Proceedings of Sixth ACM Symposium on Operating Systems Principles (November 1977) 109-116.

AUTOMATIC AND GENERAL SOLUTION TO THE ADAPTATION OF PROGRAMS IN A PAGING ENVIRONMENT

J.Y. BABONNEAU, M.S. ACHARD, G. MORISSET and M.B. MOUNAJJED

Institut de Recherche d'Informatique et d'Automatique, 78, Rocquencourt, France

Abstract

The efficiency of replacement algorithms in paged virtual-storage systems depends on the locality of memory references. The restructuring of the blocks which compose the program may improve this locality. [HATFIELD and GERALD 71] [MASUDA SHIOTA NOGUCHI and OHKI 74] [FERRARI 76].

In confining this restructuring to the link-editing operation, a general and completely automatic solution has been implemented, in the form of a self-adaptative system, on the SIRIS 8 operating system. A reduction of 40 to 70% in the page fault rate has been obtained.

I. Problem Specification

A multiprogramming environment requires a well organized allocation of computer resources among users. This allocation requires, at the level of main storage management, hardware paging or segmentation mechanisms. Performances of such virtual memory systems is strongly influenced by the locality of the programs running on them. In order to improve their efficiency, it is necessary to find a solution to enhance this locality.

The awareness of some manufacturers (in France : Compagnie Internationale pour l'Informatique-Honeywell Bull : CII-HB) and the researches carried out in this field (Comeau, Hatfield and Gerald, Ferrari, Masuda, Shiota, Noguchi and Ohki) prove the importance attached to removing this problem. As early as 1971 Hatfield and Gerald reported that an improvement in the number of page exceptions can be obtained of the order of two-to-one to ten-to-one, improvement much more important than that is realized by replacement algorithms.

Enhancing locality of heavily-used programs can considerably improve performances of the systems. It would appear, however, that no general solution has yet been implemented on a system for the following reasons :

- the effort needed to obtain information on program behavior,
- the impractibility of expecting a programmer to provide the effort for the adaptation,
- the profusion of programming languages.

A program can always be considered as being composed of information "blocks". For example :

- procedures in high level languages,
- sections in assembly languages,
- data blocks : for example vectors and matrices.

To adapt a program requires a restructuring based on an optimal placement of these blocks in the virtual space determined by :

- examination of the program structure before execution,
- dynamic measurements.

Most of the restructurings are based upon measurements obtained after one or several executions. This research shows, as we have verified, the stability of program behavior with respect to the input-data and has enabled them to achieve a reduction of 30-40% of the size of the working-set. While Ferrari is interested mainly in the construction of a restructuring graph representing the program behavior, Masuda et al. studied essentially clustering algorithms, which fix a new section-placement* from dynamic measurements. The solution

*A section is the image of a group of contiguous code or data, characterized by its size and protection requirements.

proposed in other papers involves action at the compiling level.

The method that we have implemented occurs at link-edit time and is characterized by :

- wide applicability, due to its independance from the programming language used,
- high degree of automation attained, releasing the programmer from the burden of adaptation,
- efficiency, as demonstrated by the results of tests carried out on the CII-HB operating system.

Such an environment permits the application of these investigations to a large and varied number of programs.

II. Adaptation tool : the "RELIEUR".

All translation processors transform a source program into an object module described in an intermediate language : the binary object language. The necessity for such a language is closely linked with the fact that any program may be made up of several modules written in different languages. The "RELIEUR", whose primary function is link-editing, assembles the object modules constituting a program to form the load module of the program to be executed. It thus occupies a privileged position in the translation chain, since it has a global knowledge of all object modules and has the responsibility for storage allocation of program blocks in the virtual address space. Moreover, the object modules are all in the same language, making the "RELIEUR" independent of the source languages used.

At this level these blocks are referred to as sections whose mean size is of the order of 1/3 to 1/2 page. They are very different from one to another : few big sections and many small ones. The placement decision is essentially that of deciding where to locate these sections in the linear virtual address space, with particular attention to the location of page boundaries. The section placement is founded upon two kinds of measurements deduced from the program :

- during link-editing, the "RELIEUR" displays the inter-sectional links and stores these informations into a static matrix,
- at execution time, the inter-sectional references are accumulated in a dynamic matrix.

The static matrix is useful in the study of the other aspects of program structure : with regards to the constitution of a program library, the transitive closure of this matrix allows in a single access, the retrieval of all library elements necessary and sufficient to the program. But it is the dynamic matrix which provides an image of the program behavior and thus confers to the "RELIEUR" its fundamental role in the restructuring process.

III. Mechanisms of adaptation to the physical environment.

A mechanism has been implemented to provide two new functions for the "RELIEUR" :

- visualization of program behavior,
- adaptation to the physical memory (paginated or segmented).

The different adaptation stages are represented with the help of a triangle, referred to as self-adaptation triangle (figure 1). The "RELIEUR" constitutes the triangle's entry-point. Exchange of information between these three processors is assured through the load module of the studied program. It is in this module, created by the "RELIEUR", that all information relative to the restructuring is filed, in particular the static matrix during the first link-editing and the dynamic matrix after execution.

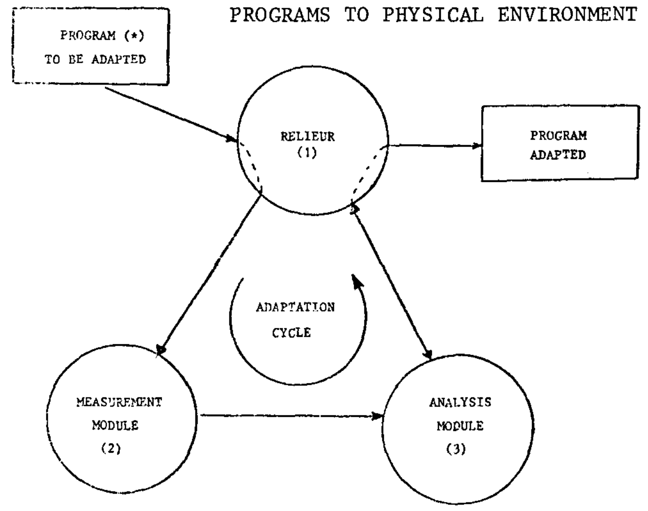
Let us consider all possible ways of traversing this triangle :

- during the first link-edit the "RELIEUR", having created the static matrix, could ask the analysis module for a study of the structure (connection 1-3) which will then be taken into account for the section placement of the program,
- later the program behavior is examined during one or several executions by the measurement module (2) which constructs the dynamic matrix.
- once these measurements are available, the analysis module (3) deduces a new section placement schema,
- the "RELIEUR" restructures the program by creating a new load module, making use of this new schema.

A great deal of flexibility has been introduced in this design. This process could be repeated any number of times while surveying the restructured or the initial program with other sets of data.

During a second survey, the dynamic matrix could be reinitialized to zero or used to store the accumulated references. A second restructuring would then be possible and the whole process repeated if required.

SELF-ADAPTATION TRIANGLE OF PROGRAMS TO PHYSICAL ENVIRONMENT



*No assumption is made about :
- program's nature (scientific, management..)
- problem-oriented language

Figure 1.

We provide a number of flexible tools whose mechanism will be described below :

1) Measurement module

The analysis of program behavior and the restructuring is made in terms of sections ; therefore the intersectional references should be noted during execution. An attempt to access a page absent from the main storage memory provokes a trap exhibiting an address-pair (ad_1, ad_2) :

- ad_1 is the statement address provoking the trap,
- ad_2 is the address of the word referenced.

Having a knowledge of the section placements we can then transform the address-pair (ad_1, ad_2) into a section-pair (I,J) which will be represented by a section-to-section matrix, referred to as the dynamic matrix.

This method of measurements is quite rapid since it makes use of the machine's interrupt-system, but it cannot reveal all intersectional references since only these provoking a trap are traced. However, in reducing memory space, we obtain more and more references, so one can approximate the page reference string, i.e. program behavior [Batson 76]. The matrix so obtained tends towards the nearness matrix as defined by Hatfield and Gerald. Nevertheless, in all our experiments, it was not necessary to go below a memory space inferior to 1/4 of the size of the program. The great interest of this measurement system is its low cost and the ease with which it can be used. A program of one minute (C.P.U.) needs on the average only 30 more seconds for obtaining the dynamic matrix. The principal difference between our work and that of Ferrari [Ferrari 74] is that Ferrari is interested mainly in restructuring algorithms which compute the labels of the restructuring graph from the dynamic behavior. Our measurement system yields a better stability in the efficiency of restructuring than that of Ferrari, owing to the accumulation of information in the dynamic matrix obtained in different cycles.

Moreover, it is possible to distinguish code from data to compare the relative influence of each set on program behavior.

The analysis module then makes use of these measurements.

2) Analysis module.

This module performs three functions :

a) visualization of matrices and automatic curve tracing which enable a user to evaluate his program's behavior.

b) Segmentation based on the static or dynamic matrix. The segments form a partition of a program's sections such that :

- all sections belonging to the same segment are strongly connected with one another,
- segments are weakly connected with each other.

A cluster analysis algorithm realizes this partitioning [Diday 71].

c) Segment placement in a physical paged memory.

Two kinds of placement algorithms have been developed.

Those of the first kind are simple and very fast and aim at forming a nucleus while avoiding all overlaps of sections with page boundaries. First, the sections are classified in a list in decreasing order of their density :

$$\text{density}(i) = \frac{\sum_j (r_{ij} + r_{ji})}{\text{size}(i)}$$

where r_{ij}'s are the elements of the matrix.
Then we place in a page the sections in this order except those which cause an overlap until the page is filled or the list is completely examined. The next page is then filled in the same way with a new list composed of the remaining sections classified in the same order. And so on, until the list is empty. This placement algorithm leads to a set of heavily used pages (the nucleus) and to another set of less used or unused pages while losing very little space at the end of each page. In this article we present the results obtained with this algorithm. It shows that simple and fast algorithms can be efficient for restructuring.

The algorithms of the second type are more complex algorithms based on a hierarchical classification and have also been implemented. Such algorithms have been used by Masuda et al. [74]. At each step the two nearest elements are grouped according to a similarity measure, if the size of the result is inferior to that of a page. This measure is the ratio of the internal connections (between the two elements) on the external connections (of the two elements with all other elements). In the next step, the two previously grouped elements become a single element. We found, as Hatfield and Gerald did, that it is better to keep however some overlaps in an attempt to reduce fragmentation caused by elements not filling pages. The results obtained by these hierarchical classification algorithms will not be discussed here, but we can however state that simple algorithms lead sometimes to better results than that of sophisticated ones.

IV. Presentation of results.

The principal results presented here have been obtained under the experimental system ESOPE [Bétourné et al. 1970] and with the C10 version of the paging operating systems SIRIS 8 of CII-HB.

1) Comparison of static and dynamic matrices (figures 2 & 3).

One notes from these figures that :

- the elements of the dynamic matrix (weight of the connections between sections) are very different from one to another,
- the dynamic matrix is considerably more sparse than the static matrix.

These differences reveal clearly the impossibility of program behavior forecasting based on a static analysis.
The results of several executions of the same program have been accumulated in a dynamic matrix without causing any noticeable variations to appear. This indicates the stability of program behavior with respect to input-data.

2) Relative influence of code and data.

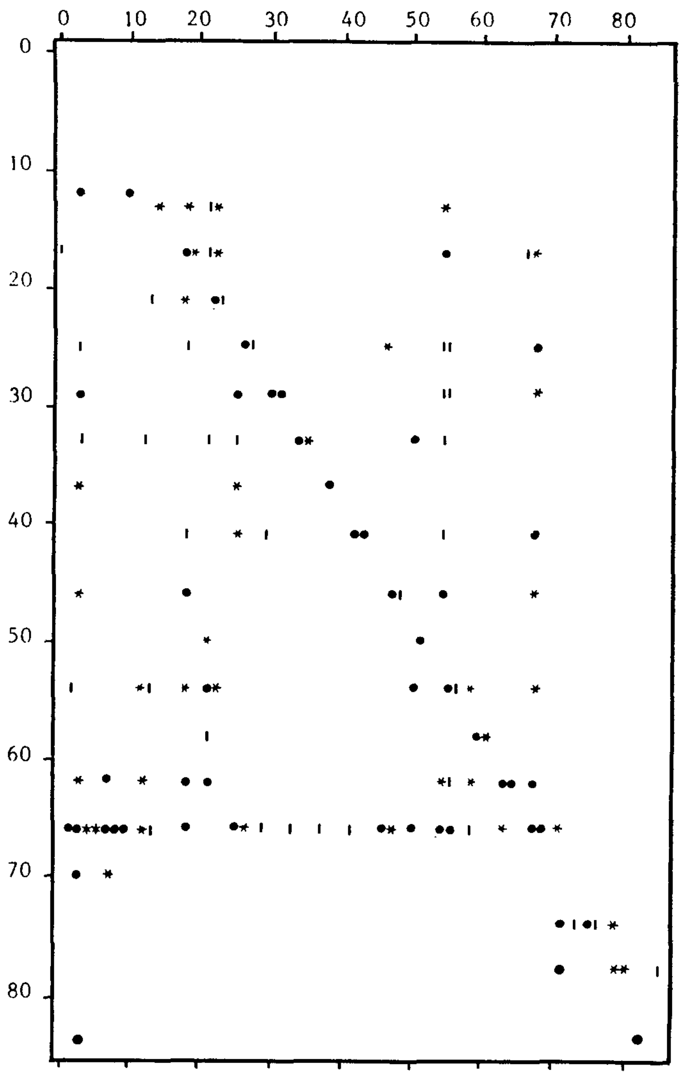
This study has been carried through with the experimental system ESOPE and has led to the construction of the table representing the gain of page exceptions (P.E.) as a function of the percentage of code pages (P.C.P.) and the percentage of data pages (P.D.P.) allocated in the main memory (figure 4).
The gain G is defined by : G = (N-N')/N
where N and N' are the number of page exceptions before and after restructuring, respectively.
It can be seen on this figure that in A, where restructuring concerns data, there is à 9% gain while in C, where the restructuring affects the code, there is a gain of 88% ; in the same constraints of physical space for code and data (close to B) the number of page exceptions is greater than in A and C and the gain approximatively of 45%.

These results have awakened the interest of the CII-HB manufacturer and encouraged us to validate our restructuring through an efficiency verification on the operating system SIRIS 8-C10 (paging version) using a wide range of programs.

3) Efficiency of restructuring.

The results represented in figures 5, 6 describe the adaptation to a paging environment of the system's text-editor and of a translator i.e. the "RELIEUR" itself.

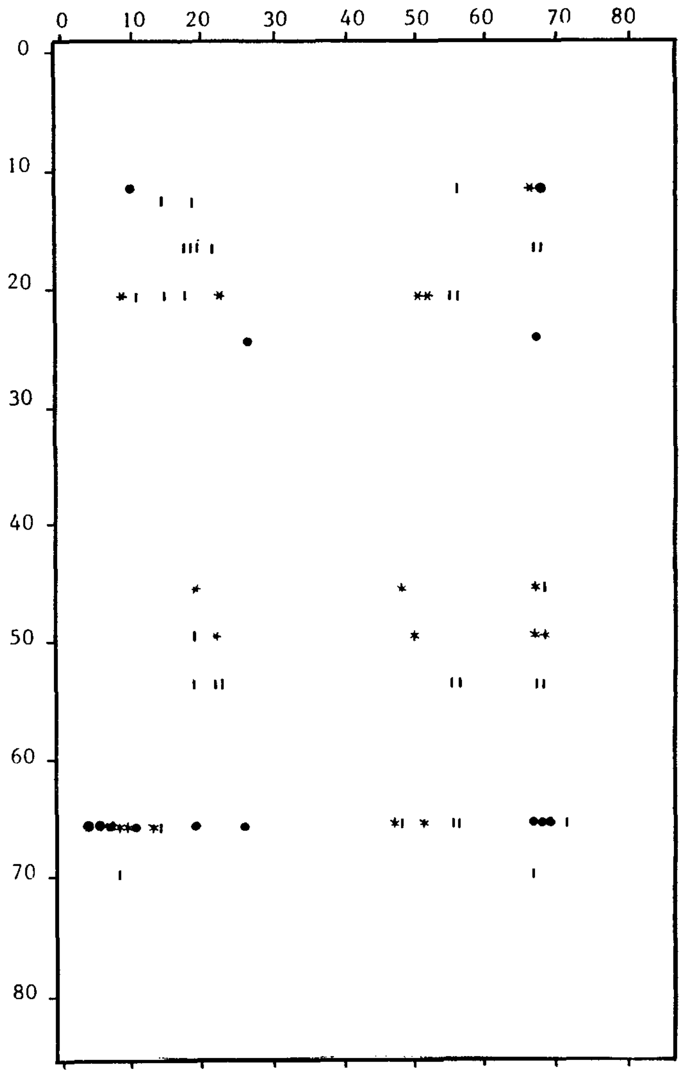
The x-axis represents the ratio R/V where R corresponds to the physical space allocated to the processor during the execution and V the virtual space required. The y-axis represents the efficiency E of the paging performance, calculated by the



Inter-sectional static matrix

ı	0 < weight ≤ 3
*	3 < weight ≤ 10
●	weight > 10

Figure 2



Inter-sectional dynamic matrix

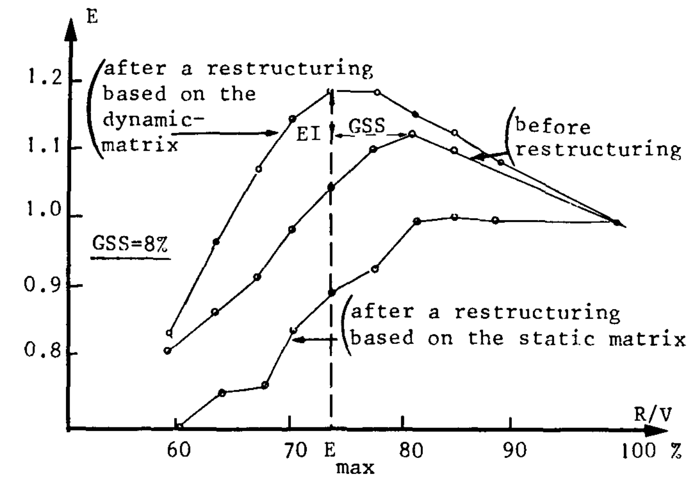
ı	0 < frequency < 100
*	100 ≤ frequency < 1000
●	frequency ≥ 1000

Figure 3

		Gain
A	PCP = 100% PDP = 25%	9%
B	PCP = 20% PDP = 25%	50%
C	PCP = 20% PDP = 100%	88%

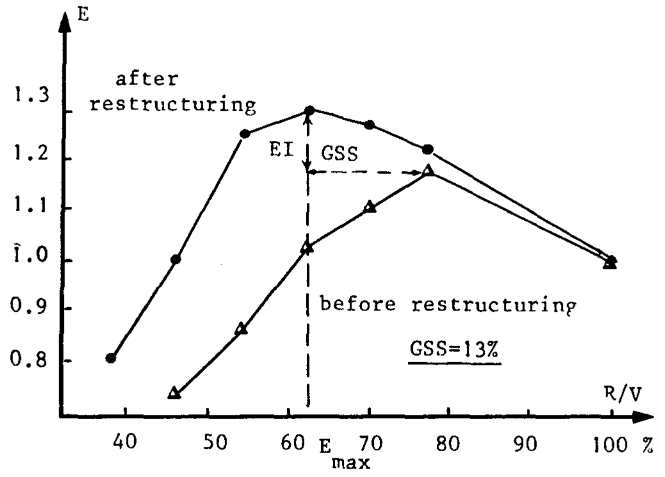
$$\text{Gain} = \frac{\text{PE before} - \text{PE after}}{\text{PE before}}$$

Figure 4



Paging efficiency curves for text-editor (V=27 pages)

Figure 5



Paging efficiency curves for the RELIEUR (V=67 pages)

Figure 6

SIRIS 8 system.

E is given by :

$$E(R) = \frac{V \times T_{R=V}}{R \times T_R}$$

Where : - R and V are the variables defined above,

- $T_{R=V}$ corresponds to the total execution time of the program with no paging (R=V),
- T_R corresponds to the total execution time with paging (C.P.U. time + Input/Output waiting time) when the system allocates R pages to the program (R<V).

When the restructuring is based on the dynamic matrix, figures 5 and 6 show, in particular at points of maximal efficiency E_{max} (zone of optimal use), simultaneously :

- an efficiency improvement (E.I),
- a gain in storage space (G.S.S.).

We observe, as Ferrari did, that restructuring based on the static matrix leads to a deterioration when using the same algorithm that was used with the dynamic matrix.

We detail on figure 7 the different gains obtained after restructuring the "RELIEUR" in function of the ratio R/V.

GT : Gain of total number of page exceptions.

GO : Gain of number of pages modified and written upon disk (PAGE-OUT).

GI : Gain of number of page exceptions causing a page to be effectively loaded (PAGE-IN). For the SIRIS 8 system the cost of "page-in" transfers is greater than that of "page-out". At the point E_{max} as previously defined and corresponding to that of figure 6, we observe that the gain is maximum for the "page-in" transfers.

In collaboration with CII-HB, the measurements were obtained under the following conditions :

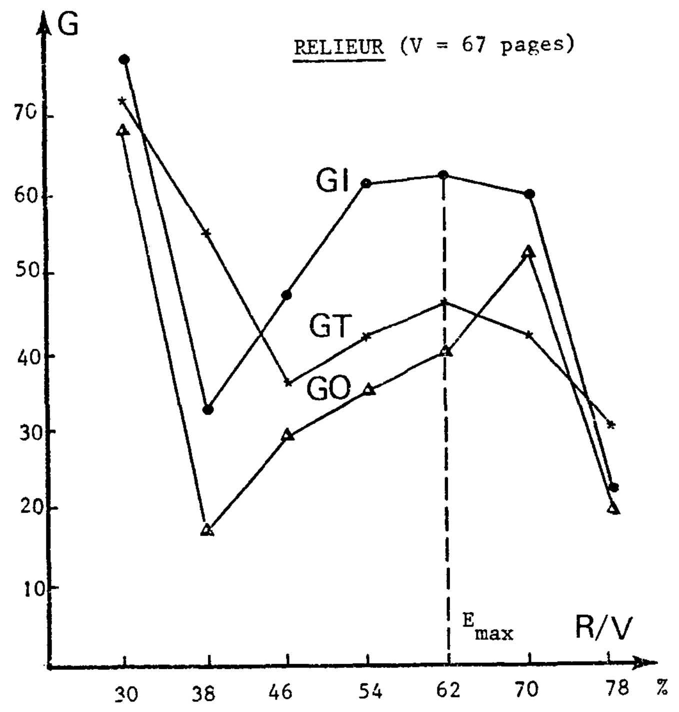
- only 60% of the editor and 80% of the "RELIEUR" were in the static zone, so were controlled, while the rest was in the dynamic zone : under this system a program always contains a static part in which the placement of sections is determined at link-editing and can be restructured.

Sometimes a part of memory space can be allocated dynamically during execution (for instance Input-Output buffers) and cannot be restructured, and this space comprises the dynamic zone . We hope to attain still greater improvement for entirely controllable programs.

- only nucleus-constituting algorithms were used. New measurements are being carried out using segmentation algorithms.

The results obtained show that :

- the phenomenon of "thrashing" is delayed in the zone where the ratio R/V is small. The improvement in the total number of page exceptions is 70% to 80%. In the optimal paging conditions of a program's use, the average improvement is :
  - for the total number of page exceptions 25 to 45%,
  - for page-out transfers : 30 to 40%,
  - for page-in transfers : 60 to 70%,
- the cost of restructuring which corresponds to two circuits in the triangle followed by the final link editing is equivalent, for the results we have presented, to the cost of five link-editing. This is negligible in comparison with the gains realized and the frequency of use of the processors.



Different gains for the RELIEUR

Figure 7

CONCLUSION

The problem of program adaptation to a paging environment arises from the superposition of a logical structure onto a physical medium. The latter is submitted to constraints of memory management which have no connection with the logical structure of programs. The importance of restructuring research as a solution for program adaptation comes from the great influence of such adaptations on system efficiency.

The solution we have proposed is characterized by its complete transparency at the user level. It is automatic and independent of the program and the source language used. Moreover we point out :

- its applicability to a non-limited number of programs in a computing system,
- its negligible cost when judged in terms of the processors' frequency of use.

Its efficiency confirms the validity of this solution.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge many stimulating discussions and advice with Professor J. ARSAC of University of Paris VI. Also appreciated were the encouragement of Professor E. GELENBE of the University of Paris-Nord and the assistance of the following members of the CII-HB System Center : J.L. ROULE, P. COUPAYE and M. HAMARD.

The authors are grateful to Professor D. FERRARI for the interest he shared in this research.

They also wish to express their gratitude to Professor A. BATSON for his extremely penetrating comments and constructive suggestions, which have greatly contributed to the improvement of the paper.

REFERENCES

1. M.S. ACHARD. "Segmentation automatique des programmes indépendamment des langages de programmation". Thesis, University of Paris VI. 1975.
2. A. BATSON. "Program behavior at the symbolic level". Computer november 1976.
3. C. BETOURNE, J. BOULENGER, J. FERRIE, C. KAISER, J. KOTT, S. KRAKOWIAK, J. MOSSIERE, "Process management and resource sharing in the multiaccess system ESOPE". C.A.C.M. December 1970. (13, 12 pages 727-733).
4. L.W. COMEAU. "A study of the effect of user program optimization in a paging system" A.C.M. Symposium on operating systems principles.1967.
5. E. DIDAY. "Une nouvelle méthode en classification automatique et reconnaissance des formes : la méthode des nuées dynamiques". Revue de statistique appliquée. Vol XIX n° 2 1971.
6. D. FERRARI. "A tool for automatic program restructuring". conférence de l'A.C.M. Atlanta Georgia. August 1973.
7. D. FERRARI. "Improving program locality by strategy oriented restructuring". Information processing 1974. North Holland publishing company.
8. D.J. HATFIELD. "Experiments with page size, program access pattern and virtual memory performance". I.B.M. Journal of research and development : 16,1 pages 58-66. 1972.
9. D. FERRARI. "Improving locality by critical working sets". C.A.C.M. 17(11). 1974 pages 614-620.
10. D. FERRARI. "The improvment of program behavior", Computer. 1976.
11. D.J. HATFIELD and J. GERALD. "Program restructuring for virtual memory". I.B.M. System Journal 10 n° 3, pages 168-190. 1971.
12. T. MASUDA, H. SHIOTA, K. NOGUCHI and T. OHKI. "Optimization by cluster analysis". I.F.I.P. 1974.
13. G. MORISSET. "Adaptation automatique des programmes au milieu paginé". Thesis, University of Paris VI. 1975.