournée, E' inclus dans E , E_1 inclus dans E', $E_0 \cap E' = \emptyset \}$

c'est à dire que $S(E_0,E_1)$ est constitué des tournées qui ne contiennent aucune arête de E_0 et toutes les arêtes de E_1.

Notons $G_0 = (X, E_0)$; $G_1 = (X, E1)$; $G = (X, E)$
$d_{G'}(x)$ = degré du sommet x dans le graphe G'

La séparation

Si le 1-arbre G' déterminé pour effectuer l'évaluation de $S(E_0,E_1)$ est une tournée, on a alors une évaluation exacte (donc on n'a pas à séparer le sommet correspondant). Sinon, il existe un sommet x avec $d_{G'}(x) \geq 3$, donc au moins une des arêtes adjacentes à x n'appartient pas à la tournée, on peut alors procéder à une séparation de l'ensemble $S(E_0,E_1)$. Soient E_0' et E_1' les ensembles désignant respectivement E_0 et E_1, des descendants de $S(E_0,E_1)$, qui peuvent alors être construits comme suit:

On choisit 2 arêtes e' et e" de G' incidentes à x et on sépare en trois sous-ensembles :

i. $E_0' = E_0 \cup \{ e' \}$, $E_1' = E_1$

ii. $E_0' = E_0 \cup \{ e" \}$, $E_1' = E_1 \cup \{ e' \}$

iii. $E_0' = E_0 \cup \{ e <> e', e" / e$ adjacent à x $\}$,
$E_1' = E_1 \cup \{ e', e" \}$

A.4 AMELIORATION DE LA FONCTION D'EVALUATION [CHR 75]

Le graphe partiel obtenu après résolution du problème d'affectation est composé de plusieurs circuits, soit n_1 le nombre de ces circuits et $S_{1,i}$ le $i^{ème}$ circuit, ($S_{1,i}$ représentera aussi l'ensemble des sommets de ce circuit).

La **contraction** est définie comme étant le remplacement de chaque circuit par un sommet unique, ce qui formera un graphe "contracté" contenant n_1 sommets $S_{1,i}$ ($i=1,2, \ldots n_1$). La matrice des coûts C_1 est déduite de la manière suivante :

$$c(S_{1,i}, S_{1,j}) = \min_{\substack{k_i \in S_{1,i} \\ k_j \in S_{1,j}}} [\, f(k_i, k_j)]$$

où $F_1 = [f_1(k_i, k_j)]$ est la matrice qui résulte à la fin de la résolution du problème d'affectation, utilisant l'algorithme du Hongrois. La résolution du nouveau problème (du graphe contracté) peut à son tour produire des circuits, le processus itératif de résolution-contraction peut être poursuivi jusqu'à ce que l'on obtienne un graphe réduit à un seul circuit.

La **compression** est la transformation d'une matrice ne satisfaisant pas la propriété de triangularité dans l'espace métrique, ainsi pour comprimer une matrice M, il est nécessaire de remplacer chaque élément m_{ij}, tel que :

$$m_{ij} > m_{iK} + m_{Kj} \quad \text{pour un } K \text{ donné}$$

par le $\min [\, m_{iK} + m_{Kj}]$ pour tout k, et continuer le remplacement jusqu'à ce que tous les m_{ij} vérifient:

$$m_{ij} <= m_{iK} + m_{Kj} \quad \text{pour tout } k$$

Théorème [CHR 75]

La somme des valeurs des solutions au problème d'affectation obtenues durant le processus itératif "résolution-contraction-compression", jusqu'à ce que le graphe contracté ne soit réduit qu'à un seul sommet, est une évaluation valide pour le problème de voyageur de commerce.

BIBLIOGRAPHIE

[BAU 65] L. D. Baumert et S. W. Golomb,
"Backtrack programming", Jet Propulsion Laboratory, Pasadena, California (1965).

[BLA 88] P. Blanc,
"Détection de la terminaison d'un calcul distribué asynchrone avec deséquencement"
Rapport de recherche, Université P. et M. Curie, Labo MASI, Paris VI, (1988).

[BRO 81] C.A. Brown, et P.W. Purdom Jr.,
"An average time analysis of backtracking"
SIAM J.Comput., Vol 10, 583-593 (1981)

[CHR 75] N. Christifides,
"Graph theory - An algorithmic approach"
Academic Press, London, (1975).

[IBA 76] T. Ibaraki,
"Theoretical comparisons of search strategies in B&B algorithms"
Int.J.Comput.Inform.Scie. Vol 5, n·4 315-344 (1976).

[JAN 88] V. K. Janakiram, E. F. Gehringer, D. P. Agrawal et R. Mehrotre,
"a randomized parallel Branch & bound algorithm",
Internat. Journal of parallel programming, vol 17, n· 3, pp 277-300, (1988).

[KOH 74] W. Kohler, et K. Steiglitz,
"Characterization and theorical comparaison of B&B algorithms for permutation problems"
J.ACM, Vol 21, n·1, 140-156 (1974)

[LA LA 85] I. Lavallée et C. Lavault,
"Algorithmes parallèles et distribués", Rapport de Recherche n· 471, INRIA, Rocquencourt (1985).