MASTER EN SCIENCES INFORMATIQUES

Comparaison expérimentale de méthodes de restructuration de programmes

Collard, Michel

Award date:
1977

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX

NAMUR

Institut d'Informatique

Année académique 1976-1977

COMPARAISON EXPERIMENTALE DE METHODES DE RESTRUCTURATION DE PROGRAMMES

Michel COLLARD

Mémoire présenté en vue de l'obtention du grade de licencié et maître en informatique.

Je tiens à remercier toutes les personnes ayant contribué à la réalisation de ce travail .

Ma profonde reconnaissance va à Monsieur le professeur J. RAMAEKERS pour son aide, ses conseils et critiques constructives sans lesquels ce mémoire n'aurait jamais vu le jour.

Je remercie également Monsieur G. VANGHELUWE pour son assistance constante lors de la mise en oeuvre des programmes et l'instrumentation, et Monsieur TRAN-QUOC TE dont les conseils furent précieux lors du dépouillement des résultats.

Je remercie enfin madame PIJCK pour la diligence et le soin qu'elle a apportés à la frappe.

TABLE DES MATIERES

CHAPITRE 1 : INTRODUCTION

Toute instruction doit se trouver en mémoire centrale pour pouvoir s'exécuter. Dans les systèmes à mémoire virtuelle paginée, les parties de programmes se retrouvent alternativement en mémoire centrale ou sur le support de mémoire auxiliaire. Lorsque le programme à exécuter référence une zone de données ou une instruction qui ne se trouve pas en mémoire centrale, il se produit un défaut de page. La page référencée doit être amenée en mémoire centrale pour que l'exécution puisse se poursuivre, avec, le cas échéant, libération d'un cadre occupé. Le phénomène est coûteux puisqu'il correspond à un travail supplémentaire à effectuer par le système et allonge le temps d'exécution (elapsed time) des programmes des utilisateurs. (01) Dès que la dégradation devient trop importante (trashing), des mesures radicales doivent être prises : arrêt d'introduction de nouvelles tâches, ... Les systèmes courants de gestion de la mémoire sont conçus de façon à éviter ces situation, bien qu'ils ne soient pas optimaux.

Plutôt que de remettre en cause la stratégie de gestion de la mémoire d'un système, une amélioration substantielle peut être obtenue en adaptant les programmes à exécuter à la stratégie en vigueur ou tout au moins, en les organisant quelque peu. Ce travail est appelé restructur tion.

Il consiste à réarranger les blocs résultant d'une découpe logique des programmes, dans des pages, unité d'échange entre mémoire centrale et auxiliaire.

L'intérêt de la restructuration ne fait aucun doute, et a été montré dès 1967 par Comeau. (08)

CHAPITRE 2 : SITUATION ET BUT DU TRAVAIL

2.1. Cadre du travail

Nous nous plaçons dans le cadre d'un système à mémoire virtuelle paginée et multiprogrammé. Nous considérons que la pagination s'effectue à la demande, c'est-à-dire que toute page n'est amenée en mémoire principale qu'à partir du moment où elle est nécessaire pour la suite de l'exécution du programme demandeur. Aucune page n'est chargée préventivement.

Indépendamment de l'environnement, c'est-à-dire sans remettre en cause la stratégie de gestion de la mémoire ni la charge du système considéré, nous désirons restructurer les programmes des utilisateurs - et éventuellement certains utilitaires tels que compilateurs, programmes de tri, de conversion ... - de façon à améliorer leur schéma de pagination et d'utilisation de mémoire centrale.

La restructuration aura des conséquences au niveau du comportement global du système (throughput). Notre investigation se limitera cependant à déterminer son influence au niveau des programmes concernés.

Le travail à effectuer poursuit celui entrepris par G. Gaspard (voir 16) et consistera notamment en un examen d'algorithmes supplémentaires.

2.2. Le problème vu par le gestionnaire du centre

Avant d'entrer plus avant dans les techniques de restructuration proprement dites, il serait bon d'avoir une vision globale du problème.

Les programmes sont de plus en plus conçus de façon modulaire, et leur comportement interne est généralement mal connu. Si, pour une exécution donnée, la consommation totale en temps CPU est facilement identifiable, la ventilation par routine est moins évidente. La façon dont le contrôle du processeur évolue dans le temps est encore plus difficile à déterminer. Si, parallèlement, on désire connaître les zones successivement référencées, des outils d'espionnage du programme deviennent nécessaires. Le phénomène de localité est défini comme étant le fait que, durant tout intervalle d'exécution, un processus référence certains de ses constituants plus que d'autres. (24) La séquentialité des instructions, la modularité et les mécanismes de programmation tels les boucles, en sont les principales raisons. Ces éléments sont intrinsèques aux programmes.

Associés à chaque programme, nous trouvons d'autre part les facteurs suivants :

- la durée de vie, ou plus précisément, le nombre d'exécutions futures ;
- le coût de chacune, plus exactement, la consommation des ressources qui nous occupent plus particulièrement : mémoire centrale et défauts de page.

L'environnement est constitué de composantes de deux types :

- fixes : - stratégie de gestion mémoire du système ;
  - taille de la page ;
  - support auxiliaire de la mémoire ...
- variables : retenons essentiellement la charge, c'est-à-dire l'ensemble des processus concurrents

Ces considérations vont être prises en compte pour déterminer quel est, pour chaque programme, l'intérêt d'effectuer une restructuration. Cette dernière n'est pas un but en soi. En effet, les procédures utilisées pour la réaliser sont coûteuses, alors que le défaut de page n'est gênant que dans la mesure où il dégrade les performances globales. Ceci dépend essentiellement de la charge.

La démarche proposée vise en premier lieu à déterminer les périodes où une amélioration globale (c'est-à-dire pour la charge totale) est nécessaire, ainsi que le niveau de cette dernière.

Elle est fonction du dommage entraîné par le défaut de page qui représente l'influence de celui-ci sur le "throughput" et dépend des facteurs suivants :

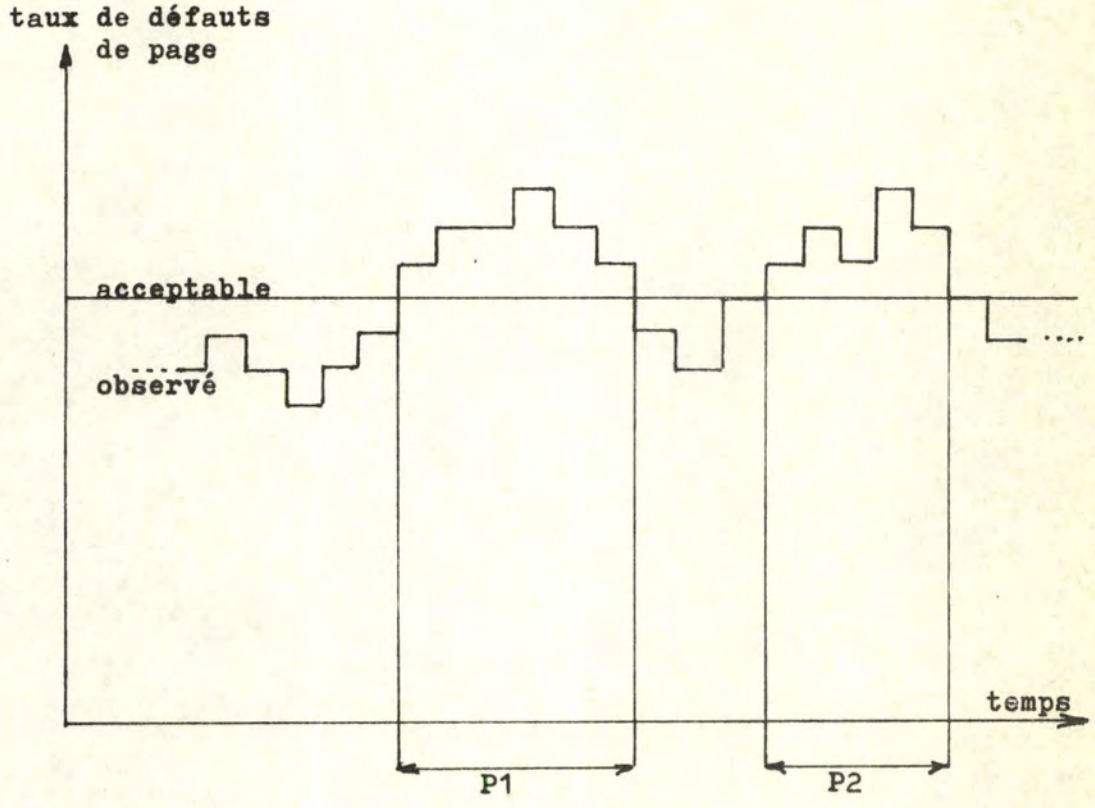
- overhead CPU entraîné : temps CPU consommé par le système pour prendre en compte et satisfaire la requête, avec gestion des tables et files d'attente concernées ;
- durée de satisfaction du défaut de page : temps "elapsed" entre la détection du défaut de page et l'instant où le programme demandeur est à nouveau prêt à poursuivre son exécution.

  En plus de la précédente, les composantes suivantes entrent en ligne de compte :

  - encombrement des canaux dû au paging ;
  - taille mémoire centrale disponible. Dans le cas où la recherche d'un ou plusieurs cadres libres n'aboutit pas, le système est amené à libérer et éventuellement à recopier, une ou plusieurs pages appartenant au processus demandeur ou à un autre ;
  - taux de défauts de page. Ce sont en effet les "pointes" correspondant à un nombre élevé de défauts de page pendant un court intervalle de temps, qui sont les plus néfastes.

Il est clair que le dommage varie dans le temps, essentiellement en rapport avec la charge. Tenir compte de tous ces éléments en tant que tels pour se définir un critère pratique d'amélioration est pratiquement irréalisable vu la difficulté de mesure et du coût engendré. Un critère acceptable est le taux de défauts de page, c'est-à-dire le nombre de défauts de page apparaissant dans un intervalle de temps à déterminer. Si ce dernier est trop grand, les effets des pointes sont minimisés. S'il est trop petit, le taux est réduit artificiellement.

Il faudra en premier lieu déterminer quel est le taux acceptable pour l'installation, l'amélioration à obtenir découlant de la différence entre taux observé et acceptable. Visualisons cela sur un schéma :



L'étape suivante vise à ventiler la dégradation entre les différents constituants de la charge. Il s'agit d'identifier la responsabilité de chacun et d'atteindre l'objectif fixé à un coût minimum. La ventilation se fera sur base du nombre de défauts de page provoqués par chacun des processus concurrents pour chacune des périodes considérées (P1 et P2 sur l'exemple précédent), l'amélioration devant alors être exprimée en nombre de défauts de page.

Outre les solutions radicales qui remettent en cause l'installation ou la charge, apparaît la possibilité de restructuration. Pour un programme donné, l'intérêt dépend :

- du nombre d'exécutions futures ;
- de la consommation en défauts de page.

Il conviendra donc de délaisser les programmes peu utilisés ou peu consommateurs et de reporter sur les autres, l'amélioration à obtenir. De même pour ceux qu'il ne sera pas possible de restructurer. La restructuration n'ayant lieu qu'une fois pour un programme donné, la procédure à choisir pour celui-ci sera celle qui assure le niveau de performance maximum exigé pour ce programme.

Le problème se résume à déterminer quels sont les programmes à restructurer, et à quel degré.

Pour le formaliser, dénotons par

Ci, le coût de restructuration pour le programme i

Eih, l'efficacité à atteindre pour le programme i dans la période h (Ei = max Eih)

Oh, l'amélioration à réaliser pour la période h, déterminée à l'étape précédente.

Si de plus on admet que le coût de restructuration est fonction de l'efficacité à atteindre : Ci = Fi (Ei)

$$\min \sum Fi\,(Ei)$$

$$\begin{cases} \sum_{1} Ei1 \geqslant O1 \\ \vdots \\ \sum_{1} Eih \geqslant Oh \end{cases}$$

La troisième étape consiste à déterminer la procédure de restructuration à choisir pour chacun des programmes à restructurer, et à effectuer les restructurations.

Le but de ce mémoire est de réaliser un outil qui permette de comparer, tant du point de vue coût qu'efficacité, une série de procédures de restructuration, de façon à effectuer le choix de la procédure ad hoc. Cet outil doit en même temps être utilisable pour effectuer les restructurations en question.

La dernière étape consistera en une vérification des résultats obtenus et une éventuelle mise en cause de la valeur des procédures choisies.

2.3. Comparaison des procédures de restructuration

La comparaison, comme évoqué précédemment, doit se faire à deux niveaux : coût et efficacité. Il nous faut donc déterminer des critères qui permettent de la réaliser.

a)point de vue coût :

Les critères sont ceux ayant rapport aux ressources consommées pour effectuer la restructuration : temps CPU, temps de connexion, activité des périphériques, mémoire utilisée ...

b)point de vue efficacité :

L'efficacité de la restructuration se traduit à deux niveaux : celui de la pagination et celui de l'utilisation de la mémoire.

Les critères les plus communément cités sont :

- le nombre de défauts de page
- Parachor curve
- taille du working set moyen et maximum
- nombre de pages en excès
- "Page survival index"
- temps moyen entre deux défauts de page

et d'autres pour lesquels nous revoyons à la bibliographie (03,07,11,17,24)

CHAPITRE 3 : RESTRUCTURATION ET OUTILS

3.1. Etapes classiques de la restructuration (12,25)

Les étapes classiques de la restructuration, ainsi que les hypothèses faites sont exposées dans ce qui suit.

Etape 1 : Définition de blocs relocatables

Le programme est découpé en blocs relocatables qui seront arrangés dans un ordre nouveau afin d'atteindre l'objectif relatif au nombre de défauts de page provoqués.

Une première hypothèse concerne la dimension de ces blocs qui, idéalement, doivent être de petite taille vis à vis de la taille de la page du système considéré. Les chiffres de 1/10 à 1/3 sont généralement avancés. (17)

Une autre hypothèse est que l'on s'interdit de descendre à l'intérieur des blocs définis : leur structure ne sera pas remise en cause. Il n'y aura pas discrémination entre zones contenant des données et zones d'instructions constituant ces blocs.

Etape 2 : Acquisition des données

Suivant le type des algorithmes de l'étape 3, les données sont acquises de façon différente.

- elles sont enregistrées lors d'une exécution du programme à restructurer. Celui est espionné. Les références faites sont utilisées directement, ou stockées de façon à constituer une chaîne. La finesse du suivi du programme dépend de l'outil disponible pour la prise d'information et peut aller de l'interprètation au simple suivi des passages de contrôle entre blocs. Le coût varie en fonction : de 50 fois le coût original dans le premier cas (23), à moins de 10% de supplément dans le second. Les références aux zones de données peuvent être ou non prises en compte. L'échantillonnage est une possibilité supplémentaire.

- elles sont déduites de la structure du programme - par exemple sur base de la source - où les possibilités potentielles de passage de contrôle entre blocs, de référence aux zones de données ... sont étudiées.

Etape 3 : Calcul d'affinité entre blocs (algorithme de "restructuration")

Une matrice d'affinité qui estime les liens plus ou moins grands existant entre les blocs, est constituée.

Ce point est développé au Chapitre 5.

Cette matrice est créée par les algorithmes dits de restructuration ("restructuring").

Etape 4 : Définition du nouvel arrangement

Les blocs relocatables sont groupés dans une ou plusieurs pages, sur base de la matrice construite à l'étape 3. Ce sont les listes créées par les algorithmes de "clustering" examinés au chapitre 6. Ces listes sont elles-mêmes séquencées. La possibilité de dédoubler certains blocs, c'est-à-dire d'en inclure plusieurs exemplaires dans des listes différentes ne sera pas prise en considération. Celle de tasser les listes afin de réduire la consommation mémoire - densification - le sera (chapitre 7).

Etape 5 : Création du programme restructuré

L'arrangement défini est réalisé par une édition de liens (LINKAGE-EDITOR) de façon à placer chacun des blocs dans l'ordre défini.

Pour cette étape, il est nécessaire de disposer de la bibliothèque contenant les modules objets correspondant aux blocs constituant le programme. Nous ferons de plus l'hypothèse suivante : le bloc contenant le point d'entrée initial peut ne pas être le premier inclus lors de l'édition de liens. L'arrangement effectué est en effet indépendant des contraintes propres à l'éditeur de liens. Si ce n'est le cas, il faudra réviser légèrement la séquence obtenue de façon à répondre aux contraintes techniques du système sur lequel on travaille.

Etape 6 : Test du nouveau programme

Le comportement du programme modifié est testé de façon à vérifier que les améliorations répondent aux objectifs assignés. Cette étape n'aura plus de raisons d'être dès que les assurances relatives à la validité des procédures seront suffisantes.

N.B. : Un enchaînement du type restructuring - clustering - densification sera appelé "procédure de restructuration".

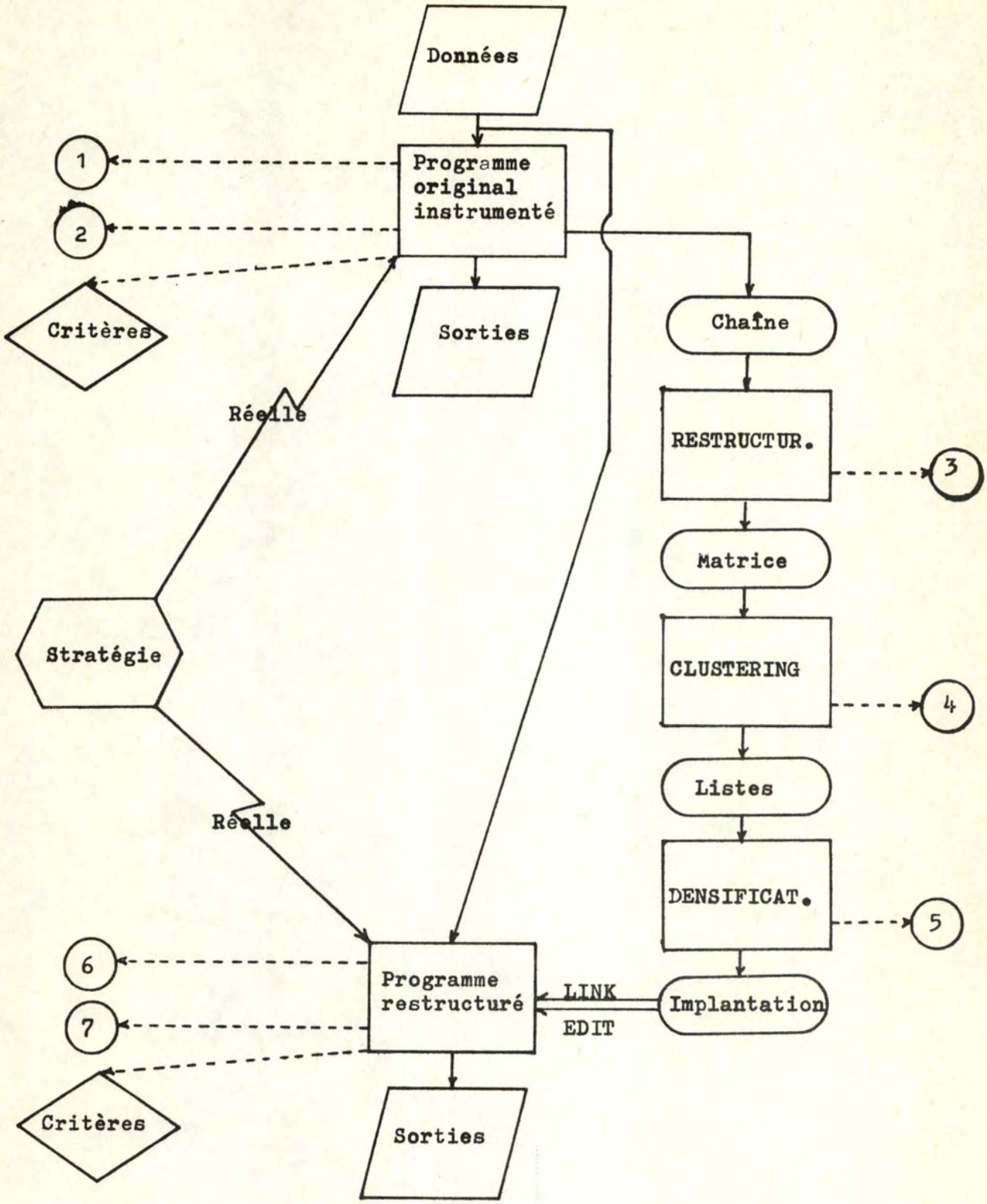
3.2. Schémas possibles

Nous ne considérons ici que l'approche basée sur les données obtenues lors de l'exécution du programme à restructurer. Cette option sera justifiée plus loin (chapitre 5).

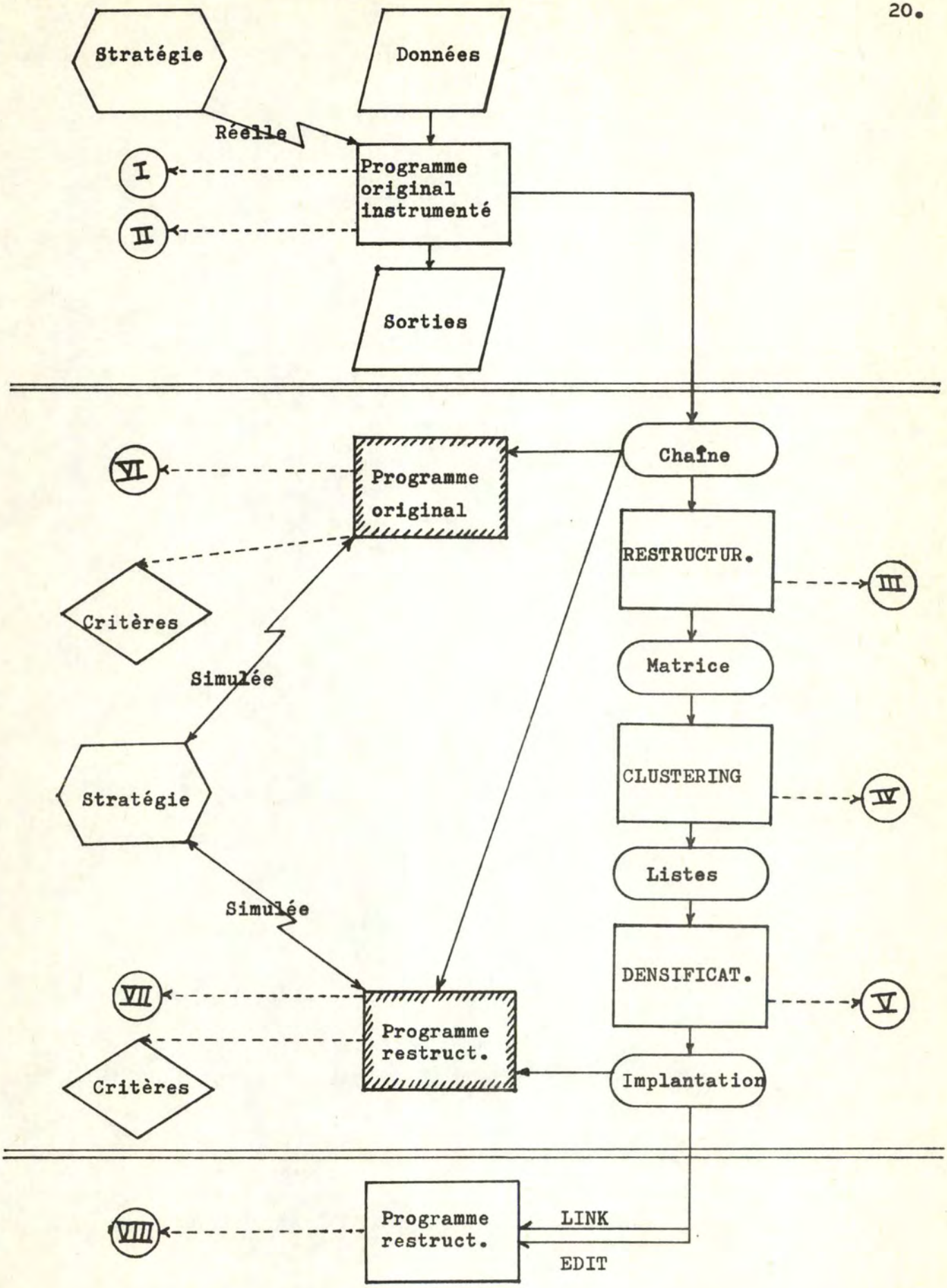
Essentiellement deux possibilités peuvent être considérées : la restructuration s'effectuera de façon "on line" ou différée. Dans le premier cas, l'avantage est de ne pas devoir enregistrer (entièrement) la chaîne des références et de suivre l'évolution du programme en temps réel, entraînant une connaissance plus concrète de son comportement. (23) Pour ce qui est de notre propos, nous ne considérerons que le deuxième cas.

Les schémas suivants représentent les différentes étapes citées au point précédent, leurs entrées et sorties, ainsi que les coûts afférents.

CADRE REEL


Données
Programme original instrumenté
1
2
Critères
Sorties
Réelle
Stratégie
Réelle
Chaîne
RESTRUCTUR.
3
Matrice
CLUSTERING
4
Listes
DENSIFICAT.
5
Implantation
LINK
EDIT
Programme restructuré
6
7
Critères
Sorties

CADRE SIMULATION



Deux cadres sont représentés : réel ou simulé.

Le coût total pour restructurer un programme est la somme des coûts représentés et passés en revue ci-après. La notation

= signifie l'égalité des coûts spécifiés pour chacun des deux cadres considérés.

1 = I coût d'instrumentation du programme original, incluant une éventuelle prise de copie et passage par le "linkage editor" .(Cfr infra)

2 , II coût d'exécution du programme instrumenté. La chaîne de référence est enregistrée.

2 = II + coût d'extraction des valeurs des critères retenus pour la comparaison.

3 = III

4 = IV

5 = V coût d'application des algorithmes sélectés : respectivement, "restructuring", "clustering",densification.

VI coût de simulation du programme original, basée sur l'implantation existante des blocs.

VII coût de simulation du programme restructuré, basée sur la nouvelle implantation fournie par la procédure de restructuration sélectée.

VIII = 6
coût de"relinkage"

7 coût d'exécution du programme restructuré avec extraction des valeurs des critères

Il est essentiel de pouvoir examiner les résultats de la restructuration, ce qui se fera, sur base de critères définis ci-après, en comparant l'exécution des programmes original et restructuré. Si on travaille dans le cadre réel, il faudra exécuter programme original et restructuré avec les mêmes données, ce qui n'est pas un problème majeur. Cependant, une des deux exécution est inutile puisqu'elle doit conduire aux mêmes résultats. Ce qui n'est pas obligatoirement plus coûteux que les deux simulations nécessaires dans l'autre cas. Mais le problème le plus ardu réside dans le fait que, dans le cadre réel toujours et pour l'exécution du programme remodelé, il va être indispensable de reconstituer exactement les mêmes conditions, en clair, la charge, que lors de la première exécution. Ceci afin que la comparaison soit valable. C'est à la fois très difficile et très coûteux. Par contre, la simulation place les deux programmes exactement dans les mêmes conditions et permet l'indépendance par rapport à l'environnement (charge et système). Le coût est de plus modéré. D'autre part, il est beaucoup plus facile d'obtenir les valeurs des critères néces aires à la comparaison par le biais de la simulation que par celui de la mesure sur le système réel. Le processus de simulation est d'ailleurs utilisé par tous les auteurs consultés.

3.3. Critères de comparaison retenus

Considérant que nous devons réaliser un outil pratique, c'est-à-dire à la fois facilement utilisable, réaliste et modérément coûteux, nous avons choisi les critères de comparaison entre programme original et restructuré dans cet esprit. Ces critères serviront également à comparer les différents procédures de restructuration.

Les coûts de chacune des étapes afférentes à la restructuration sont exprimés en termes de temps CPU consommé.

L'efficacité de la restructuration sera mesurée sur base des critères suivants :

- nombre de défauts de page ;
- taille du working set moyen ;
- taille du working set maximum.

Le working set étant l'ensemble des pages utilisées, c'est-à-dire référencées, par un processus pendant un intervalle de temps déterminé (cfr 5.1.2.)

3.4. Les contraintes

Nous nous proposons ici de définir les contraintes à respecter par l'outil de restructuration à mettre en place.
En premier lieu, un impératif évident est de respecter l'intégrité du programme à restructurer. La restructuration ne peut en aucun cas conduire à une modification des résultats produits par le programme en question, ni être source d'erreurs.
D'autre part, l'exécution du programme réalisée pour obtenir les données nécessaires à la restructuration ne doit pas être perdue et reste utile pour l'utilisateur tout comme une exécution "ordinaire".
En ce qui concerne les coûts, la restructuration doit être possible sans dépasser des bornes acceptables, ce qui annulerait son utilité. Dans cette optique, certains algorithmes seront délaissés. Modifications dans les programmes et re-compilations sont à éviter. Dans la mesure du possible, la restructuration d'un programme devra se faire indépendamment des langages de programmation et du système. Notons cependant que certains algorithmes sont mieux adaptés à une stratégie de gestion mémoire qu'à une autre, entraînant donc des résultats de qualité variable suivant le système sur lequel le travail est réalisé.

Il est cependant évident qu'en contrepartie, un certain nombre d'hypothèses vont être faites concernant les programmes à restructurer, en relation avec les outils qui seront utilisés.
Nous tâcherons de les limiter à un minimum.

3.5. Les outils

Les schémas précédents (3.2.) font ressortir la nécessité de disposer de deux outils annexes à la procédure de restructuration proprement dite : le mesureur qui extrait la chaîne de référence lors de l'exécution du programme à restructurer, et le simulateur qui permet la comparaison entre programme original et restructuré.

3.5.1. Le mesureur

Il possède la structure proposée par Bergeron et Bulterman (05) et respecte les conditions posées par Holtwick (18).

L'hypothèse faite sur les programmes est qu'ils respectent les conventions standard de "linkage" en O.S..

Rappelons brièvement que toute routine appelante doit mettre à la disposition d l'appelée une zone (SAVE AREA) où seront stockés les registres généraux dans l'état où ils se trouvent lors de la réception du contrôle par ce dernier, les "save areas" étant doublement chaînées.

Les instructions standard utilisées sont de la forme :

```
L      13,    = A (SAVE)    chargement adresse de la
                             zone de sauvetage
L      15,    = V (APPELE)  chargement adresse de
                             branchement
BALR   14,15                 chargement adresse de
                             retour ; Branchement
 .
 .
 .
```

```
STM    14,12,12(13)          sauvetage des registres de
 .                           l'appelant
 .
 .
LM     14,12,12(13)          restauration

BR     14                    retour
```

Les conventions sur les registres sont les suivantes :

R15 : adresse de branchement

R14 : adresse de retour

R13 : adresse de la zone de sauvetage (SAVE AREA)

L'appelant a la responsabilité de les charger ; l'appelé doit effectuer les sauvetage et restauration nécessaires.

Ainsi, pour intercepter le contrôle lors d'un appel, il suffit de détecter l'instruction STM. Le contenu du registre 14 à cet instant donne l'adresse de retour.

Le STM sera remplacé par une instruction d'appel superviseur SVC qui , créant une interruption, provoque le déroutement lors de la réception du contrôle , et par un numéro de routine.

Le contenu du registre 14 sera modifié pour permettre un nouveau déroutement lors du retour à l'appelant.

Lors d'un appel, les informations suivantes sont enregistrées :

- registres 14, 15 ;
- n° routine ;
- instant CPU (exprimé en 10^{-4} secondes).

Lors d'un retour, l'instant CPU et un indicatif de retour sont mémorisés. Périodiquement, ces informations sont vidées sur une bande. Toutes les précautions sont prises pour respecter l'intégrité du programme.

Les interférences entre le mesureur et le programme mesuré, notamment au point de vue temps CPU consommé, sont pratiquement nulles.

L'obtention des données nécessaires à la restructuration se fera donc en deux étapes :

- instrumentation du programme : détection des instructions STM et remplacement par un SVC avec affectation du numéro de routine ;
- exécution du programme instrumenté et création de la bande contenant les données désirées.

L'instrumentation peut se faire sur le programme chargeable (Load module) ou sur la bibliothèque des modules objets, auquel cas une édition de liens sera nécessaire.

Remarques supplémentaires :

- les routines qui ne respectent pas les conventions standard doivent être rendues standard ou seront négligées ;

- lorsque le programme à restructurer a été rédigé dans un langage "évolué" (ex. : FORTRAN, COBOL), une série de modules supplémentaires sont inclus pour la construction du programme chargeable. Il existe un problème pour ces routines qui ne respectent pas toujours les conventions standard, et dont l'utilité n'est pas toujours très claire. Si on désire les faire participer à la restructuration, il faudra modifier la librairie sur laquelle elles se trouvent, ce qui entraîne un risque difficile à évaluer. Elles n'ont pas été prises en considération.
- la "routine" sera "l'unité de restructuration". Autrement dit : bloc relocatable = routine.

3.5.2. Les simulateurs

Afin de valider les résultats, et pour ne pas pénaliser les algorithmes orientés dans le sens d'une stratégie de gestion mémoire particulière comme exprimé lors du passage en revue des différents algorithmes, nous avons implémenté deux simulateurs. Les stratégies choisies, à savoir LRU (Least Recently Used) et working set pur, sont suffisamment simples pour que le coût de la simulation ne soit pas excessif. Il aurait été inutile de compliquer ces simulateurs pour être au si proche que possible de l'une ou l'autre stratégie réellement implémentée sur un système réel.

En effet, en plus du coût de l'opération, le résultat atteint serait de toute façon resté à un stade approximatif vu le nombre élevé d'éléments différents pris en considération sur un système opérationnel. De plus, la généralité en aurait souffert. Nous avons simplement essayé de voir si deux approches différentes dans leur philosophie, qui sont d'ailleurs des standards de référence, conduisent à des résultats semblables. Toujours avec le soucis de ne pas pénaliser certains algorithmes. Le comportement du programme restructuré et original est simulé sur base de la bande extraite lors de l'exécution. Il suffit de disposer d'une table qui fasse correspondre à chaque routine (bloc relocatable) identifiée par un numéro, son adresse début et fin dans chacune des deux implémentations : originale et restructurée.

Les adresses enregistrées sur la bande (registres 14 et 15) sont exploitées directement, ou sont transformées pour obtenir la nouvelle adresse et partant, la page référencée.

3.5.2.1. Simulateur LRU (least recently used)

Les paramètres de fonctionnement sont :

- la taille de la page considérée ;
- le nombre de pages allouées.

Les résultats sont exprimés en nombre de défauts de page détectés. Au début de l'exécution, un certain nombre de cadres de mémoire centrale sont alloués au programme. Cet espace reste constant durant toute l'exécution. Les pages référencées sont empilées par instant de référence croissant, les plus récemment utilisées se trouvant au sommet.

L'enregistrement des instants des références sur la bande n'est pas nécessaire.

A chaque référence, deux cas peuvent se présenter. Ou bien la page référencée est déjà dans la pile. Auquel cas cette dernière est mise à jour de façon à ce que cette page rejoigne le sommet. Si la page référencée n'est pas reprise dans la pile, il se produit un défaut de page. La page du fond de la pile, c'est-à-dire la plus "vieille", est éjectée et la nouvelle est placée au sommet, les autres reculant d'une place.

3.5.2.2. Simulateur working set

Les paramètres de fonctionnement sont :

- la taille de la page considérée ;
- la fenêtre du working set.

Les résultats sont exprimés en nombre de défauts de page détectés, ainsi que par la taille du working set moyen

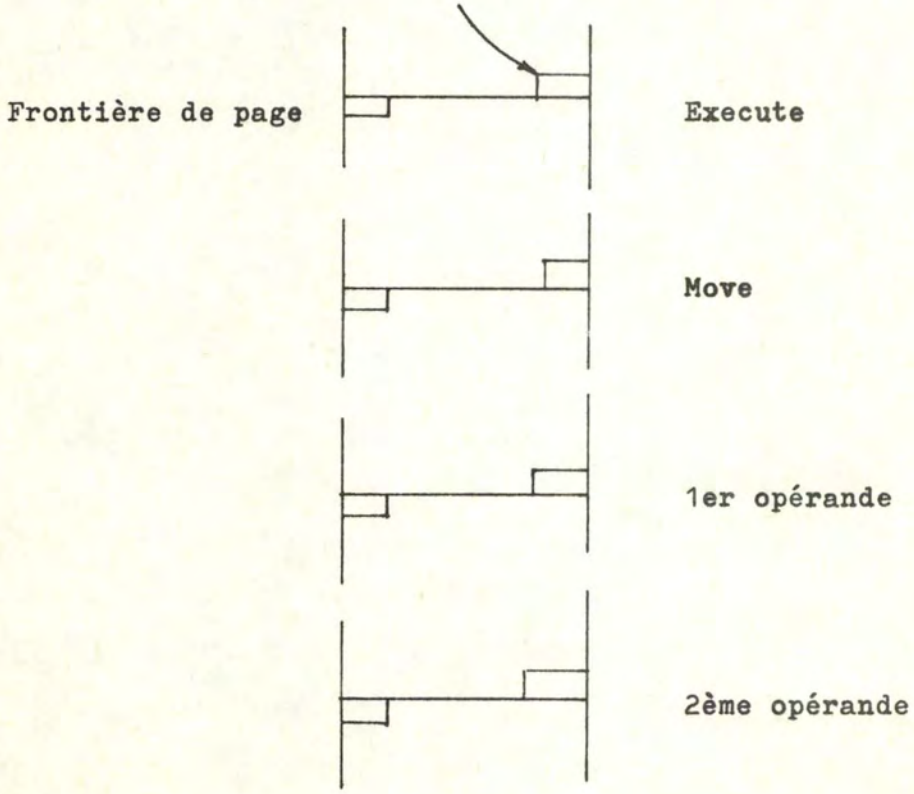
et maximum observés pendant la simulation.

Il faut disposer des instants des références.

Le working set de départ est constitué par l'ensemble des pages référencées durant une période de temps correspondant à la fenêtre considérée. A partir de cet instant, lors de l'examen de chaque nouvelle référence venant de la bande, les pages qui ne font plus partie du working set sont éliminées. Ce sont celles dont l'instant de la dernière référence est inférieur à l'instant courant moins la fenêtre. Deux cas peuvent se présenter. La nouvelle page fait partie du working set actuel et son enregistrement de temps est mis à jour. Sinon, un défaut de page est détecté ; la nouvelle page est incluse dans le working set et son enregistrement de temps est mémorisé. Le nombre de pages qui constituent le working set varie dynamiquement. La taille moyenne et maximale est calculée en conséquence.

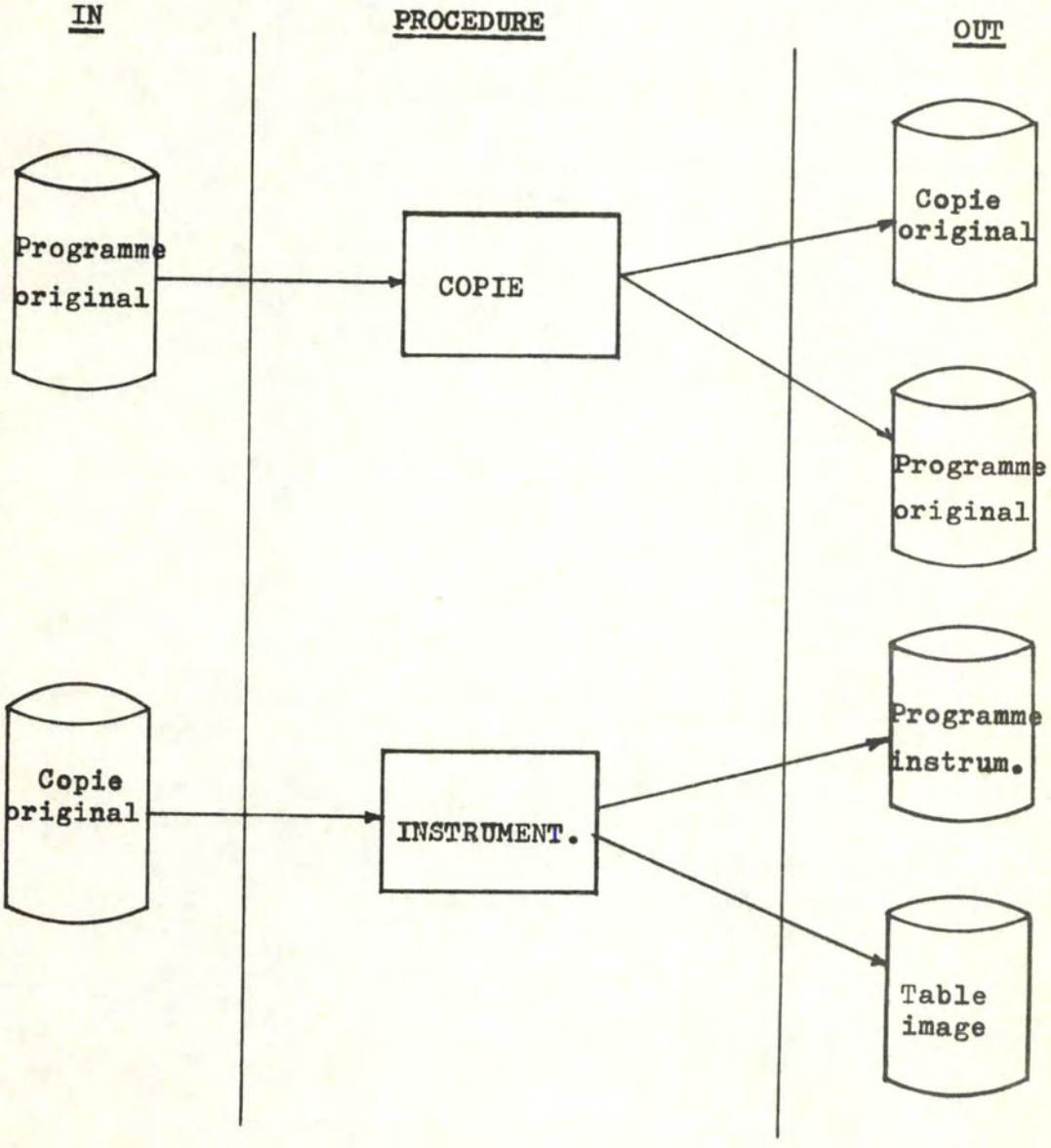
Une faiblesse des deux simulateurs décrits est qu'ils ne prennent en compte qu'un seul défaut de page à la fois. Dans la réalité, un programme peut réclamer plus qu'une seule page pour poursuivre son exécution. Le nombre maximum est de 8 défauts de page simultanés pour la même instruction, dans le cas BS 2000 (28). Cette remarque est illustrée par l'exemple suivant. Supposons qu'aucune des pages représentées ne se trouve en mémoire et considérons les instructions :

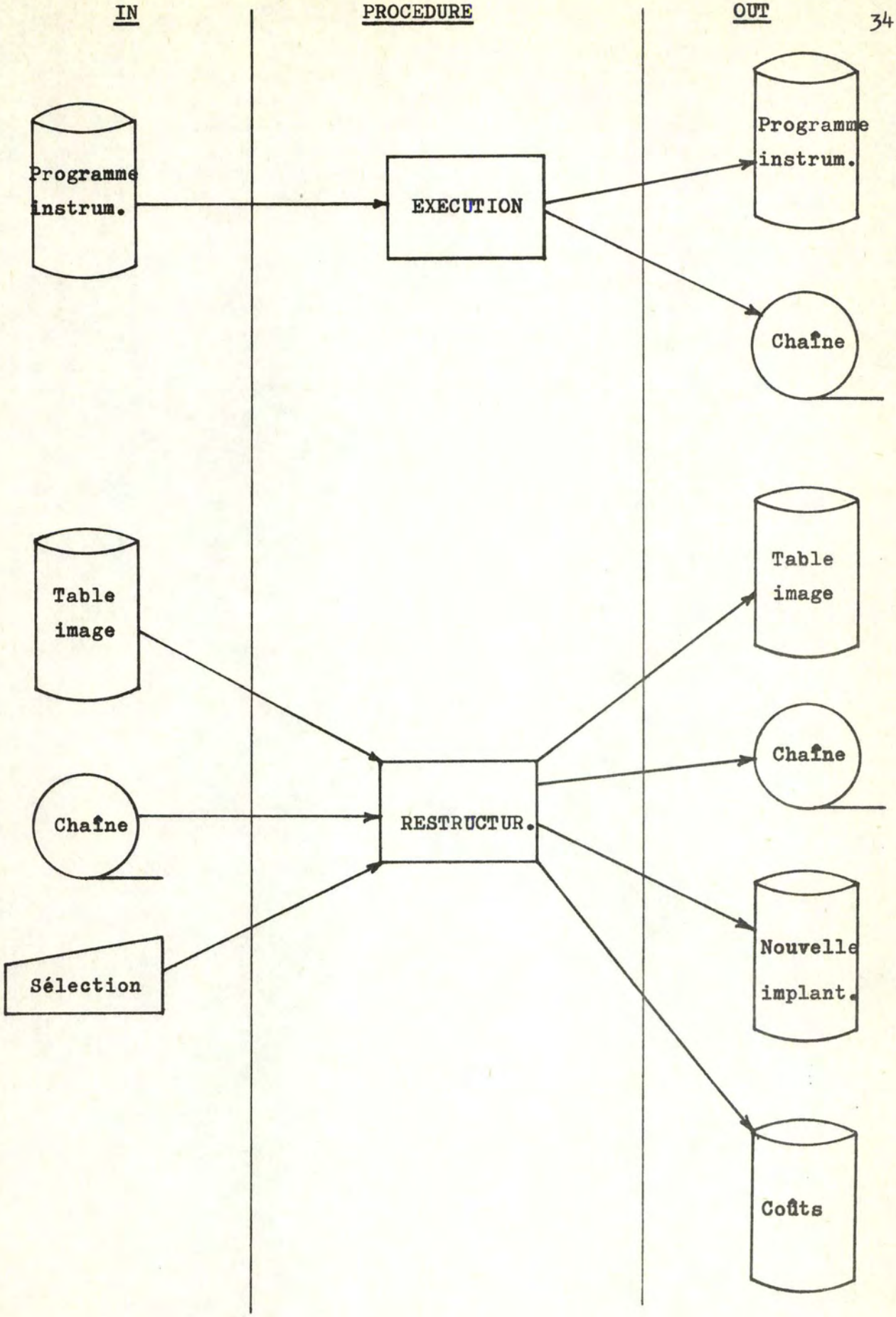
```
        EX      R, ADR
        .
        .
        .
ADR     MVC     ZONE 1, ZONE 2
        .
        .
        .
ZONE 1  DS      CL 10
        .
        .
        .
ZONE 2  DS      CL 10
```

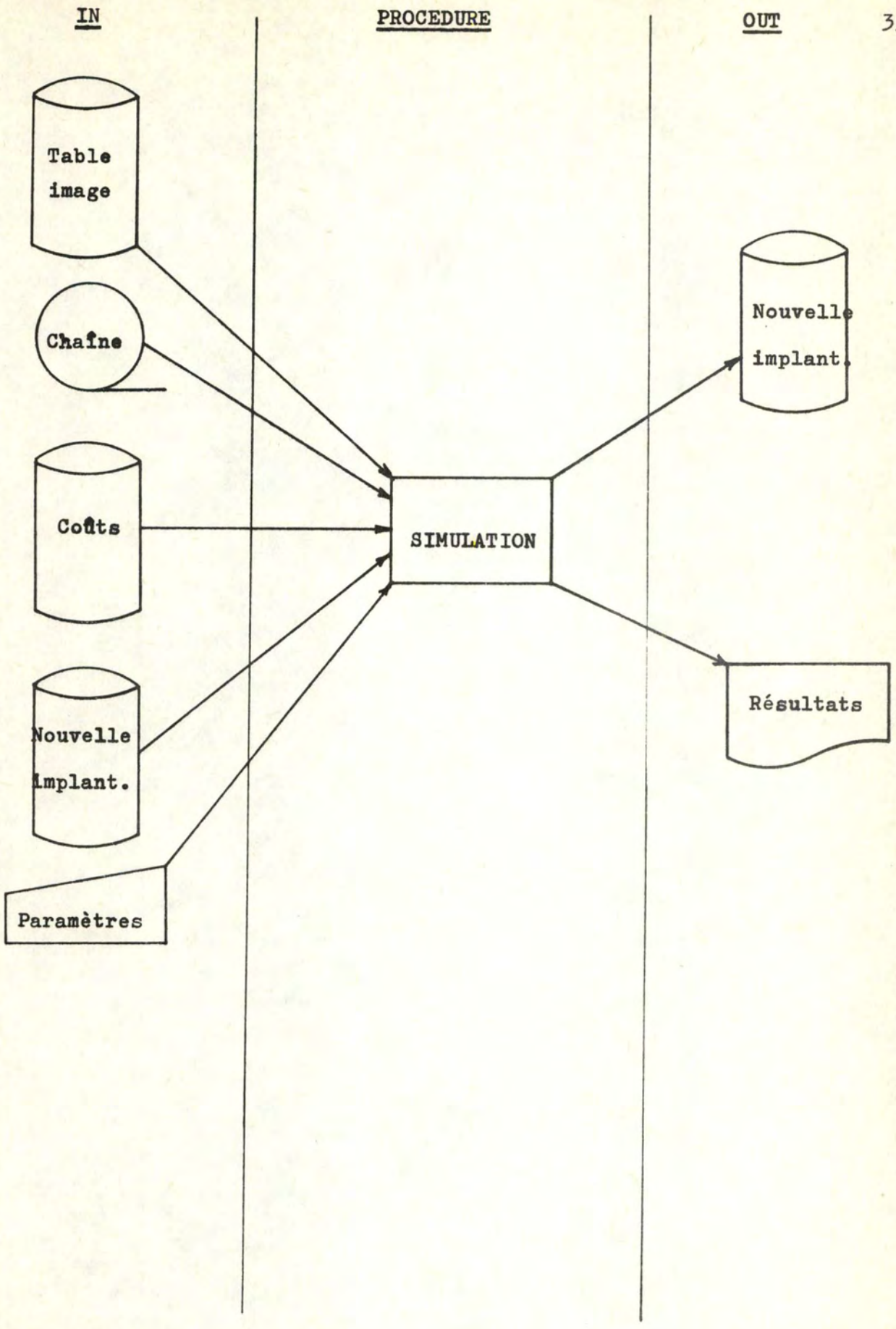


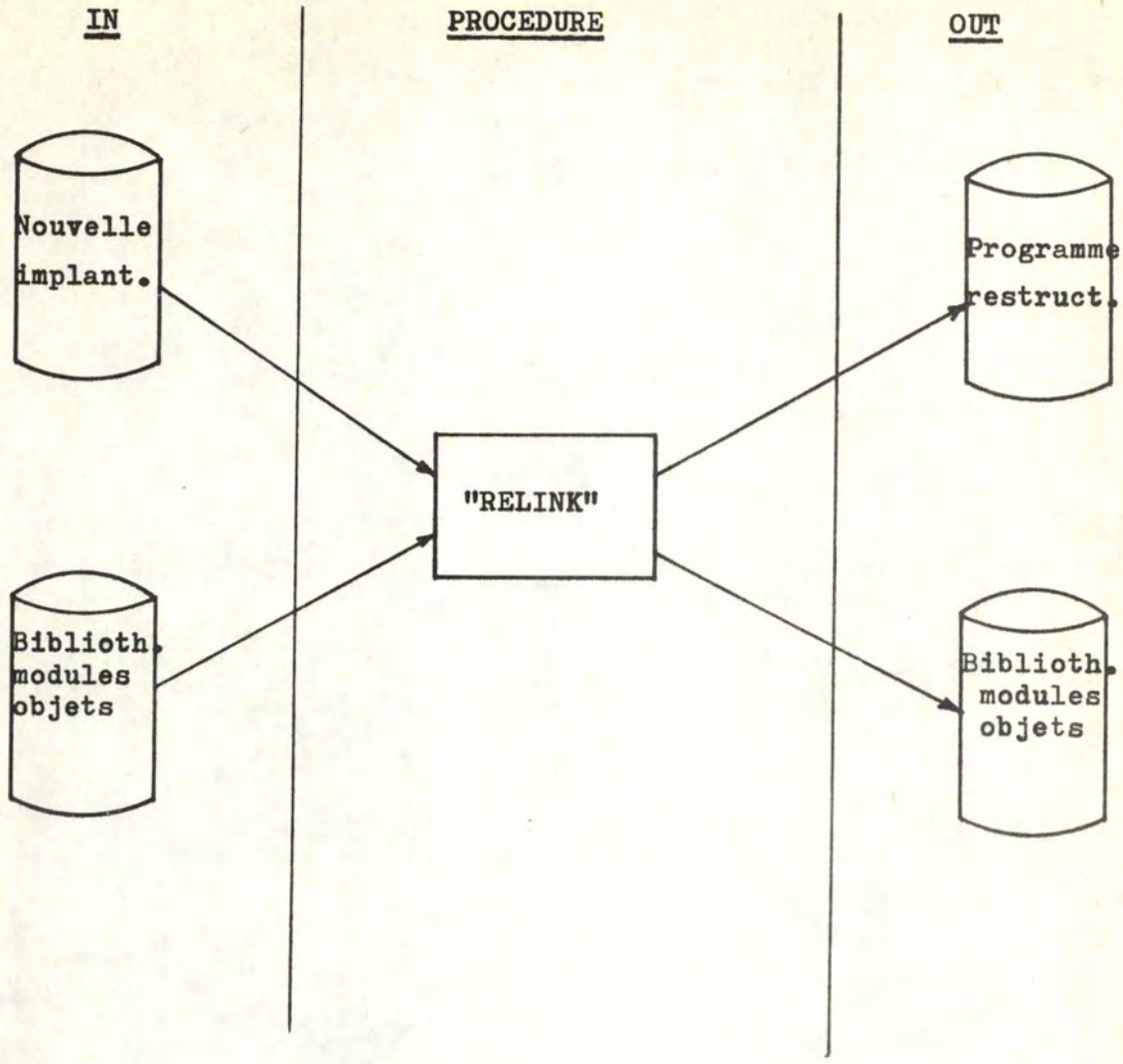
CHAPITRE 4 : STRUCTURE DE L'OUTIL DE RESTRUCTURATION IMPLEMENTE

Sont reprises ici les différentes opérations nécessaires pour effectuer la restructuration, ainsi que les fichiers associés. Les supports choisis pour leur implantation sont également symbolisés. Sous la rubrique IN sont notés les fichiers d'entrée ; sous la rubrique OUT, tous les fichiers qui se retrouvent à la sortie, soit qu'ils servent aux stades suivants, soit qu'ils sont à conserver.

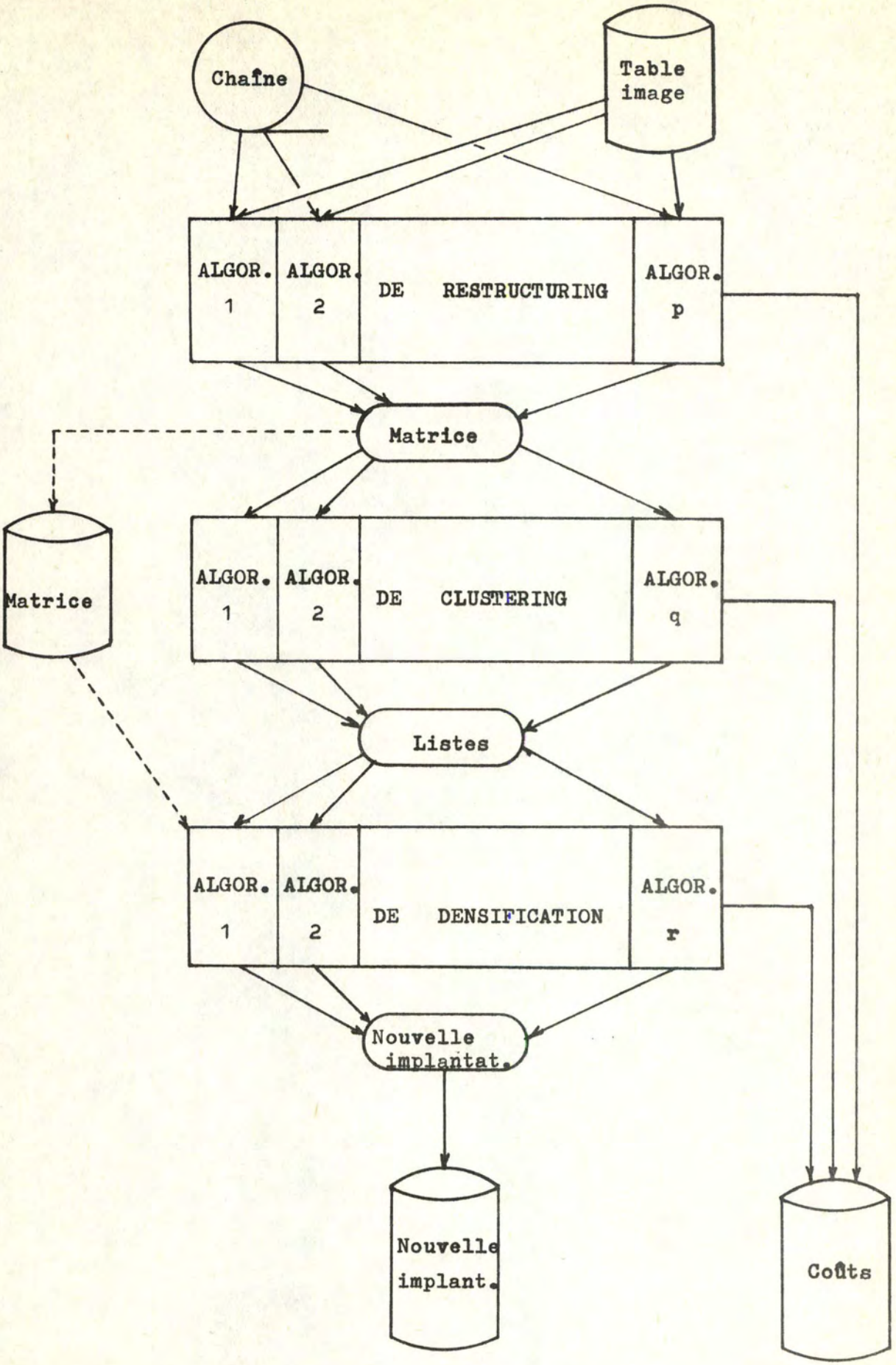



IN
PROCEDURE
OUT
Programme instrum.
EXECUTION
Programme instrum.
Chaîne
Table image
Table image
Chaîne
Chaîne
RESTRUCTUR.
Nouvelle implant.
Sélection
Coûts


IN
PROCEDURE
OUT
Table image
Chaîne
Coûts
Nouvelle implant.
Paramètres
SIMULATION
Nouvelle implant.
Résultats



La structure de l'outil de restructuration propement dit est donnée ci-après. Le soucis majeur lors de la réalisation a été de créer un outil qui soit utilisable pour des tâches aussi diverses que possible. Citons comme exemples : comparer des procédures de restructuration différentes, restructurer unprogramme à l'aide d'une procédure bien précise,déterminer l'influence de la taille de la page sur la valeur d'un algorithme de "clustering",...


Chaîne
Table image
ALGOR. 1
ALGOR. 2
DE RESTRUCTURING
ALGOR. p
Matrice
Matrice
ALGOR. 1
ALGOR. 2
DE CLUSTERING
ALGOR. q
Listes
ALGOR. 1
ALGOR. 2
DE DENSIFICATION
ALGOR. r
Nouvelle implantat.
Nouvelle implant.
Coûts

CHAPITRE 5 : LES ALGORITHMES DE RESTRUCTURATION (RESTRUCTURING)

5.1. INTRODUCTION

5.1.1. Autour du concept d'affinité

L'exécution d'un programme peut être représentée par une chaîne composée de toutes les références aux blocs constituant ce programme, ainsi que des instants de ces références.

L'affinité entre deux blocs quelconques est alors assimilée à la tendance qu'ont les références à ces deux blocs, d'apparaître de façon conjointe dans la chaîne (16).

Ceci constitue une approche dynamique du concept d'affinité.

Cette approche fait une hypothèse implicite quant à la stabilité du comportement du programme suivi en fonction des données d'entrée, ce qui est généralement le cas (12,25).

De là, la nécessité d'observer le programme à restructurer lors d'une exécution basée sur des données "typiques".

(12,17,25)

Une autre approche, statique cette fois, est de considérer l'affinité comme l'éventail des possibilités qu'il y ait, à l'exécution, passage de contrôle - ou référence - d'un bloc vers - à - un autre. (02,20,27)

L'affinité dynamique dépend de

- la proximité des références aux deux blocs, cette proximité étant exprimée en terme de distance entre références dans la chaîne ou en terme de temps écoulé entre les références ;
- la fréquence d'apparition conjointe.

L'affinité "statique" néglige ces aspects (Le rapport code exécuté / code total est généralement faible : cfr routines de traitement d'erreurs et de cas particuliers, routines d'initialisation et de clôture rarement utilisées... (19)) pourtant fondamentaux au niveau de la stratégie de gestion mémoire qui va conditionner le comportement de pagination du programme.

Les algorithmes de restructuration "statique" seront donc délibérément passés sous silence.

5.1.2. Principe général des algorithmes

Le principe est de constituer une matrice d'affinité entre blocs en analysant la chaîne de références - et de temps - enregistrée lors de l'exécution du programme à restructurer.

A tout couple de blocs distincts est associée une valeur reflétantleur affinité.

Deux aspects doivent être pris en considération :

- la proximité dans la chaîne, de références à deux blocs définis ;
- l'éloignement de ces occurences simultanées.

Pour éclairer ce point important, considérons l'exemple suivant : (22)

Soient deux chaînes de longueur 100 000.

Dans la première, des références consécutives aux blocs I et J apparaissent une fois toutes les 1 000 références, soit 100 fois au total.

Dans la deuxième,des références consécutives aux blocs I et J apparaissent 100 fois entre la 10 000ème et la 11 000ème soit toujours 100 fois au total.

Il est bien clair que le groupement de I et J dans la même page est plus intéressant dans le premier cas que dans le second,du point de vue charge mémoire.

Dans la négative, le nombre de défauts de page avoisinerait 100 dans le premier cas et vraisemblablement un seul dans le second.

L'affinité calculée devrait donc tenir compte de la distance - en nombre de références ou en temps - entre apparitions conjointes.

De façon générale, les algorithmes sont basés sur la propriété de localité vérifiée par la plupart des programmes (21,24) qui, durant leur exécution, comportent alternativement des zones stables et instables. Dans une zone stable, les références se rapportent au même ensemble de blocs. Schématiquement, on peut représenter l'exécution comme suit :
après une période instable correspondant au chargement et initialisation, le processus se déplace vers une zone stable qui conserve le contrôle du processeur pendant un certain temps. Cette phase terminée, le contrôle du processeur va évoluer vers une autre zone stable qui sera atteinte après un nouveau laps de temps ...

Le working set w(t,T) représente l'ensemble des blocs référencés entre l'instant t - T et l'instant t (09)

M (ti, ti+1) = w (ti+1, T) \w(ti, T) est appelé ensemble des blocs manquants

E (ti, ti+1) = w (ti, T)\ w(ti+1,T) est appelé ensemble des blocs en excès. (12)

Afin de réduire le nombre de défauts de page, il s'avérera utile de grouper un bloc faisant partie du working set es-

timé avec un bloc manquant, et un bloc en excès avec un bloc faisant partie du working set estimé.
La matrice construite peut être rendue symétrique puisque l'affinité entre I et J est la même que celle de J avec I. De plus, les affinités du type (I,I) n'ont pas de sens pour notre propos.
L'implémentation des algorithmes est réalisée en ce sens et nous nous contenterons donc d'une sous-matrice triangulaire.

5.1.3. Remarques

- la chaîne de référence devrait normalement contenir :
  1° les passages de contrôle entre blocs = références aux instructions.
  2° les références aux données, paramètres ... contenus dans les blocs.

  L'outil de mesure dont nous disposons ne permet malheureusement pas de tenir compte du point 2. Ceci ne change en rien la philosophie et la démarche reste valable, bien que les informations de départ soient amputées de données importantes (10).

- Une autre remarque concerne la taille de pareilles chaînes. Vu les caractéristiques (taille - temps d'exécution) des programmes à restructurer, il va s'avérer utile, voire

nécessaire d'échantillonner la chaîne de référence. Certains algorithmes sont spécialement conçus en ce sens (MASUDA)

- Notons aussi que la restructuration est quelquefois orientée dans le sens d'une stratégie de gestion mémoire déterminée (CWS, AB sont orientés "working set").

5.2. RELEVE

5.2.1. Algorithmes AB (12)

Cet algorithme est basé sur la notion de working set. Toute entrée (i,j) de la matrice est incrémentée de 1 chaque fois que l'une des deux situations suivantes se présente :

$i \in w(tk,T)$ et $j \in M(tk, tk+1)$ $\quad (T = tk+1-tk)$

ou $i \in E(tk,tk+1)$ et $j \in w(tk+1,T)$

Chaque nouvelle référence j est donc comparée aux anciennes (celles qui constituent le working set actuel) et les entrées (i,j) sont incrémentées de une unité $\forall i \in w(tk,T)$ chaque fois que j n'appartient pas à ce même working set. De plus, il faudra attendre la fin de la période tk+1 (soit l'instant tk+1 +T) pour pouvoir déterminer E (tk,tk+1) et incrémenter les entrées de la matrice.

La valeur j est conservée pour construire w (tk+1, T) qui remplacera w(tk, T) lorsque la première référence de temps supérieur à tk+1 +T sera atteinte.

L'algorithme tient compte à la fois des blocs "manquants" et des blocs en "excès". Il favorise le groupement des blocs "manquants" avec les blocs précédemment utilisés et tend à minimiser le nombre moyen de pages occupées en mémoire centrale.

D'autre part, il favorise le groupement des blocs "en excès" avec les blocs utilisés ensuite et tend à maintenir le plus longtemps possible en mémoire centrale, toute page qui y a été amenée.
Le résultat escompté est d'obtenir un working set aussi petit et stable que possible.

5.2.2. Algorithme A (12)

C'est une simplification du précédent : il ne tient compte que des blocs dits "manquants" et permet la mise à jour immédiate de la matrice à chaque nouvelle référence.

5.2.3. Algorithme B (12)

Il résulte également d'une simplification de l'algorithme AB et ne tient compte que des blocs dits "en excès".

5.2.4. Algorithme CWS (Critical Working Set) (11,14)

Toujours basé sur la notion de working set, cet algorithme recherche les "références critiques".
Le working set de départ étant constitué, l'entrée suivante j de la chaîne de référence est examinée et considérée comme référence critique si le bloc qu'elle stipule n'appartient pas au working set à cet instant. Les éléments (i,j) de la matrice, $\forall i \in$ Working set sont alors incrémentés d'une unité. Le working set est mis à jour en

tenant compte de cette référence et la suivante dans la chaîne examinée à son tour ...

CWS peut être considéré comme un algorithme A continu où le ti varie à chaque référence examinée.

Cet algorithme tend à minimiser le nombre de défauts de bloc provoquant des défauts de page.

5.2.5. Nearness Matrix. (17)

Pour toute paire de références (i,j) contigües dans la chaîne, l'élément (i,j) de la matrice est incrémenté d'une unité.

La chaîne est donc examinée en travers d'une fenêtre de longueur 2. L'algorithme tend donc à grouper ensemble les blocs qui soient les plus proches dans la séquence d'exécution du programme.

Les éléments (i,j) de la matrice ainsi constituée représentent à tout instant le nombre de passages directs de contrôle (ou d'accès) de $i \leftrightarrow j$.

L'algorithme ne demande pas l'enregistrement d'informations de temps lors de l'extraction de la chaîne. Il en sera de même pour les suivants.

5.2.6. Algorithme de RYDER avec fenêtre de taille m et vecteur d'incrémentation b de dimension m-1 (25)

Le principe est le même que précédemment, maisla taille de la fenêtre d'observation n'est pas réduite à 2.
Le vecteur d'incrémentation donne la valeur à ajouter aux éléments (i,j) en fonction de leur distance respective dans la chaîne. La seule restriction faite sur les valeurs contenues dans le vecteur b est qu'elles doivent être décroissantes. L'auteur préconise une fenêtre de taille 6 et un vecteur b = (5,4,3,2,1). Cette recommandation est corroborée dans les observations faites par Gaspard.

5.2.7. Algorithmes de Masuda (22)

Ils sont basés sur un échantillonnage périodique de la chaîne de référence.
Cet échantillonnage peut se faire :

- sur base des références seules : les échantillons, de taille fixe et comportant m références, sont alors constitués toutes les M références ;
- sur base du temps CPU : les échantillons sont constitués des références survenues pendant un intervalle de temps fixe t toutes les T unités de temps.

Les deux algorithmes ne différent que par la valeur à ajouter aux éléments (i,j) de la matrice :

(1) pour chaque échantillon, tout élément (i,j) de la matrice est incrémenté de 1 si i et j sont présents dans l'échantillon.

(2) pour chaque échantillon, tout élément (i,j) de la matrice est incrémenté de ni * nj où ni et nj représentent respectivement le nombre d'occurences de i et j dans l'échantillon.

Ces algorithmes tendent à favoriser le groupement de blocs apparaissant conjointement dans la chaîne de référence. Le deuxième tient compte dans le calcul, du nombre d'occurences conjointes.

5.3. COMPARAISON

La philosophie des algorithmes ayant été exprimée lors de leur énumération, nous nous contenterons d'une sommaire comparaison de structure et nous attacherons plutôt au point de vue de leurs exigences pour l'implémentation.

5.3.1. Structurelle

AB-A-B-CWS

Ils tiennent compte de

- la proximité des références dans le temps ;
- la fréquence d'apparition conjointe dans un intervalle de temps , qui est le paramètre de working set.

Basés sur la notion de working set, ils sont orientés dans le sens d'une stratégie de gestion mémoire de ce même type.

NEARNESS MATRIX

Il tient compte de la

- proximité des références ;
- fréquence d'apparition consécutive.

Il est très restrictif dans le calcul d'affinité (notamment lors d'appels en cascade : si on ne considère que les passages de contrôle, des séquences du type $A \rightarrow B \rightarrow C \rightarrow D \rightarrow C \rightarrow B \rightarrow A$ ne contribuent pas à rendre l'affinité entre les blocs A et C plus importante).

Il néglige le facteur temps.

Soient A ⟶ B ⟶ A ⟶ C ⟶ A

instant CPU de la transition	10	100	110	120
temps CPU passé dans le bloc	90	10	10	

Si nous supposons une stratégie de gestion mémoire du type WS (le plus courant) il a paraît plus intéressant de grouper A et B plutôt que A avec C, en regard du temps passé dans les blocs B et C.

RYDER (m,b)

Il tient compte de

- la proximité des références ;
- la fréquence d'apparition conjointe.

Il est d'autant moins restrictif dans le calcul d'affinité que la fenêtre d'observation est grande.

Il néglige le facteur temps.

MASUDA (1) et (2)

Ils tiennent compte de

- la proximité des références ;
- la fréquence d'apparition conjointe (et plus spécialement le second).

Ils négligent le facteur temps à l'intérieur des échantillons, ou même totalement si l'échantillonnage se fait sur base des références seules.

Un échantillonnage basé sur le temps permet de faire inter-

venir ce facteur dans le calcul d'affinité, puisque la fenêtre choisie définit le seuil à partir duquel on considère la "distance" (en terme de temps écoulé entre deux références) comme trop grande pour influer sur le calcul d'affinité.

5.3.2. TECHNIQUE

Notre objectif pour l'implémentation a été de minimiser le temps d'exécution nécessaire pour effectuer la restructuration, parfois au détriment du facteur espace mémoire. Les stacks ont donc fait place à des tables, même si celles-ci sont creuses.

5.3.2.1. Exigences en place mémoire

A. Nécessite la mémorisation d'un working set : w(tk,T) ;w(tk+1, T) doit cependant être construit alors que le premier existe toujours.
Taille = 2 * le nombre (maximum) de blocs.

B. Idem. L'incrémentation est simplement postposée à l'instant où le second est complet.

AB. Idem.

CWS. Nécessite 1 seul working set mis à jour à chaque nouvelle référence.
Taille = nombre maximum de blocs.

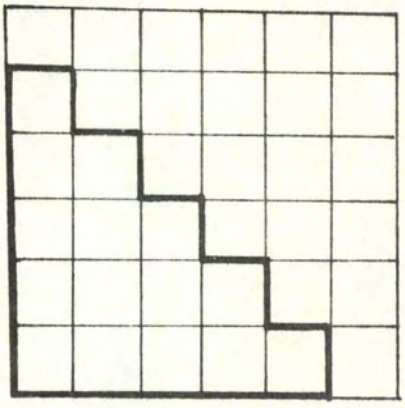
Nearness Matrix. Néant.

RYDER (m,b). Nécessite la mémorisation du vecteur d'incrémentation.

MASUDA (1) et (2). Mémorisation en table des blocs référencés.
Taille = nombre maximum de blocs.

Implémentation :

- toutes ces zones sont mises en "COMMON" ;
- la chaîne est traitée par blocs de 1024 * 2 ou 1024 entrées suivant le cas où l'algorithme utilise le temps des références ou non ;
- comme évoqué dans l'introduction, la matrice d'affinité construite est de la forme :



6 blocs

Les caractéristiques sont :

- taille = $\frac{n(n-1)}{2}$ si n est le nombre de blocs différents ;
- enregistrée ligne par ligne ;
- fonction d'accès : adresse de l'élément (i,j)

$= \frac{(i-1)\ (i-2)}{2} + j - 1 \qquad i > j$

La place mémoire est calculée, réservée et libérée dynamiquement. Les algorithmes de restructuration (= construction de la matrice) sont écrits en langage FORTRAN.

L'accès à la matrice se fait par

- une "fonction" de lecture ;
- 2 "subroutines" d'écriture et incrémentation.

La structure de la matrice est rendue invisible pour FORTRAN.

Les corrections d'adressage se font automatiquement.

5.3.2.2. Tableau général des exigences

Algorithme	Mémorisation de la chaîne	Présence du temps de la chaîne	Echantillonnage obligatoire	Echantillonnage permis
A		X		X
B		X		X
AB		X		X
CWS		X		X
Nearness Matr.				X
Ryder				X
Masuda 1			X	X
Masuda 2			X	X

5.3.2.3. Estimation du temps de calcul

Cette estimation n'est pratiquement possible que pour les algorithmes suivants :

Nearness Matrix : N-1

Ryder : (N-n) (n-1)

où N est la taille de la chaîne de référence (en nombre de références), et n la taille de la fenêtre d'observation.

L'échantillonnage permet une réduction du temps de calcul dans une proportion M/m (M et m représentant respectivement la fenêtre et la fréquence d'échantillonnage).

La façon dont est réalisé l'échantillonnage dépend du mesureur disponible pour espionner l'exécution du programme à restructurer. Il peut permettre d'échantillonner directement lors de la constitution de la chaîne. Si ce n'est pas le cas la chaîne sera enregistrée complètement, et amputée par la suite des références non désirées.

Notre façon de procéder correspond à la dernière citée.

CHAPITRE 6 : LES ALGORITHMES DE CONSTITUTION DE LISTES (CLUSTERING)

6.1. INTRODUCTION

Dans le chapitre précédent, nous avons vu comment constituer une matrice d'affinité entre blocs relocatables. Cette matrice va être exploitée par les algorithmes de "clustering" afin de constituer des listes de chargement. Le but est de grouper, sur une ou plusieurs pages, des blocs relocatables dont l'affinité estimée est maximale. Ils tiennent compte de la taille de la page et de la taille des blocs considérés, et tendent à minimiser le nombre de blocs traversant une frontière de page. Il subsistera cependant un certain nombre de blocs non groupés, soit parce que leur affinité avec les autres est insignifiante, soit à cause des contraintes mises sur la taille des blocs à grouper ensemble. Les blocs non casés seront alors placés en fonction d'une utilisation aussi dense que possible de la mémoire.

6.2. RELEVE

6.2.1. Algorithme de RYDER (25)

La matrice est d'abord simplifiée sur base des règles suivantes :

- annuler les affinités entre blocs de taille supérieure à une page (de façon à ce que les blocs soient alignés sur une frontière de page : lors de la rédaction du programme, éviter de placer une boucle de part et d'autre d'une frontière de page) ;
- annuler les affinités entre tout bloc de taille proche d'une page, et les autres blocs ;
- annuler les affinités entre tout couple de blocs tels qu'ils occupent autant de pages ensemble que s'ils étaient placés séparément. (ex. : 1bloc de 3/4 de page avec un bloc d'1/2 page)

L'algorithme demande aussi que les blocs ayant une affinité très faible soient placés en fonction d'une utilisation aussi dense que possible de la mémoire. Etant donné la complexité de définition du seuil de cette affinité, cette règle n'a pas été implémentée.

La matrice est ensuite exploitée comme suit :

- rechercher l'élément de valeur maximale de la matrice. Si la matrice est nulle, l'algorithme est terminé ;
- Sinon, les blocs correspondants sont
  - groupés en nouvelle liste ;
  - ou - ajoutés à une liste existante ;
  - ou - s'ils appartiennent déjà à une liste, les 2 listes correspondantes sont fusionnées.

Les affinités afférentes sont annulées. Les conditions suivantes doivent être respectées :

- jamais deux blocs plus grands que la taille de la page ne seront placés dans la même liste ;
- un bloc plus grand qu'une page doit être aligné sur frontière de page ;
- 2 blocs, ou 2 listes, ou une liste et un bloc sont groupés uniquement s'ils occupent moins de pages ensemble que s'ils étaient placés séparément.

6.2.2. Les algorithmes de MASUDA (22)

L'algorithme débute avec n essaims , constitués chacun d'un bloc relocatable. Il recherche l'élément (i,j) maximum de la matrice. S'il est nul, l'algorithme prend fin. Sinon, les essaims i et j sont groupés, sous contrainte qu'ils doivent occuper moins de pages que s'ils restaient séparés. La matrice d'affinité est alors mise à jour :

- les lignes et colonnes des 2 essaims associés sont groupées en une seule (i et j $\longrightarrow$ i), l'autre étant annulée ;
- les algorithmes diffèrent dans la façon de regrouper lignes et colonnes précitées (voir plus bas).

Il convient ensuite de rechercher le nouveau maximum ...

Nous supposons pour ce qui suit que :

n = nombre d'essaims au départ

Mpq = l'élément (p,q) de la matrice d'affinité

Technique NN (Nearnest Neighbour)

$\forall$ k = 1 ... n k $\neq$ i et k $\neq$ j : Mi,k = max (Mik,Mjk)

(Mk,i = max (Mki,Mkj))

Technique FN (Furthest Neighbour)

$\forall$ k = 1 ... n k ≠ i et k ≠ j : Mik = min (Mik,Mkj)

(Mki = min (Mki,Mkj))

Technique AV (Average)

$\forall$ k = 1 ... n k ≠ i et k ≠ j : $Mik = \frac{1}{nik} \sum_p \sum_q Mpq$

p & q représentent les blocs appartenant respectivement aux essaims (actuels) i et k, l'essaim i étant le groupement des essaims i et j de l'étape précédente.

Les Mpq sont ceux de la matrice initiale.

nik = $\#\{Mpq \neq 0\}$ = nombre de Mpq non nuls

Technique MA (Modified Average)

$\forall$ k = 1 ... n k ≠ i et k ≠ j : $Mik = \frac{1}{ti+tk} \sum_p \sum_q Mpq$

Cette technique fait intervenir la taille des blocs et donc des essaims lors de la mise à jour de la matrice. La valeur (moyenne) calculée pour déterminer l'affinité du nouvel essaim avec les autres est cette fois fonction de leurs tailles respectives, ti et tk représentant la taille des essaims i et k respectivement soient la somme des tailles des blocs qui les composent.

6.2.3.

Il existe encore d'autres algorithmes dûs principalement à Hatfield et Gerald, et à Ferrari. (17,13)
La complexité de leur implémentation et des techniques utilisées nous ont poussé à ne pas les prendre en considération. De plus, une étude comparativeréalisée par Masuda ... (22) tend à montrer que les résultats fournis par ces algorithmes ne justifient pas leur complexité.

6.3. REMARQUES

- les algorithmes de constitution de listes étudiés ne sont pas biaisés dans le sens d'une stratégie de gestion mémoire particulière.
  Ils posent des contraintes quant à la taille des listes constituées, par rapport à la taille de la page (Ryder étant d'ailleurs plus restrictif à ce sujet) ;

- il est à noter que,lorsque l'algorithme touche à sa fin, un certain nombre de blocs n'ont pu être ratachés aux listes constituées. De plus, les listes ne remplissent pas complètement les pages qui leur sont allouées. Une procédure est employée afin d'arriver à une utilisation aussi dense que possible de la mémoire, ceci afin de n'utiliser qu'un minimum de pages supplémentaires pour caser les blocs non

encore groupés.

Une stratégie MAX-MIN essaye de placer successivement les blocs les plus grands dans les "trous" les plus petits.

CHAPITRE 7 : LE PROBLEME DE LA DENSIFICATION

7.1. CADRE

Les blocs relocatables se trouvent, à ce moment, rassemblés dans des listes de chargement. A l'étape précédente, les blocs non groupés ont été placés de façonà obtenir une utilisation aussi dense que possible de la mémoire. Les listes constituées occupent un nombre entier de pages, ce qui entraîne encore un espace inutilisé en fin de chaque liste. Le problème de la densification est double :

1) Quel est l'intérêt de tasser les listes obtenues, afin de supprimer le gaspillage de place, et par le fait même,d'éviter le chargement en mémoire d'espaces inutilisés ? (Soient les espaces restés vierges dans la dernière page de chaque liste). Pour le même nombre d'instructions exécutées ; il faudrait en moyenne plus de place mémoire centrale. (17) Ceci aura cependant pour conséquence de modifier les alignements opérés sur les blocs par les algorithmes du chapitre précédent.

2) Comment définir l'ordre dans lequel les listes vont être placées à la suite une de l'autre ? Le but est ici, non pas de grouper les listes, ce qui n'a pas d'intérêt au point de vue pagination, mais bien de les séquencer. Ce problème n'existe évidemment que

dans le cas où on décide de "tasser". Pour s'assurer que les 2 pages qui contiennent la routine "à cheval" soient en mémoire en même temps, on concatène les listes ayant l'affinité la plus grande.

Nous avons donc considéré trois possibilités :

1er cas : On ne densifie pas, de façon à respecter strictement les indications fournies par les algorithmes de constitution de listes. Il faudra spécifier au LINKAGE EDITOR les adresses de chargement correspondant à chaque liste, ou si ce n'est pas possible, générer des routines "bidon" de tailles équivalentes aux espaces à laisser inoccupés.

2e cas : On tasse simplement les listes dans un ordre arbitraire. Ceci ne demande l'application d'aucun algorithme et permet de récupérer les "trous" précités.

3e cas : On séquence les listes avant d'opérer la densification. Le problème, semblable à celui du voyageur de commerce, est généralement résolu par PSEP (Branch and Bound) ; voir (15) Les listes sont considérées comme étant les sommets d'un graphe non orienté. Les poids des arêtes sont définis comme suit :

$aij = \sum_{\substack{p \in i \\ q \in j}} Cpq$, l'affinité entre deux listes i et j étant la somme des affinités entre les blocs qui les composent.

La matrice d'affinité, détruite par les algorithmes de l'étape précédente, devra donc être sauvée pour le calcul de l'affinité entre listes.

Vu la complexité de PSEP, tant au point de vue implémentation que coût d'utilisation (il existe (n - 1) ! circuits hamiltoniens dans un graphe complet et symétrique d'ordre n) nous avons imaginé un algorithme simplifié.

7.2. ALGORITHME PROPOSE

Notre but est de construire un vecteur C ordonné de listes de chargement.

Etape 1 Calculer la matrice d'affinité associée aux listes à partir de la matrice d'affinité entre les blocs.

$$M(j,i) = \sum_{\substack{p \in i \\ q \in j}} aff(p,q) \quad \forall\, p,q = 1 \ldots n$$; n = nombre de blocs du programme.

Le vecteur C de départ est vide.

Etape 2 Rechercher l'élément maximum de la matrice ; soient i et j ses coordonnées.

Annuler M (i,j) .

Introduire i et j dans C.

Etape 3 Rechercher l'élément maximum dans chaque zone (ligne et colonne) associée aux éléments extrêmes de C.

Soient Maxi et Maxj de coordonnées (k,i) et (j,l)

Si MAXi = MAXj = 0, aller à l'étape 4.

Si MAXi $\geqslant$ MAXj annuler les éléments M(k,z) $\forall$ z $\in$ C
introduire k avant i dans C
aller à l'étape 3.

Si MAXi $<$ MAXj annuler les éléments M(l,z) $\forall$ z $\in$ C
introduire l après j dans C
aller à l'étape 3.

Etape 4 Introduire dans C, à la suite une de l'autre, toutes les listes non groupées.

Remarques

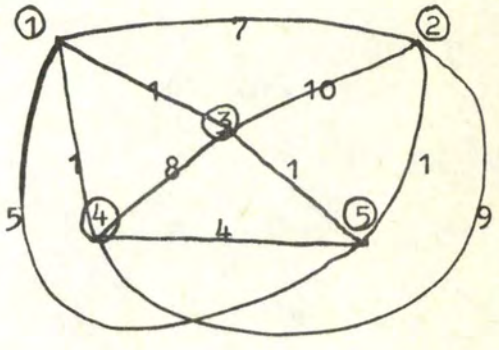
1) Comme nous l'avons fait pour la matrice d'affinité entre blocs, nous ne construirons qu'une demi matrice d'affinité entre listes.

Les principes d'adressage - ainsi que les routines associées - restent les mêmes. Ceci explique que, dans la description de l'algorithme précédent, les notations (r,s) et (s,r) représentent le même élément.

2) Le schéma proposé n'est pas optimal.

Pour s'en convaincre, il suffit d'examiner l'exemple suivant. Soient la matrice d'affinité entre listes et le graphe correspondant

	1	2	3	4	5
1					
2	7				
3	1	10			
4	1	9	8		
5	5	1	1	4	



On trouverait C = (15423), soit un chemin de poids

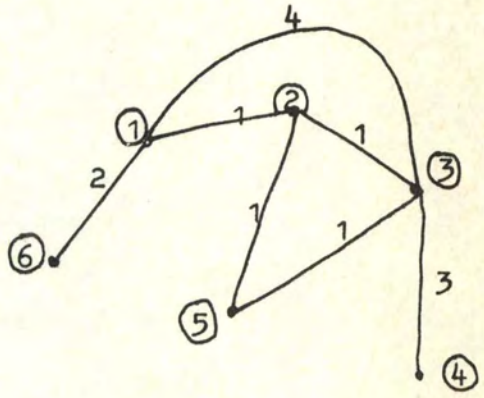
5 + 4 + 9 + 10 = 28

or il existe le chemin 12345 de poids 29.

3) Le schéma proposé ne fournit pas toujours un chemin hamiltonien.

Considérons

	1	2	3	4	5	6
1						
2	1					
3	4	1				
4	0	0	3			
5	0	1	1	0		
6	2	0	0	0	0	



Le résultat obtenu est C = (6134)

or, il existe 612534.

CHAPITRE 8 : METHODOLOGIE

Nous avons déjà émis l'opinion que le niveau d'amélioration à obtenir n'est pas constant pour tous les programmes à restructurer. Nous désirons de ce fait réaliser la correspondance entre coût et efficacité.

La méthode suivie pour comparer des procédures de restructuration ("restructuring" + "clustering" + densification) est exposée.
Elle consiste en un essai des combinaisons possibles des différents algorithmes sur des programmes "types". Du point de vue pratique, l'intérêt de comparer des algorithmes de "restructuring" entre eux, ou des algorithmes de "clustering" entre-eux, est limité. Les deux étapes étant obligatoires pour effectuer une restructuration, ce sont les résultats de leurs combinaisons qui importent. Ce serait cependant intéressant si on désirait, par exemple, développer de nouveaux algorithmes.

Avant d'exposer la méthode suivie, il nous paraît utile d'établir une classification des algorithmes implémentés et des paramètres associés.

8.1. Discussion des paramètres

Les paramètres de la restructuration sont en rapport direct avec le programme ou le système, certains étant cependant très difficilesà situer et intervenant aux deux niveaux.
Le même paramètre peut avoir une influence, une "valeur", différente suivant l'algorithme concerné. Le relevé des principaux paramètres intervenant dans les différents algorithmes implémentés est accompagné d'une discussion de leur signification.

8.1.1. La fenêtre du working set (CWS, AB, A, B)

Les algorithmes fournissent une matrice d'affinité basée sur la chaîne de référence extraite du programme lors de son exécution et ce, pour la fenêtre spécifiée. Les blocs référencés conjointement dans un intervalle de temps correspondant sont déterminés. L'affinité est pondérée par la fréquence d'apparition simultannée et servira à grouper les blocs dans des pages. Une fenêtre inadéquate conduit à une mauvaise approximation de la localité : trop petite, elle contribue à une sous-estimation de l'espace mémoire réellement nécessaire au programme, trop grande à une sur-estimation.

Si la stratégie de gestion mémoire est du type working set, la fenêtre correspond au temps maximum de maintien d'une page en mémoire principale. Il apparaît idéal de faire correspondre la fenêtre utilisée lors de la restructuration avec la précédente citée, de façon à ce que l'arrangement obtenu épouse au mieux les objectifs de l'algorithme de pagination. Malheureusement, celui-ci est rarement "working set" pur, et la fenêtre associée, difficile à déterminer avec précision.
Il peut même être d'un type totalement différent. Rien pourtant ne permet de dire que les algorithmes CWS, AB, A et B sont inutilisables, ni même à délaisser. Sur quelles bases définir le choix de la fenêtre à utiliser dans ce cas ?
Nous avons déjà fait ressortir qu'un programme évolue d'une localité vers une autre. La fenêtre devrait être choisie en vertu des principes suivants :

- elle est grande assez pour que la probabilité qu'une page appartenant à la localité courante ne soit pas dans le working set, soit petite ;
- elle est petite assez pour que, lors du passage d'une localité à une autre, la probabilité d'avoir plus d'une page de la nouvelle dans le working set, soit petite (C6).

8.1.2. Fréquence et fenêtre d'échantillonnage (algorithmes de"restructuring")

Ils correspondent à une facilité que l'on s'autorise à prendre et sont soumis à des contraintes provenant de sources diverses :

- des algorithmes utilisant une fenêtre (CWS, AB, A, B) de façon à ce que les tailles des deux fenêtres soient compatibles. Pour AB, par exemple, la fenêtre d'échantillonnage doit au moins être le double de la fenêtre choisie pour l'algorithme ;
- du temps d'exécution du programme à restructurer de façon à ce que le nombre d'échantillons extraits soit suffisant ;
- de la densité des informations enregistrées par le mesureur lors de la création de la chaîne, de façon à ce que les échantillons soient suffisamment garnis, et surtout non vides.

Des précautions doivent être prises de façon à détecter et supprimer les échantillons inutilisables pour le calcul d'affinité et à avertir l'utilisateur. Notons cependant que, pour les algorithmes de Masuda, la fenêtre prend une signification comparable à celle de la fenêtre utilisée par les algorithmes orientés working set (CWS, AB, A, B).

8.1.3. Vecteur d'incrémentation (Ryder M,B)

Il définit la force des connexions entre blocs en fonction de leur distance dans la chaîne de références. La taille de ce vecteur n'est pas une fenêtre à proprement parler. Le facteur temps n'intervient pas. Le coût d'utilisation devient exhorbitant pour des valeurs de M supérieures à 10. Les liens existant entre les valeurs "acceptables" de ce paramètre et une éventuelle fenêtre de temps au niveau de la stratégie de gestion de la mémoire sont quasi nuls. La détermination de la valeur à choisir pose de ce fait peu de problèmes à ce niveau et peut être fixée dans l'absolu. (5.2.6.)

8.1.4. Taille de la page (algorithmes de "clustering")

Elle est fixe et connue pour un système donné et ne présente pas matière à discussion.

8.2. Classification des algorithmes

"Restructuring"

- présence du temps : CWS AB A B

  paramètres relatifs au système : fenêtre du working set

  au programme : fenêtre, fréquence d'échantillonnage

- échantillonnage "obligatoire" : MASUDA (2 algorithmes)

  paramètres relatifs au système : fenêtre d'échantillonnage

  au programme : fenêtre, fréquence d'échantillonnage

- autres : Nearness Matrix ; Ryder (M,B)

  paramètres relatifs au programme : fenêtre, fréquence d'échantillonnage

  "propres" : taille et contenu du vecteur d'incrémentation.

"Clustering"

- très restrictif sur taille des essaims (listes) RYDER

  paramètre relatif au système : taille de la page

- peu restrictifs : MASUDA NN,FN, AV, MA

  paramètre relatif au système : taille de la page

8.3. Méthode suivie

Nous désirons répondre à la double question de

- déterminer comment effectuer la comparaison de procédures de restructuration dans le cadre d'unsystème donné ;
- définir les coût et efficacité de chacune par l'utilisation de l'outil développé.

Les résultats obtenus seront donnés àtitre indicatif, non à titre définitif, en raison des différences qui existent entre les systèmes. (taille de la page, fenêtre du working set à choisir ...).

Nous pensons par contre que la méthode suivie est générale et applicable quel que soit le système considéré.

L'idée directrice de la méthode exposée a été exprimée par Hatfield et Gerald : la comparaison doit se faire dans des conditions normales d'exécution du programme observé, et pas aux extrêmes (17).

Etape préliminaire :

Fixation des paramètres "propres" aux algorithmes.
Cette étape relève de la comparaison de l'algorithme avec lui-même. Afin de ne pas multiplier les essais effectués à l'étape 3, il convient au départ de déterminer dans quelles conditions les algorithmes implémentés ont un rendement maximum au niveau de leur fonctionnement interne. Ce problème est généralement résolu dans la littérature, notemment par les auteurs qui ont développé ces algorithmes. Citons comme exemple la dimension et le contenu du vecteur d'incrémentation associé à l'algorithme de Ryder (5.2.6.).

Etape 1 : Choix des conditions de départ

Cette étape concerne la fixation des paramètres ayant un rapport direct avec le système considéré.
La taille de la page ne pose pas de difficultés, puisqu'elle est généralement fixe et connue pour un système donné.
Le problème majeur concerne le choix de la fenêtre du working set à utiliser dans le cas où aucune indication ne permet de la déduire du système sur lequel on travaille. Aucune méthode générale n'existe pour la définir de façon précise. Les auteurs s'avancent très peu dans ce domaine et se contentent de simples conseils. (cfr 8.1.1.)

Pour le modèle LRU, où la taille de mémoire centrale allouée au programme au début de son exécution réelle est connue, ou résulte d'une fonction connue, nous proposons la méthode suivante utilisant le simulateur working set.
Comme montré par Coffman et Dennin (06), la probabilité de défaut de page du modèle working set, où le working set moyen est de taille p, est un bon estimateur de la probabilité de défaut de page du modèle LRU allouant la même quantité de mémoire p. p étant connu, il suffit de paramètrer le simulateur WS.
Dès que le working set moyen devient égal à p, la fenêtre correspondante est relevée.

Etape 2 : Choix d'un cobaye

Le processus suivi pour la comparaison consiste en un essai des différentes combinaisons possibles des algorithmes sur un certain nombre de programmes représentatifs de la charge et des utilisateurs.
En plus des considérations de coût, la sélection de ces programmes est basée sur les éléments suivants :

- taille ;
- consommation CPU : programme de gestion ou de calcul ;
- type : conversationnel ou "batch" ;
- langages de programmation : langage machine ou évolué.

La stabilité des comportements vis à vis des données d'entrée devrait idéalement être vérifiée, les données choisies pour l'exécution des programmes sélectés étant représentatives des exécutions normales. La difficulté de tenir compte de tous les facteurs cités résulte des coûts engendrés. L'investissement consenti sera en rapport avec le degré de sécurité voulu. L'hypothèse que les résultats obtenus sur un nombre restreint de programmes sont valables pour tous, est faite par tous les auteurs cités.
Une mesure sur le système réel, bien que coûteuse, serait d'une grande utilité pour vérifier la correspondance des résultats avec la réalité.

Etape 3 : Essais des différentes procédures de restructuration

Les principaux aspects à examiner sont les suivants :

A. Détermination de l'influence de l'échantillonnage

Afin de réduire le coût de chacune des restructurations à réaliser dans le futur, il convient de déterminer :

- les conditions dans lesquelles l'échantillonnage est possible (Fréquence, fenêtre) sans oublier les contraintes exprimées dans l'introduction du chapitre ;
- les procédures où il est applicable et les dégradations de performances / amélioration de coût imputables.
- les procédures où il est à proscrire.

B. Détermination de l'utilité de la densification

La densification ne constitue pas une étape obligatoire dans la procédure de restructuration. Son coût est par ailleurs modique. L'expérience montre que son utilité dépend essentiellement du programme considéré. Les conditions à réunir pour que son utilisation soit bénéfique seront précisées.

C. Détermination des procédures à proscrire

A première vue, aucun élément ne permet de déduire des contradictions entre les algorithmes des différents types. Il se peut cependant que des incompatibilités existent. Les associations à proscrire seront relevées.

D. Déduction d'un tableau comparatif coût - performance

Sur base des essais effectués sur les différents programmes choisis à l'étape 2, la comparaison des procédures utilisables s'effectue à deux niveaux :

1) Niveau des coûts :

La difficulté est de définir une référentiel de base pour la comparaison. Les éléments entrant en ligne de compte dans le coût d'un algorithme sont nombreux. Pour un algorithme donné, chacun des paramètres influe de façon variable sur le coût d'utilisation.

Il est pour ainsi dire impossible de prévoir avec précision le coût de la restructuration d'un programme à l'aide d'une procédure donnée.
Le même algorithme aura un coût d'utilisation différent d'un système à l'autre (taille de la page) et sur le même système, d'un programme à l'autre (nombre de blocs relocatables, longueur de la chaine de référence).
Comme référence de base, le plus simple est de choisir le coût de la procédure la moins consommatrice.

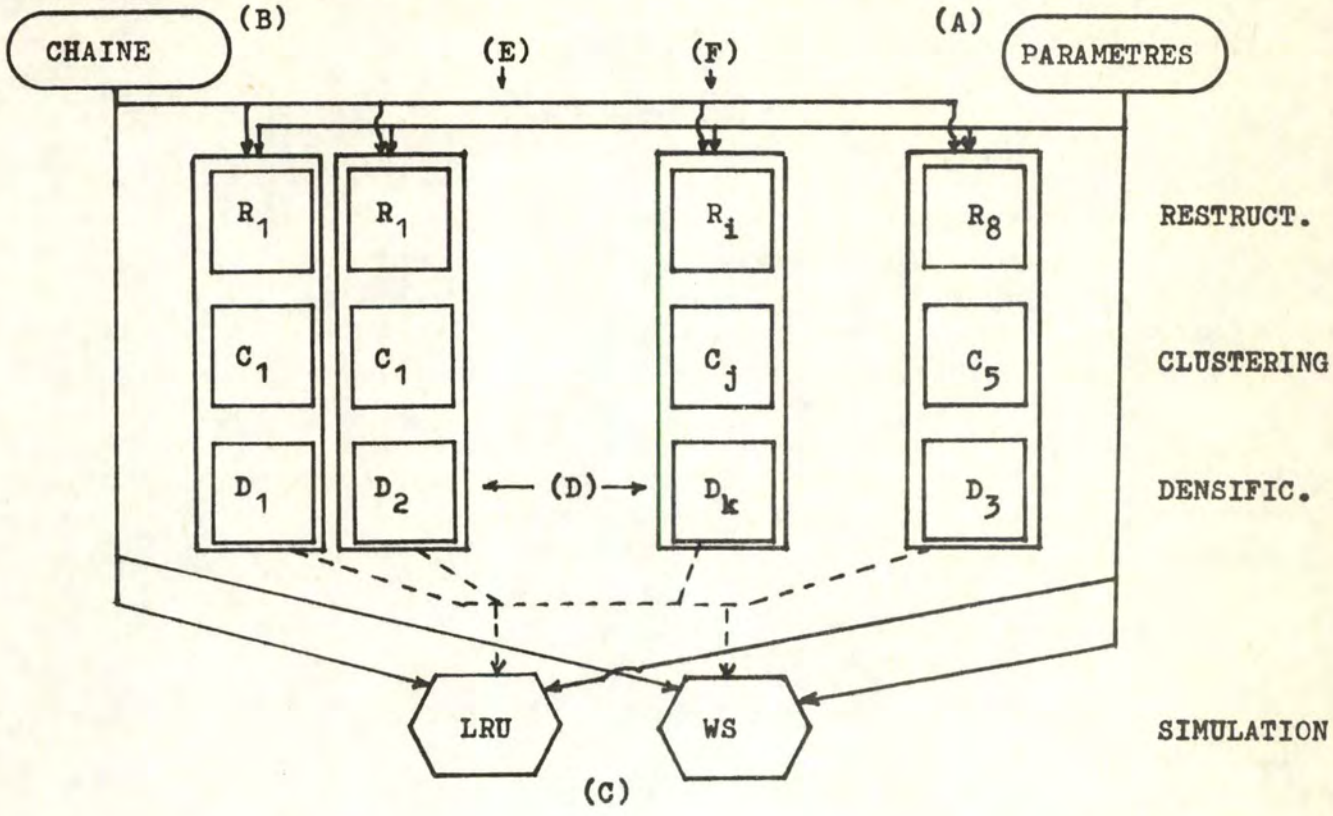
2) Niveau des performances :

Les améliorations seront exprimées par rapport au comportement du programme original (en nombre de défauts de page).

Etape 4 : Décision

Il reste à vérifier la cohérence des résultats obtenus lors des essais réalisés. Les divergences observées permettent d'évaluer le degré de sensibilité des procédures vis à vis des caractéristiques propres des différents programmes, et par le fait même, le degré de certitude quant à leur utilisation globale pour tous les programmes à restructurer. En fonction du degré de certitude voulu, la décision est prise de stopper ou de poursuivre des essais, auquel cas, un nouveau "cobaye" est sélecté.

Le schéma ci-après fait ressortir la complexité du problème, pour les algorithmes implémentés.



Le but avoué est, dans un premier temps, de simplifier progressivement le problème afin de satisfaire à l'impératif de réduction des coûts. La démarche suivie vise à

- fixer le plus grand nombre possible de paramètres (A)
- définir où l'échantillonnage est possible (B)
- définir le simulateur à utiliser (C)
- séparer la densification de la procédure de restructuration (D)
- éliminer les procédures où des incompatibilités existent (E)
- trouver les procédures moins performantes et plus coûteuses que d'autres (F)

CHAPITRE 9 : TESTS ET RESULTATS

9.1. CADRE DES ESSAIS

9.1.1. Difficultés

Pour que les résultats obtenus en suivant la méthode proposée soient valable, il est impératif de réaliser les essais sur un nombre suffisant de programmes "types". Deux obstacles majeurs surgissent lors du passage à la pratique :

- la difficulté d'obtention de "cobayes" utilisables et d'instrumentation de ces programmes. Les outils utilisés pour ce faire sont au stade expérimental et les lacunes qui en découlent sont encore nombreuses. De plus, les compilateurs (CO BOL) ne génèrent pas des objets rigoureusement respectueux des conventions standard . Par contre, les coûts d'instrumentation sont négligeables ;
- les coûts observés lors de l'utilisation de l'outil de restructuration et des simulateurs sont très élevés. Ceci a constitué un obstacle majeur quant au nombre des essais effectués, étant donné le budget dont nous avons disposé. Quelques chiffres sont donnés dans la suite, à titre indicatif. Ils montrent que la consommation de ressource temps CPU devient rapidement exhorbitante.

Force est donc de reconnaître que l'intérêt de l'outil développé est malheureusement limité par des considérations matérielles.

9.1.2. Limitations

L'analyse technique des sorties fournies par les simulateurs est inspirée des recommandations exprimées par Tsao, Comeau et Margolin. Désirant déterminer l'influence de 4 facteurs sur le processus de pagination, ils ont consenti un investissement de 60 heures CPU (26).

Notre problème, bien que différent, possède une dimension comparable. Dans notre cas, l'étude complète et rigoureuse des 120 combinaisons (8x5x3) des algorithmes implémentés - chacun étant de plus paramétrable - est impossible.

Nous avons été contraints de ne considérer qu'un seul programme, choisi relativement important. Cette option de préférer un seul "gros" programme à plusieurs petits est justifiée dans le fait que la restructuration, par essence même, s'applique à des programmes importants. Le choix inverse faisait courir le risque de fausser les résultats vu le petit nombre de routines impliquées et de minimiser l'intérêt de la restructuration.

Au niveau des coûts, une appréciation erronée aurait été probable.

9.1.3. Détermination des paramètres

Pour l'algorithme de Ryder (M,b), la taille du vecteur a été fixée à 6, le contenu étant (5,4,3,2,1), valeurs qui correspondent aux recommandations de (25) et(16). La taille de la page est de 4096 bytes (4K) sauf indications contraires. La fenêtre choisie (100 ms) correspond à des conditions "normales".
Des essais réalisés avec des valeurs variant entre 10 et 500 ms sur le simulateur WS portent à croire que la fenêtre idéale est comprise entre 50 et 200 ms.
Afin de conserver la correspondance avec les travaux des auteurs consultés, la valeur 100 a été sélectionnée.

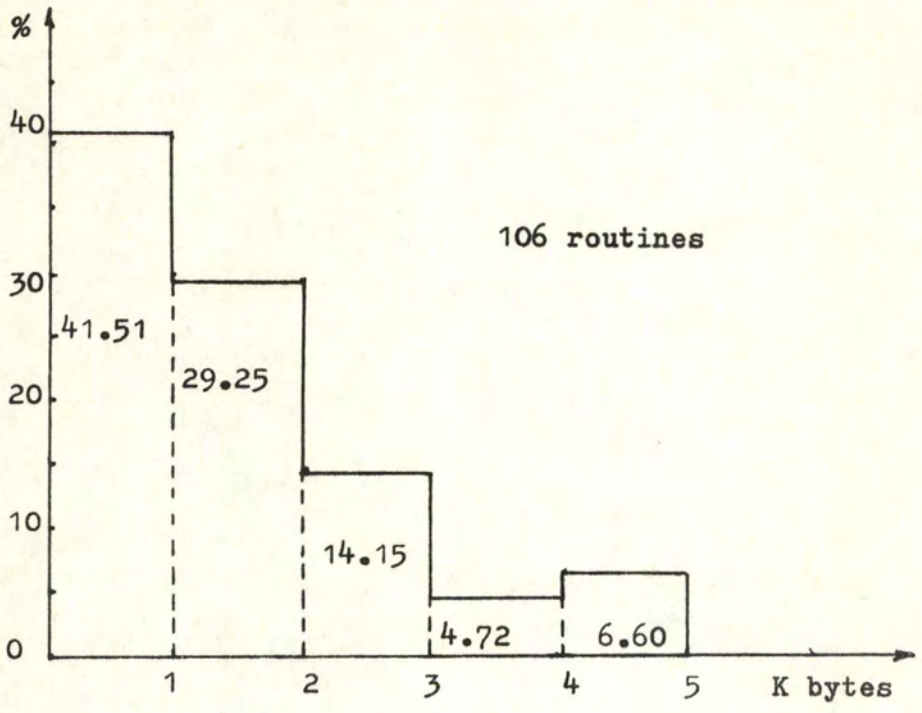
9.1.4. Programme testé

FUNTSP est un programme d'analyse statistique écrit en langage FORTRAN.
Les données d'entrée sont représentatives de son utilisation normale, bien que relativement réduites de façon à ne pas générer une chaîne de référence trop importante et de ce fait, à réduire le coût des essais, notamment au niveau des algorithmes de "restructuring" et des simulateurs.

Ses caractéristiques sont les suivantes :

- taille :
  - totale : 508 K bytes
  - des routines programmées (par opposition à celles incluses par le Linkage Editor et provenant de la librairie FORTRAN) 207 K
  - des zones "COMMON" : 254 K
- nombre total de routines : 182, soient 106 "utilisateur" et 76 autres (non suivies)
- nombre de routines effectivement référencées lors de l'exécution : 66 (sur 106)
- temps CPU consommé lors de l'extraction de la chaîne : 21 secondes
- nombre de références enregistrées dans la chaîne : 11860
- nombre moyen de références dans une fenêtre de 100 ms : 56.

Distribution de la taille des blocs relocatables de FUNTSP.



A ceci s'ajoutent 3 blocs de grande taille

(respectivement 8, 15 et 16 K)

Nombre de routines < 1/3 de page : 55 (page de 4 K)

Nombre de routines > page : 11 "

9.1.5. Quelques chiffres indicatifs des coûts :

Le nombre de combinaisons possibles des algorithmes est de 120 (8x5x3). Un essai de chacune des procédures a été réalisé, puis les nouveaux arrangements des blocs, simulés par working set et LRU. La consommation totale en CPU se répartit comme suit :

- pour la restructuration : 2800 sec
- simulation WS : 3000 sec (arrêtée après simulation de 60 arrangements)
- simulation LRU : 1150 sec (complète)

9.1.6. Coût des simulateurs

Influence des principaux facteurs sur le coût de simulation :

A) LRU :	nombre de pages allouées	nulle
	taille de la page	nulle
	nombre de blocs relocatables	non identifiée
B) WS :	Fenêtre du working set	nulle
	taille de la page	inversément proportionnelle
	nombre de blocs relocatables	non identifiée

Dans les deux cas, le coût de simulation est proportionnel à la longueur de la chaîne enregistrée.

A titre indicatif, voici les coûts de simulation d'un arrangement pour une chaîne de longueur 10 000 avec une page de 4 K (106 blocs)

WS	LRU
40 sec	8 sec

Un facteur 5 apparaît entre les coûts des deux méthodes de simulation.

9.2. ESSAIS REALISES

9.2.1. Coût des algorithmes

Le coût d'une procédure de restructuration est constitué des composantes suivantes :

- coût de restructuring ;
- coût de clustering ;
- coût de densification.

Pour prévoir le coût d'utilisation d'un algorithme, il faut

- identifier les facteurs susceptibles d'influer (examen de l'implémentation réalisée) ;
- déterminer les facteurs dont l'influence est la plus marquée ;
- en déduire une fonction de coût.

Etude du cas CWS

Les facteurs susceptibles d'influer sont :

- la longueur de la chaîne exprimée en nombre de références (plutôt qu'en temps d'exécution ce qui n'est guère significatif) ;
- le nombre de blocs relocatables référencés ;
- la taille de la fenêtre du working set.

Des variations sur la longueur de la chaîne et le nombre de blocs référencés ont été obtenuespar échantillonnage. Un essai d'ajustement par moindres carrés (basé sur 30 observations) s'est révélé inefficace. Le seul résultat tangible est le coefficient de corrélation entre le coût et la longueur de la chaîne (0,92).
Cette expérience montre la difficulté de prévision du coût des algorithmes.

9.2.2 Influence de l'échantillonnage.

Conditions: fréquence d'échantillonnage 1 000 ms (F)

fenêtre d'échantillonnage 500 ms (f)

fenêtre du working set 100 ms (w)

Les tableaux suivants donnent la dégradation observée par rapport aux résultats de l'arrangement effectué sans échantillonnage dans les mêmes conditions, ainsi que les diminutions de coûts obtenues. Aucune densification n'a été réalisée. Rappelons que les critères concernés sont respectivement le nombre de défauts de page et le temps CPU consommé.

Dégradation des performances (%)

R. \ C.	Ryder	NN	FN	AV	MA
CWS	17.14	17.04	30.78	13.96	16.37
A	24.30	25.12	32.45	17.53	26.41
B	27.81	32.70	47.30	29.64	31.45
AB	19.72	26.46	25.76	22.38	20.97

Diminution des coûts totaux (Res.+Clus.)(%)

R. \ C.	Ryder	NN	FN	AV	MA
CWS	43.27	55.10	53.95	53.04	48.13
A	39.21	32.72	32.40	28.83	23.44
B	30.58	32.06	31.53	33.65	21.99
AB	34.94	37.75	35.95	33.28	34.64

Diminution du coût de restructur.(%)

CWS	A	B	AB
70.70	61.33	57.51	65.30

Diminution du coût de clustering (%)

C. / R.	Ryder	NN	FN	AV	MA
CWS	33.69	16.40	11.72	17.36	12.52
A	37.86	20.41	20.43	18.09	13.07
B	28.79	20.41	20.35	25.55	11.92
AB	32.31	21.24	18.88	17.39	23.35
Moyenne	33.16	19.61	17.84	19.59	15.21

Les algorithmes "sophistiqués" sont moins sensibles à l'échantillonnage au niveau des performances.

Des essais supplémentaires réalisés sur la combinaison CWS-MA en faisant varier F, f et w, il ressort que :

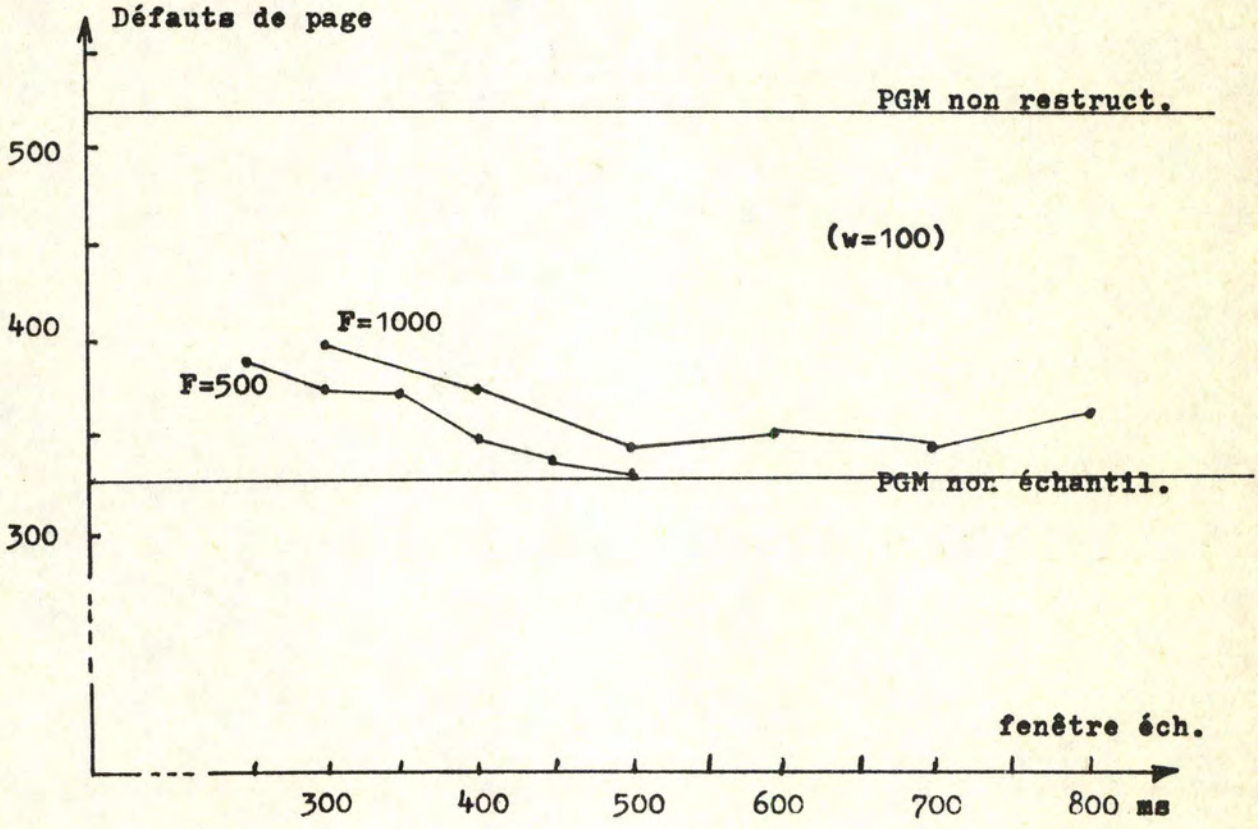
$$C(f/F) < C(\frac{kf}{kF}) \quad \text{pour } k > 1$$

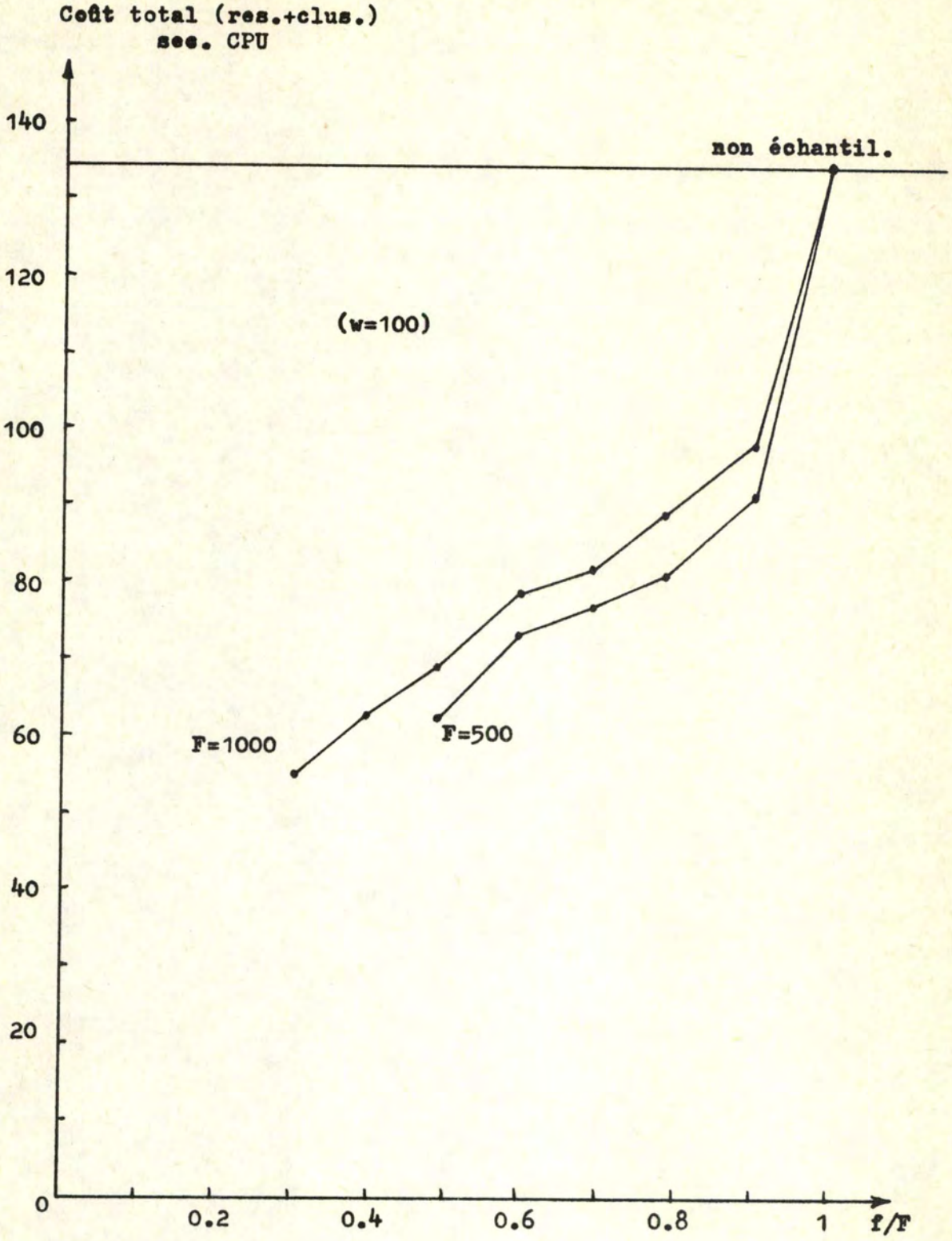
et que $P(f/F) < P(\frac{kf}{kF})$ $(k > 1)$ sauf si $F-f < a*w$ avec $a \simeq 1$

où C et P représentent respectivement le coût de la restructuration et les performances enregistrées (par rapport au programme original).

Le rapport $\frac{\text{Performance}}{\text{Coût}}$ apparaît alors comme étant maximisé pour F=1 000 et f=500.

Les graphiques suivants illustrent ces constatations. Seules ont été reprises les observations relatives aux fréquences 1 000 et 500.




Coût total (res.+clus.)
sec. CPU
140
120
100
80
60
40
20
0
non échantil.
(w=100)
F=1000
F=500
0.2
0.4
0.6
0.8
1
f/F

9.2.3. Utilité de la densification

L'utilisation d'une méthode de densification dans le cas du programme étudié n'a été favorable pour aucun des algorithmes de "clustering". Nouns ne pouvons cependant pas généraliser cette constatation. La densification a le désavantage de perturber l'arrangement effectué. Elle devrait le compenser par l'économie de place qu'elle réalise. Pour FUNTSP, l'explication de son inefficacité est basée sur deux constatations :

- la taille du working set moyen est faible : pour une fenêtre de 500 ms, 13 pages de 4 K pour le programme original (207 K), tombant à 10 pages après restructuration.
- le nombre de routines effectivement référencées est relativement bas (62%). Ces blocs, n'intervenant pas dans le calcul d'affinité, sont placés de façon à réaliser une occupation aussi dense que possible de la mémoire. La place gaspillée est donc très peu importante (4800 bytes, soient 2,26%). Ne permettant pas une économie de place suffisante mais contrariant cependant l'arrangement effectué par les algorithmes de "clustering", la densification s'avère nuisible dans ce cas précis.

Les essais effectués permettent cependant de tirer certains enseignements relatifs à la méthode de densification par tassement pur et simple et à la méthode utilisant la matrice d'affinité. Sur l'ensemble des essais réalisés cette dernière s'est montrée supérieure :

Algorithme	Amélioration moyenne
RYDER	3,83 %
NN	2,91 %
FN	4,79 %
AV	0,77 %
MA	9,87 %

Etant donné le coût insignifiant de l'algorithme proposé, ces constatations sont encourageantes.
Ryder étant apparemment le plus restrictif, il était surprenant de voir que l'amélioration reste aussi basse. Des essais furent réalisés en ramenant la taille de la page à 2K, l'amélioration passant dans ce cas à 10,89%.

9.2.4. Incompatibilités

Par incompatibilité, nous entendons le fait que des algorithmes associés conduisent à un arrangement des blocs manifestement mauvais, résultant par exemple, d'objectifs contradictoires, et se traduisant dans une dégradation des performances, par rapport au comportement du programme original. Nous n'en avons relevé aucune entre les algorithmes de "restructuring" et de "clustering" implémentés.

A noter cependant que les combinaisons Nearness-NN, Nearness-FN et Nearness-AV sont redondantes puisque, pour la même chaîne et le même jeu de paramètres, elles conduisent toujours à des résultats identiques. AV étant le plus coûteux, il est inutile de l'utiliser.

9.2.5. Comparaisons. (Programme non densifié, non échantillonné)
Les chiffres donnés doivent être pris comme des indications, non comme des certitudes, vu le nombre restreint d'essais réalisés.

Améliorations p. r. au programme original (% du nb. de déf. de p.)

R. \ C.	RYDER	NN	FN	AV	MA
CWS	36.51	31.65	30.87	31.06	37.08
A	28.34	26.21	24.07	26.60	19.02
B	31.26	32.03	22.71	32.42	26.60
AB	29.12	28.15	24.66	33.00	33.39
Nearness	16.50	14.56	14.56	14.56	15.72
Ryd.(m,b)	21.94	20.38	11.84	20.77	22.91
M1	24.85	19.41	21.55	21.94	21.55
M2	18.05	12.81	9.90	9.90	19.22

Coûts totaux enregistrés (res.+clus.) (sec. CPU)

R. \ C.	RYDER	NN	FN	AV	MA
CWS	318	116	115	125	135
A	234	45	46	54	63
B	227	45	47	56	64
AB	246	52	55	59	73
Nearness	85	37	37	43	56
Ryd(m,b)	152	47	50	56	69
M1	136	32	32	39	48
M2	149	33	33	37	47

N.B. Pour M1 et M2, les fréquence et fenêtre d'échantillonnage sont respectivement 500 et 100 .

Rapport performance / coût

R. \ C.	RYDER	NN	FN	AV	MA
CWS	0.11	0.27	0.26	0.25	0.27
A	0.12	0.58	0.52	0.49	0.30
B	0.13	0.71	0.48	0.57	0.41
AB	0.11	0.54	0.46	0.55	0.45
Nearness	0.19	0.39	0.39	0.33	0.28
Ryd(m,b)	0.14	0.43	0.23	0.37	0.33
M1	0.18	0.60	0.67	0.56	0.44
M2	0.12	0.38	0.30	0.26	0.40

Classement des algorithmes de clustering

(par algorithme de restruct. sur base des améliorat.)

	Ryder	NN	FN	AV	MA
CWS	2	3	5	4	(1)
A	1	3	4	2	5
B	3	2	5	1	4
AB	3	4	5	2	1
Nearn.	1	3	3	3	2
Ryd(M,b)	2	4	5	3	1
M1	1	5	3	2	3
M2	2	3	4	4	1

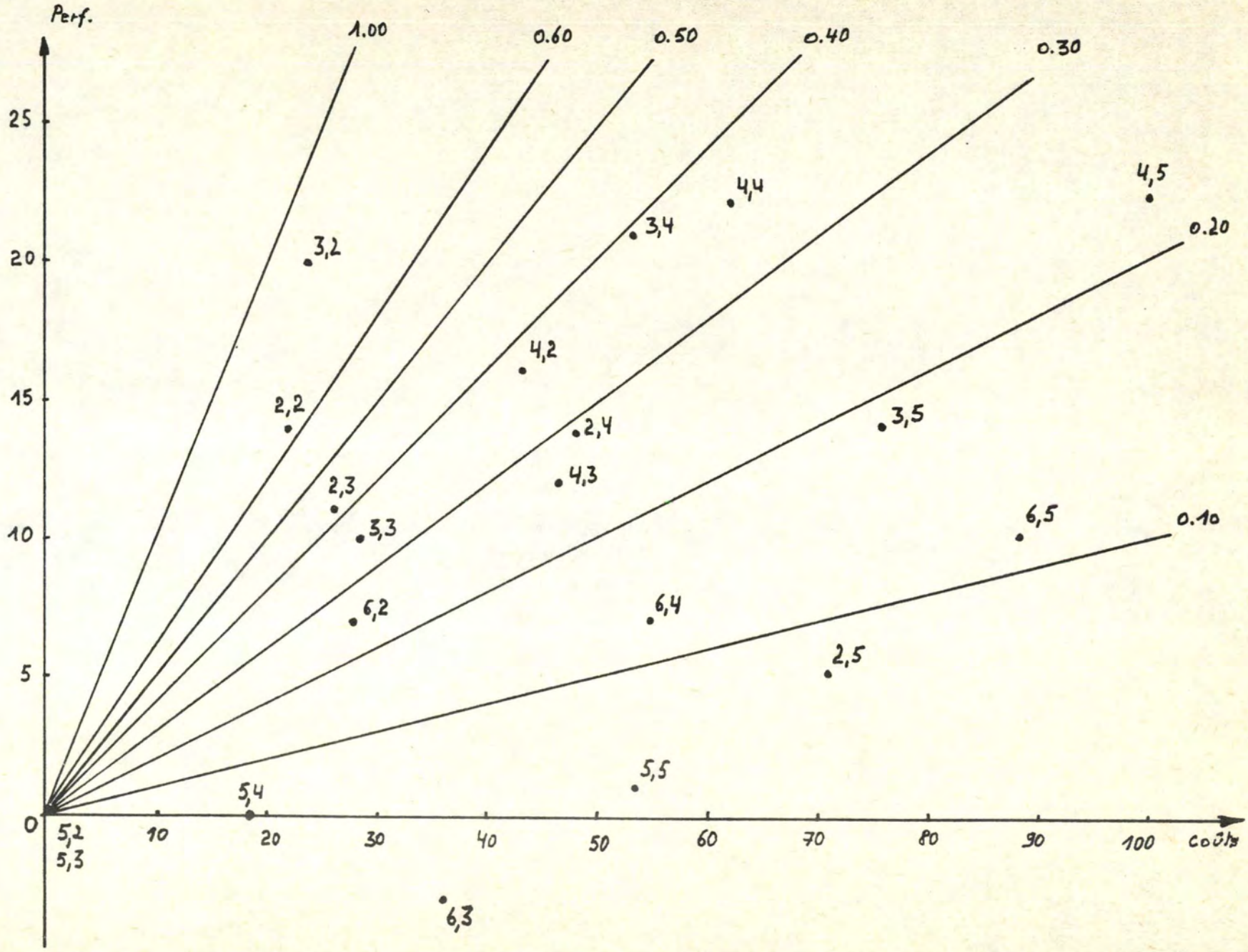
Classement des algorithmes de restructuring

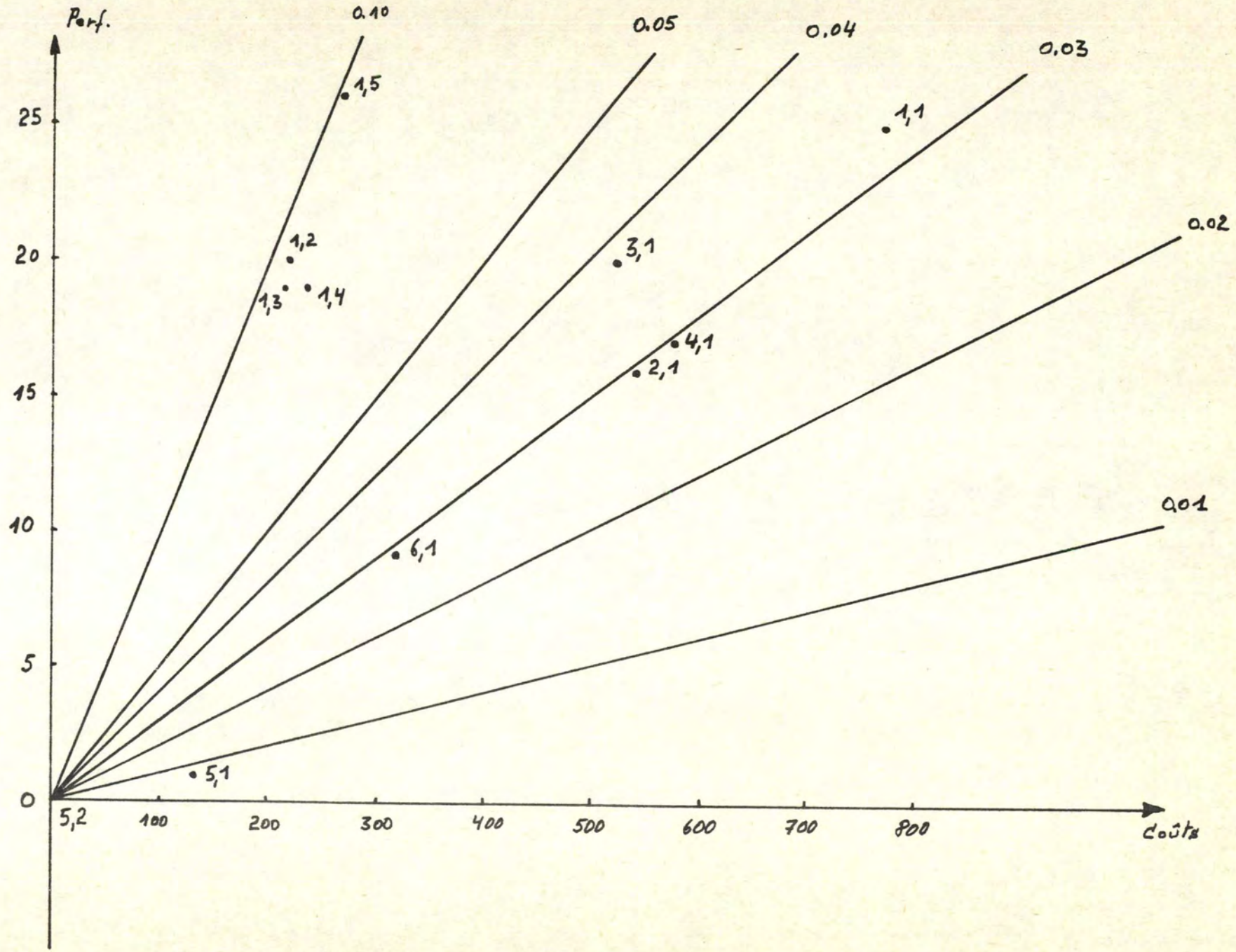
(par algorithme de clustering sur base des améliorations)

	CWS	A	B	AB	Nearness	Ryd(M,b)	M1	M2
RYDER	1	4	2	3	8	6	5	7
NN	2	4	1	3	7	5	6	8
FN	1	3	4	2	6	7	5	8
AV	3	4	2	1	7	5	6	8
MA	(1)	7	3	2	8	4	5	6

Le tableau suivant donne les augmentations de coûts (temps CPU; %) et améliorations (diminution du nombre de défauts de page ; %) observées par rapport aux résultats de la procédure la moins coûteuse (Nearness - NN). Ce tableau est transposé dans les graphiques qui suivent. Les points sont identifiés par deux chiffres représentant respectivement l'algorithme de restructuring et de clustering concernés. Les combinaisons contenant les algorithmes M1 et M2 ne sont pas reprises, étant donné la sensibilité des coûts à l'échantillonnage, qui fausserait la relativité des résultats.

		1		2		3		4		5	
	C. / R.	RYDER		NN		FN		AV		MA	
		Coût	Perf.	Coût	Perf.	Coût	Perf.	Coût	Perf.	Coût	Perf.
1	CWS	772	25	217	20	215	19	238	19	269	26
2	A	540	16	22	14	26	11	48	14	71	5
3	B	522	20	24	20	29	10	53	21	76	14
4	AB	574	17	43	16	46	12	62	22	99	22
5	Nearness	131	2	0	0	1	0	18	0	53	1
6	Ryd.(m,t)	315	9	28	7	36	-3	55	7	88	10


Perf.
1.00
0.60
0.50
0.40
0.30
0.20
0.10
25
20
15
10
5
0
3,2
2,2
2,3
3,3
6,2
5,4
5,2
5,3
6,3
4,2
2,4
4,3
3,4
4,4
4,5
3,5
6,5
6,4
2,5
5,5
10
20
30
40
50
60
70
80
90
100
coûts


Perf.
0.10
0.05
0.04
0.03
0.02
0.01
25
20
15
10
5
0
1,5
1,2
1,3
1,4
3,1
1,1
4,1
2,1
6,1
5,1
5,2
100
200
300
400
500
600
700
800
Coûts

9.2.6 Commentaires.

La taille du working set moyen et maximum diminue, pour toutes les combinaisons (sans densification), dans une proportion variant de 1/4 à 1/8.

La plus importante partie des améliorations réalisables lors de la restructuration d'un programme, peut être obtenue pour un coût modeste. Une diminution du nombre de défauts de page provoqués,de l'ordre de 20 %, est possible pour un investissement limité. Par contre, le passage à un niveau de performances supérieur nécessite l'utilisation d'algorithmes plus complexes et se répercute sensiblement dans les coûts.

Les algorithmes de "clustering" les plus élaborés permettent de relever le niveau des résultats obtenus par des algorithmes de "restructuring" peu coûteux, mais relativement peu performants.

Les algorithmes les plus performants étant les plus coûteux, leur utilisation sera vraisemblablement réservée à des programmes de taille moyenne. Nous avons de plus le sentiment qu'il est préférable d'utiliser les algorithmes les plus performants en échantillonnant la chaîne de références, plutôt que les algorithmes les moins coûteux.

9.2.7. Options proposées.

La poursuite des essais passe par un impératif de réduction des coûts. Le processus étant itératif, les résultats seront affinés au fur et à mesure des tests effectués sur les cobayes sélectés.
La contrainte de coût nous amène à prendre des options. Certaines décisions vont paraître arbitraires dans la mesure où elles ne reposent sur aucune "certitude statistique", mais plutôt sur les indications résultant des premiers tests, sur l'expérience acquise en les réalisant et sur les travaux effectués par certains auteurs. (11,22)
Dans cette optique, les décisions ci-après apparaissent raisonnables :

- élimination des algorithmes suivants :
  - clustering FN
    Son coût est toujours supérieur à celui de NN pour des performances inférieures (sauf un cas où la différence est minime).
  - restructuring A
    Pour des coûts sensiblement égaux, l'algorithme B fournit de meilleurs résultats (sauf un cas où la différence est minime).
  - restructuring de Masuda $m_i * m_j$ (noté M2)
    Pour les mêmes raisons, vis-à-vis de l'autre algorithme de Masuda (noté M1). Pareilles conclusions ne peuvent être tirées à ce stade pour

les algorithmes Ryder (m,b) et Nearness Matrix. La sensibilité de M1 vis-à-vis du programme à restructurer et des paramètres d'échantillonnage n'étant pas connue, le risque d'erreur est de ce fait trop grand, étant donné la différence de philosophie entre M1 et les deux précédents cités.

- de la méthode de densification par tassement arbitraire pour les raisons exprimées au § 9.2.3.

- L'utilisation du simulateur LRU est sujette à caution. Il a tendance à pénaliser les algorithmes orientés "working set". Les résultats fournis varient très fort en fonction du nombre de pages alleuées, et les améliorations sont surestimées, ce qui est apparu en vérifiant la concordance de quelques résultats avec les études réalisées par certains auteurs. Par contre, WS donne entière satisfaction à ce niveau. L'utilisation du simulateur LRU devrait être limitée aux problèmes de comparaison de la même procédure sous des conditions différentes.

- Le processus de densification est à dissocier de la procédure de restructuration telle que nous l'avons considérée jusqu'à présent. Son utilisation sera subordonnée
  - au programme à restructurer (taille des blocs et du working set),
  - à la taille de la page,
  - dans une moindre mesure, à l'algorithme de "clustering" utilisé.

9.2.8. Prolongements.

La poursuite des essais viserait essentiellement à

- acquérir des certitudes quant à l'influence de l'échantillonnage. Ce dernier constitue un moyen des plus efficaces pour réduire les coûts dans une large mesure, impératif à satisfaire si on désire effectuer des tests à une échelle plus vaste.
- définir les conditions précises d'utilisation de la densification.
- déterminer l'opportunité de conserver les algorithmes de "restructuring" Nearness matrix et Ryder (m,b).
- La phase de simplification terminée, des essais intensifs (avec paramètrage) sur les algorithmes conservés devraient permettre de définir valablement leurs coûts et performances pour un investissement devenu acceptable.

CHAPITRE 10 : CONCLUSION

Dans un système à mémoire virtuelle paginée, la restructuration des programmes est hautement utile. Sans remettre en cause le système, ni poser de contraintes inacceptables, elle permet d'adapter les programmes de façon à réduire le nombre de défauts de page provoqués et la taille du working set. L'amélioration se traduit aussi bien dans le comportement des programmes concernés (elapsed time et vraisemblablement, temps CPU consommé) que dans le "throughput". Des résultats appréciables peuvent être obtenus pour un investissement modeste. La modularité de plus en plus grande constatée dans les programmes, lui ouvre de larges perspectives.
En dépit du nombre restreint d'essais effectués, des indications intéressantes ont pu être déduites. Résumons sommairement les principales :

- difficulté de prévision du coût des algorithmes,
- efficacité de l'échantillonnage quant à la réduction des coûts, sans dégrader excessivement les performances,
- utilisation sélective de la densification,
- résultats comparatifs (coût-efficacité) des combinaisons des algorithmes,
- élimination de certains algorithmes,
- défiance vis-à-vis du simulateur LRU.

La nécessité de poursuivre les essais est évidente pour tirer des conclusions précises. Le coût des essais reste cependant un facteur limitatif.

CONTENU DES ANNEXES

Le lecteur qui le désire pourra trouver dans une farde annexe les listings suivants :

- -Programme source de la restructuration
- -Programme source du simulateur LRU
- -Programme source du simulateur WS
- -Programme source du générateur du fichier de commandes pour le "RELINK"
- -Exemples de sorties des simulateurs
- -Exemples de fichiers de commandes générées

BIBLIOGRAPHIE

01 ANDERSON, REISER, GALATI — Tuning a virtual storage system
IBM SYSTEM J. n° 3 (1975) pp. 246-263

02 BAER, CAUGHEY — Segmentation & optimization of programs from cyclic structure analysis
AFIPS (1972) pp. 23-36

03 BARD — Characterization of program paging in a TS environment
IBM journal of Res. & Dev. (1973) pp. 387-393

04 BELADY, KUEHNER — Dynamic space-sharing in computer systems
CACM vol. 12 n° 5 (1969) pp. 282-288

05 BERGERON, BULTERMAN — A technique for evaluation of user systems on an IBM 370
Soft.,Pract. & Exp. Vol. 5 (1975) pp. 83-92

06 COFFMAN, DENNING — Operating systems theory
Prentice Hall (1973) pp. 290-305

07 COFFMAN, VARIAN — Further experimental data on the behavior of programs in a paging environment
CACM vol. 11 n° 7 (1968) pp. 471-474

08 COMEAU — A study of user program optimization in a paging system
ACM Sympos. on OS principles, Gatlingburg (1967)

09 DENNING — The working set model for program behavior
CACM vol. 11 n° 5 (1968) pp. 323-333

10 ELSHOFF — Some programming techniques for processing multi-dimensional matrices in a paging environment
National computer conf. (1974) pp. 185-193

11 FERRARI — Improving locality by critical working sets
CACM vol. 17 n° 11 (1974) pp. 614-620

12 FERRARI — Tailoring programs to models of program behavior
IBM journal of Res. & Dev. (1975) pp. 244-251

13 FERRARI — Improving program locality by strategy-oriented restructuring
Information Processing 74 (1974) p. 266

14 FERRARI — A tool for automatic program restructuring
Proc. ACM nat. conf. Atlanta (1973) pp. 228-231

15 FICHEFET — Cours de théorie des graphes & applications
Institut d'informatique F.N.D.P.

16 GASPARD — Restructuration de programmes en milieu paginé
(Mémoire de fin d'études) Institut d'infor. (1976)

17 HATFIELD GERALD — Program restructuring for virtual memory
IBM System J. n° 3 (1971) pp.168-192

18 HOLTWICK — Designing a commercial performance measurement system
ACM/SIGOPS Workshop on System Perf. Eval. (1971) pp. 29-58

19 KNUTH — An empirical study of fortran programs
Soft.,Pract. & Exp. vol. 1 (1971) pp. 105-133

20 LOWE — Analysis of boolean program models for time-shared paged environments
CACM vol. 12 n° 4 (1969) pp. 199-205

21 MADISON BATSON — Characteristics of program localities
CACM vol. 19 n° 5 (1976) pp. 285-294

22 MASUDA SHIOTA NOGUCHI OHKI — Optimization of program organization by cluster analysis
Information Processing 74 (1974) 261-265

23 MILLBRANDT RODRIGUEZ — An interactive software engineering tool for memory management and user program evaluation
National computer conf. (1974) pp. 153-158

24 MORRISON — User program performance in virtual storage systems
IBM System J. n° 3 (1973) pp. 216-237

25 RYDER — Optimizing program placement in virtual systems
IBM System J. n° 4 (1974) pp. 292-306

26 TSAO, COMEAU, MARGOLIN — Statistical computer performance evaluation
Academic Press (1972) pp. 103-162

27 VER HOEF — Automatic program segmentation based on boolean connectivity
Spring Joint computer conf. (1971) pp. 491-495

28 — VMOS 9 Narratives (SIEMENS System 4004/151)