central processor and thus attain a high resultant efficiency in conjunction with relatively low speed on the part of the central processor.

LITERATURE CITED


TOWARD MECHANICAL ANALYSIS OF THE COMPUTATIONAL COMPLEXITY OF ALGORITHMS

G. P. Kozhevnikova

Analysis of the computational complexity of algorithms and programs occupies a place of special importance among the theoretical and applied problems of modern programming.

A fairly large body of work deals with the complexity of algorithms and computations. It has been mainly motivated by the decisive role of complexity analysis for the construction of efficient algorithms in all fields of data processing. Several review papers (see, e.g., [1, 2]) highlight some theoretical and applied aspects of complexity analysis, such as computation models, hierarchy and properties of complexity classes in application to different models, methods of analysis, and evaluation of particular algorithms of various classes. However, none of these reviews touches on the subject of mechanical complexity analysis. The purpose of this article is to partly fill this gap: We consider the problem of automating or mechanizing the analysis of algorithmic efficiency, briefly review the work done in the field of mechanical analysis, propose a certain approach to the construction of mechanical analyzers, and report on some of our experience.

1. MECHANICAL ANALYSIS OF THE COMPUTATIONAL COMPLEXITY OF ALGORITHMS AS AN ARTIFICIAL INTELLIGENCE PROBLEM

The complexity characteristic of the algorithm A relative to a given measure M may be represented by the quadruple

\[ S_M(A) = (S_{\text{min}}, S_{\text{max}}, S_{\text{ave}}, S_e), \]  

where \( S_{\text{min}}, S_{\text{max}}, S_{\text{ave}} \) are, respectively, the minimum, the maximum, and the average value of the parameter \( M \), \( S_e \) is the standard deviation from \( S_{\text{ave}} \). Various parameters of the computational process generated by the algorithm A may serve as the complexity measure M. These may include the required computer resources, the number of elementary steps and/or the size of the active storage area of some abstract computer model, the size of the output, the nesting depth of loops, the frequency of referencing different blocks in the algorithm, the cyclomatic number of the program graph, etc.

In general, the complexity analysis of the algorithm A relative to a particular measure M involves successively solving the following two problems:

- identifying the properties of the input data which influence \( S_M(A) \);
- deriving analytical expressions for the components of \( S_M(A) \) in terms of these properties of the input.

If the computational process is related to the input data by fairly complex relationships, the components of $S_M(A)$ as a rule cannot be represented by exact analytical expressions in terms of the known properties of the input: They are commonly calculated by statistical techniques (e.g., using the model of Markov processes [3] assuming that the probabilities of conditional transitions are constant and independent of one another).

The difficulties encountered in the construction of the complexity characteristic $S_M(A)$ and the components of $S_M(A)$ essentially depend on the algorithmic structure of the automaton modeling the computations, the level of detail used in the description of the algorithm, and the machine representation of data. However, all these factors are of secondary importance compared to the main crucial factor, namely the objective complexity inherent in the particular class of problems that the given algorithm is intended to solve.

In order to derive analytical expressions for the components of $S_M(A)$, and especially for $S_{ave}$, a considerable mathematical effort is required even for the simplest algorithms, using nonstandard techniques and various ingenious "tricks" [2]. Combinatorial analysis, probability theory, theory of numbers, algebra, and approximate numerical calculations are all included in the arsenal of basic tools of complexity analysis of algorithms.

Yet despite the fairly low level of "mathematical intelligence" of modern computers, certain trends in system programming insistently require mechanizing the complexity analysis. These trends include, primarily, the development of automatic synthesis systems for programs and compilers using very high level languages (e.g., SETL), where automatic evaluation of problem models, program schemata, and procedures being compiled is necessary in order to generate optimal software constructs. Automatic complexity analysis is also needed in optimizing compilers which select an efficient storage mode for the data in order to maximally simplify the manipulations (the optimizing selection is done by comparing the effect of the different storage structures available in the compiler library on the complexity of the input processing).

The largest application of complexity analysis systems is their use as an instrument for elucidating the behavior of algorithms and programs during the development of problem-oriented computer software, especially in cases of structured interpretation in specialized machines (e.g., MIR). A sufficiently developed analyzer should not only identify for the programmer the "hot points" where the algorithm (the program) is inefficient, but also disclose the causes of this inefficiency by generating the complexity characteristics of the computational process at these points. Given this information, the programmer may attempt to eliminate the sources that increase the complexity of computations and thereby optimize the algorithm.

The traditional menu of artificial intelligence problems thus has been extended to include yet another topical problem, namely the development of systems for mechanical analysis of the computational complexity of algorithms.

The solution of this problem, in our opinion, should start with detailed inventorization and study of the diverse methods, techniques, and tools currently used in complexity analysis. Proceeding from this base, we should then attempt to develop a technology for mechanical modeling of the analytic process, using numerical analysis systems, various problem-oriented mathematical software packages, information retrieval techniques, interactive programming and program proving, and other tools of mechanical mathematics. It is only through this integrated approach that we can hope to solve successfully the problem of mechanizing the complexity analysis.

Some additional factors should be taken into consideration when developing mechanical analyzers for the computational complexity of algorithms:

1) The problem of constructing a universal analyzer capable of generating the complexity characteristic $S_M(A)$ for an arbitrary algorithm $A$ is algorithmically unsolvable, since a prior estimate of the running time of a computation process is not a recognized element of modern programming languages (this follows from the Rice theorem);

2) the multiplicity of mathematical and heuristic techniques used in complexity analysis is virtually infinite given the limited possibilities of modern computers.

These factors place two bounds on the set of algorithms potentially accessible for mechanical complexity analysis: The first factor sets an absolute limit, whereas the second factor identifies the demarcation line between fully mechanized analysis and man-machine analysis. The exact location of this demarcation line and the identity of the factors that may shift it in either direction remain uncertain to this day. Interactive man-machine analyzers, on the other hand, cover practically the entire set of algorithms.