Y. BREITBART
F. D. LEWIS

Combined complexity classes for finite functions

RAIRO. Informatique théorique, tome 13, nº 1 (1979), p. 87-97
<http://www.numdam.org/item?id=ITA_1979__13_1_87_0>

R.A.I.R.O. Informatique théorique/Theoretical Informatics
(vol. 13, n° 1, 1979, p. 87 à 98)

COMBINED COMPLEXITY CLASSES FOR FINITE FUNCTIONS (*) (1)

by Y. Breitbart (2) and F. D. Lewis (3)

Communicated by R. V. Book

Abstract. — *A new measure of computation which combines the three most popular computational resources (time, space, and size) is proposed in this paper. The computable sets are divided into classes according to this measure by taking unions of families of finite problems. These classes are shown to form a hierarchy and are found to differ from the classical complexity classes. Also, bounds are established for decision problems concerning the machine size needed for the computation of Boolean functions.*

1. INTRODUCTION

Studies in computational complexity usually begin with the selection of a computational model and a physical resource necessary for computation. Traditionally, a resource such as time, space, or algorithm size is bounded and the predicates or functions which are computable within the specified amount of the resource are examined. Many natural questions concerning computation can be posed within this framework and many facts about the nature of computation have been brought to light in this manner.

Sometimes, however, the facts and algorithms discovered through traditional methods are not very useful in practical settings. Quite often algorithms which are minimal in one resource use too much of another. Therefore one faces an immediate problem of finding some reasonable trade-off between computational resources.

(*) Received September 1977, revised September 1978.

(1) This research has been supported in part by grant number MCS 77-06321 from the National Science Foundation. A preliminary report on some of this research was presented at the 1977 Conference on Information Sciences and Systems at the Johns Hopkins University.

Department of Computer Science State University of New York at Albany, Albany, New York.

(2) Present address: Department of Computer Science, University of Wisconsin at Milwaukee, Milwaukee, Wisconsin.

(3) Present address: Department of Computer Science, University of Kentucky, Lexington, Kentucky.

Another problem encountered in the standard approach to complexity is that at times the behavior of an algorithm on extremely large inputs becomes so important that much "lower level" computational knowledge is neglected. In fact, optimal algorithms for general problems may be quite efficient for large inputs but require too much of some resource for small inputs. In most cases, practical algorithms are designed with finite, not infinite problems in mind. The algorithm design usually takes resource trade-offs into account for a certain range of inputs and this often has some effect on the algorithm selected to solve the problem.

So, in order to be more in tune with practical algorithm design and to gain computational insight from finite problems, a theoretical framework for the study of computation should include a mechanism for considering algorithms on inputs of various sizes.

The measure of computation we propose combines the three major indicators of complexity: time, space, and program size. This measure should help to formalize some of the trade-off considerations so important in the study of practical algorithms. The measure is defined below, and classes of problems are examined with respect to it.

Input size is taken into consideration by taking what might be called the "finite-problem" or, more specifically, the "Boolean function" approach to complexity. Instead of asking questions of the form "For all inputs...", we shall state: "For inputs of length n..." and thus complexity considerations will always be a function of input length. Therefore we shall study classes of Boolean functions (or, equivalently, monadic predicates with bounded binary inputs) rather than arbitrary recursive functions. This approach to complexity should help to provide specific answers concerning computational behavior at the "lower" levels of complexity hierarchies.

With attention focused on finite functions (or predicates with specified input size) the study of computational complexity can be pushed in directions quite different from the traditional ones followed in the classical space and time considerations.

In section two, combined complexity classes are defined. Then in the next section non-trivial size bounds for these classes are established in order to form a "true" infinite hierarchy. Section four contains several of the usual complexity results applied to the new classes as well as a comparison to standard space and time bounded complexity classes.

In the last section a problem which evolved from the search for a complex (i. e. large in size) Boolean function is considered. Upper and lower bounds of order k are found for membership in the set of Boolean functions which are computable on Turing machines of size k. This result confirms the conjectures of Yablonski [19] and Trachtenbrot [18] that enumeration is the most efficient solution (in terms of size) for this problem. This result also tightens the size bounds placed upon this problem by previous researchers for Normal Algorithms [5, 11, 12] and combinational circuits [17]. Then a more general problem formed by allowing size to grow is shown to be recursively enumerable, but not solvable.

2. PRELIMINARIES

Our computational model shall be the one-tape offline Turing machine. We assume the reader to be familiar with Turing machine concepts on the level of [4]. We shall use a standard, admissible [14] enumeration of machines [denoted M_1, $M_2, \ldots$] and define for each M_i:

(*a*) $T_i(n)$ = the maximum number of steps taken by M_i in the computation of inputs of length n (Time);

(*b*) $L_i(n)$ = the maximum number of tape squares used by M_i in computations on inputs of length n (Space);

(*c*) S_i = the state-symbol product of M_i (Size).

Other measures of size besides the state-symbol product could be used (for example, the length of a machine description) without any significant change in the following results.

By combining the time, space, and size measures described above we now define a *combined measure of complexity* in which input length is emphasized. Using this measure, we also form complexity classes.

DEFINITION: The set A is $\langle t, l, s \rangle$-computable iff for each integer there is a Turing machine M_i such that:

(*a*) M_i computes the characteristic function of A on binary inputs of length n;

(*b*) $T_i(n) \leqq t(n)$ [Time];

(*c*) $L_i(n) \leqq l(n)$ [Space];

(*d*) $S_i \leqq s(n)$ [Size].

NOTATION: The class of all $\langle t, l, s \rangle$-computable sets is denoted COMB (t, l, s).

Combined classes could have been defined as being computable within the resource bounds for "almost all" n but with space and time this is not necessary because these resources have constant speed-up.

Another way of explaining combined computability is as follows. For each set A, form a family of Boolean functions $f_1, f_2, f_3, \ldots$ such that if the binary digits of an integer x are $x_1, \ldots, x_n$ then $f_n(x_1, \ldots, x_n) = 1$ iff x is a member of A. For A to be $\langle t, l, s \rangle$-computable there must be a Turing machine M_i for each f_n such that:

$$M_i(x) = f_n(x_1, \ldots, x_n)$$

and M_i operates within the specified resource bounds on inputs of length n.

The classical deterministic time and space complexity classes are denoted DTIME(t) and DSPACE (t). They are defined in the usual manner, namely all sets computable in time and space $t(n)$.

Whenever we wish to disregard one of the dimensions of complexity it will be replaced with a star. For example, COMB $(*, l, s)$ refers just to space and size – we do not care about time. The length of a string x is denoted by $|x|$, and the size of the Turing machine M is denoted by $|M|$.

3. THE COMBINED HIERARCHY

Since every set may be broken down into a family of Boolean functions as mentioned above, we shall consider the computation of Boolean functions in the proofs which follow.

Because "brute force" table look-up by some finite automaton can be used to compute any Boolean function of n arguments the following trivial observation should be noted.

FACT: Every set is a member of COMB $(n, 1, 2^{n+1})$.

Thus if we are to have a truly infinite hierarchy, (i. e., one which does not degenerate above some point) a size bound for combined classes must be enforced. The following sequence of results provide several bounds. Each size bound contains the symbol ε, which denotes a function $\varepsilon(n)$ that goes to zero as n increases. The first theorem is a slight extension and rewording of a result due to Kuzmin [6].

THEOREM 1: *Every set is a member of* COMB $(5\,n, n, (1+\varepsilon)\, 2^{n+1}/n)$.

Some idea of the trade-offs which are possible within combined classes can be accomplished by comparing the previous theorem with the next one. Its proof involves the computation of subfunctions by finite automata, so size is sacrificed in favor of time and space.

THEOREM 2: *Every set is a member of* COMB $(n, 1, (1+\varepsilon)\, 2^{n+3}/n)$.

Proof sketch: We shall show that every Boolean function f of n arguments is computable within the bounds specified in the theorem. We shall construct a finite automaton to accomplish this by table look-up.

The automaton's states are formed by constructing equivalence classes of binary sequences. First, take all binary sequences of length $k \leqq n$. Two sequences $\bar{x}$ and $\bar{y}$ are said to be *equivalent* iff for every sequence $\bar{z}$ of length $n-k$:

$$f(\bar{x}, \bar{z}) = f(\bar{y}, \bar{z}).$$

A state is associated with each equivalence class of length k sequences for every $k \leqq n$.

To process an input such as $x_1 \, x_2 \ldots x_n$, the automaton progresses through the states associated with the equivalence classes which contain x_1, $x_1 \, x_2$, $x_1 \, x_2 \, x_3$, etc. (Showing that this finite automaton computes f is left to the reader.)

We must now count the number of classes in order to determine this machine's size. For each k the number of classes is bounded by either 2^k (the number of length k sequences), or $2^{2^{n-k}}$ (the number of subfunctions on sequences of length $n-k$). If k_0 is the integer for which:

$$2^{2^{n-k_0-1}} < 2^{k_0} \leqq 2^{2^{n-k_0}}$$

then $n-1 < k_0 + \log k_0 \leqq n$, and the number of states is less than:

$$\sum_{k=0}^{k_0} 2^k + \sum_{k=0}^{\log k_0 + 1} 2^{2^k} = 2^{k_0+2}.$$

Since one can easily verify that for some function ε:

(*a*) $n - \log n \leqq k_0 \leqq n$;

(*b*) $\log n \leqq \log(n - \log n) + \varepsilon$ [since $2^n \leqq (1+\varepsilon) 2^n/n$];

we can easily show that:

$$\log n \leqq \log k_0 + \varepsilon$$

and derive:

$$k_0 \leqq n - \log n + \varepsilon.$$

Thus $2^{k_0+2} \leqq (1+\varepsilon) 2^{n+2}/n$. □

The previous two results show that we must place an upper bound on size in order to have an infinite complexity hierarchy. We shall select $(1-\varepsilon) 2^n/n$ and all sizes mentioned below will be smaller than it. The next theorem establishes this to be the correct bound through the use of a "Lupanov-type" [9] counting argument.

THEOREM 3: *If* $s(n) < (1-\varepsilon) 2^n/n$ *then for no* t *and* l *does* COMB (t, l, s) *exhaust the recursive sets.*

Proof: We shall show that there are more Boolean functions of n arguments than there are Turing machines of size $(1-\varepsilon) 2^n/n$.

If a Turing machine has m states and k symbols, then at most mk quintuples are needed to describe it. A quintuple can be encoded as a binary number which is log mk digits long. (Logarithms are base two.) Thus a Turing machine description can be encoded as a binary number which is mk log mk digits long. Since all of the Turing machines of size mk (the state-symbol product) can be encoded in this manner the number of machines of that size is less than:

$$2^{mk \log mk}.$$

If we assume that mk is strictly less than $2^n/n$ then the number of machines of size less than $(1-\varepsilon)\,2^n/n$ is smaller than:

$$2^{(n-\log n)\,2^n/n},$$

which of course is smaller than:

$$2^{2^n}.$$

This is the number of Boolean functions of n variables and since it is greater than the number of machines of size $(1-\varepsilon)\,2^n/n$, we conclude that there are recursive sets which are not in the class COMB (t, l, s). $\square$

Since no combined class exhausts even the recursive sets (if we enforce a size bound) the previous theorem provides us with a "true" infinite hierarchy – that is, one which never contains all sets for some size.

4. PROPERTIES OF THE COMBINED CLASSES

Having established that the classes we propose to study do form a "true" hierarchy, we now wish to examine several properties of the combined complexity classes. Since the well known time and space complexity classes are recursively enumerable, this will be the first property we shall demonstrate for the combined classes.

Since we wish to compare combined class to DSPACE and DTIME, we shall restrict membership in COMB (t, l, s) to recursive sets in this section.

THEOREM 4: *For every recursive $t \geqq n$, l, and $s \leqq (1-\varepsilon)\,2^n/n$, the class of recursive sets contained in* COMB(t, l, s) *is recursively enumerable.*

Proof sketch: An enumeration of the characteristic functions for recursive sets in COMB(t, l, s) is built much the same way as in the the standard r.e.-ness proofs [3]. For every M_i, merely include the following machine in the enumeration:

$\hat{M}_i(x) = M_i(x)$ if for each $n \leqq |x|$ there is some M_j of size $s(n)$ which agrees with M_i on inputs of size n and remains within the space and time bounds; otherwise $\hat{M}_i(x) = 0$.

Each $\hat{M}_i$ either is a function (M_i) which is computable within the resource or is the characteristic function of a finite set. Since finite sets are computable by machines with a constant number of states the result follows. □

The next step in examining a new hierarchy is to compare it with old ones. By the use of unsolvability methods it follows that these combined classes are quite different from the classical complexity classes. This is accomplished in the following sequence of unsolvability results concerning index sets for DTIME, DSPACE, and the new combined classes.

DEFINITION: The *index set* for a class of sets is the set of all indices of all Turing machines which compute characteristic functions for sets which are members of the class.

If two classes have identical index sets, then of course they must be the same. We shall show that the quantifier structure of the membership problems for DSPACE, DTIME, and COMB are different, thus their index sets must differ also.

For any recursive function t it is known that the membership problems of the index sets DTIME(t) and DSPACE(t) are in Δ_3 of the arithmetic hierarchy. This means that three alternating quantifiers (i.e., $\forall \exists \forall$ or $\exists \forall \exists$) are needed to express membership in their index sets [8]. The next theorem indicates that membership in index sets for combined classes is of π_2 or $\forall \exists$ form.

THEOREM 5: *For recursive $s<(1-\varepsilon)2^n/n$ and every recursive t and l, the index set for all the recursive members of* COMB(t, l, s) *is π_2-complete.*

Proof: We must first show that membership in the index set of COMB(t, l, s) can be expressed in $\forall \exists$ or π_2 form. For any Turing machine M_i, i is in the index set iff for every n there is an M_j such that:

(*a*) M_i and M_j halt on all inputs of length n;

(*b*) M_j and M_i compute the same values on all inputs of length n;

(*c*) $T_j(n) \leqq t(n)$, $L_j(n) \leqq l(n)$, and $S_j \leqq s(n)$.

We immediately note that i is in the index set iff:

$$\forall n\ \exists j\,[(a) \text{ and } (b) \text{ and } (c)].$$

If (*a*) is true then (*b*) and (*c*) are recursive, so the quantifier form of this membership problem depends on (*a*). Since (*a*) can be stated:

$$\exists z\,[T_i(n) \leqq z \text{ and } T_j(n) \leqq z]$$

the quantifiers in the membership problem are $\forall n \exists j \exists z$ which is in $\forall \exists$ or π_2 form. Therefore the index set is a member of π_2.

In [7] it was shown that the lower bound for index sets of classes of total functions is the complete degree of π_2. Thus the index set for the recursive members of COMB(t, l, s) is π_2-complete. $\square$

This theorem allows us to observe in the following corollary that combined classes are not merely a new way of presenting the traditional time and space classes.

COROLLARY: *Combined complexity classes are never the same as any* DTIME *or* DSPACE *complexity classes.*

We shall close this section by noting the location of two sets formed from classes of Boolean functions in the combined hierarchy. An examination of the algorithms of Pippenger [13] for monotone functions and Lupanov [10] reveals that:

(*a*) Symetric$\subseteqq$COMB(n, n, kn);

(*b*) Monotone$\subseteqq$COMB$(n, n, k2^n/n\sqrt{n})$.

5. DECISION PROBLEMS

Since the membership problems for the combined classes are π_2-complete, these classes have no solvable decision problem [14]. This fact is not very surprising since the classical complexity classes have undecidable membership problems. So, in order to formulate solvable decision problems for our classes, we shall concentrate on size rather than time or space and examine a decision problem for a finite set.

In this section, we shall examine decision procesures for determining whether a Boolean function of n variables can be computed by a machine of size k. We shall describe a Boolean function by its truth table (a 2^n-tuple of outputs) and consider the set of tables which are realized by size k Turing machines. (Results about other methods of presenting functions, such as formulas, are summarized by Savage [15, chap. 3].

DEFINITION: A Boolean function f is *size-k-computable* iff there is a Turing machine of size k which computes f.

DEFINITION: T_k^n is the *set of truth tables* (or 2^n-tuples) for size-k-computable Boolean functions of n variables.

This set has previously come up in the quest for a complex Boolean function (i.e. a Boolean function which requires a very large number of states) by researchers such as Sholomov [17] and Yablonski [19]. Others [2, 16] have considered space and time bounds for related problems. Trachtenbrot has conjectured that membership in the set of size-k-computable truth tables is in NP but not in P [18]. The following theorems provide tight size bounds for this problem as well as demonstrate that enumeration is the most efficient method (in terms of size) for solving this decision problem.

THEOREM 6: *There is a constant c such that for any n and k, any machine which accepts T_k^n must be of at least size $k-c$.*

Proof: By construction of a set which is not size-k-computable from a machine which accepts T_k^n.

Let M be the Turing machine which decides membership in T_k^n. Then a machine $\hat{M}$ can be constructed so that it generates truth tables until it finds the first one which is not size-k-computable. (This is done by writing down a 2^n-tuple and checking it with the machine M.) As soon as this non-k-computable table is found then $\hat{M}$ uses it to provide output for its inputs.

Since the generation step above requires a constant number of states and since M must have size larger than k, we know that:

$$|M| = |M| + c > k \quad \text{or that } |M| > k - c. \quad \square$$

The upper size bound for membership in T_k^n is shown in the next theorem and is of the same order as the previously established lower bound.

THEOREM 7: *Membership in T_k^n can be decided by a Turing machine of size ck for some constant c.*

Proof: The machine which decides membership in T_k^n is almost a standard enumeration and simulation device. What makes it interesting is the subtle clock it employs to check the simulation times. Specifically, the machine which accepts T_k^n shall:

(*a*) enumerate machines of size k;
(*b*) enumerate inputs for them;
(*c*) simulate machines on inputs;
(*d*) run a simulation clock.

Step (*a*) can be done by marking off enough space to write down any description of a machine of size k. This requires $k \log k$ space and so $\log k$ states are needed. Since n can be obtained easily from the input which is a table of length 2^n, only a constant number of states are required for step (*b*). Simulation also requires a fixed number of states.

The clock (*d*) comes from an idea of Barzdin [1]. These is a Turing machine M_b of size k and an input x of length n such that $M_b(x)$ halts and runs for more time than any other machine of size k on any input of length n. The clock is, of course, the computation of $M_b(x)$. Our machine must write x on a tape and then use M_b to process it while carrying out steps (a) through (c). Thus n states are required to write down x and k states are needed to incorporate M_b into our machine.

Assuming that n is smaller than k, a summation of the states involved in the above computation is bounded by a constant times k. Since the number of symbols involved is fixed, the size of this machine is of order k also. $\square$

It should be noted that the size of the machine constructed in the last theorem might be smaller than a constant times k. In fact, if the number of symbols used by the "non-effective" clock is larger than the number of symbols used for enumeration and simulation, the size of the machine is k plus several lower order (i.e. $\log k$) terms. This is quite possible and would provide a very tight size bound when combined with the lower bound of the previous theorem.

The set of size-k-computable truth tables can be generalized to T_s, the set of all tables (i.e., n is not fixed) which are $s(n)$-computable. This set is obviously recursively enumerable, but as seen in the next result, not recursive.

THEOREM 8: *Membership in T_s is recursively unsolvable for increasing s.*

Proof : Assume that T_s is recursive. Let M_i be the Turing machine that computes the characteristic function of T_s and let the size of M_i be k. Since M_i accepts tables of length 2^m which are size-$s(n)$-computable we may observe that M_i is deciding membership for the family of sets of the form $T^m_{s(2^m)}$. Setting n equal to 2^m we note that

$$T_s = \bigcup_n T^{\log n}_{s(n)}$$

thus concluding that $T^{\log n}_{s(n)}$ is size-k-computable for every n. This a contradiction because of theorem 6 which states that when $s(n) > k$ then $T^{\log n}_{s(n)}$ cannot be size-k-computable.

6. CONCLUSION

This new approach to the study of computational complexity should provide both pratical as well as theoretical insight into the inherent nature of computation. The combined complexity measure is a framework in which trade-offs may be examined as well as a different way of looking at complexity classes. Also, the finite problem approach used in conjunction with these classes should provide information at the lower levels of complexity hierarchies as well as a new direction for theoretical studies of complexity.

ACKNOWLEDGMENT

The authors wish to thank Peter Bloniarz for suggesting several improvements.

REFERENCES

1. Ya. M. Barzdin, *The Complexity of Programs Recongizing Finite Subsets of Recursively Enumerable Sets*, Doklady Akad. Nauk, Vol. 182, 1968, pp. 1249-1252 (in Russian).
2. Y. Breitbart, *Tape Complexity of the Predicate to be a Complex Boolean Function*, in preparation.
3. J. Hartmanis and R. E. Stearns, *On the Computational Complexity of Algorithms*, Trans. Amer. Math. Soc., 117, 1965, pp. 285-306.
4. J. E. Hopcroft and J. D. Ullman, *Formal Languages and their Relation to Automata*, Addison-Wesley, 1969.
5. M. I. Kanovich and N. V. Petri *Some Theorems About Complexity of Normal Algorithms and Computations*, Dokl. Akad Nauk, S.S.S.R., Vol. 184, No. 6, 1969, pp. 1275-1276 (in Russian).
6. V. Kuzmin, *Evaluation of Boolean Functions by Finite Automata, Normal Algorithms, and Turing Machines*, Prob. of Cybernetics, Vol. 13, 1965, pp. 75-96.
7. F. D. Lewis, *Classes of Recursive Functions and Their Index Sets*, Zeit. f. Math. Logik u. Grund. d. Math., Vol. 17, 1971.
8. F. D. Lewis, *The Enumerability and Invariance of Complexity Classes*, J. Comp. System Sc., Vol. 5, 1971, pp. 286-303.
9. O. B. Lupanov, *Ob odnom Methode Syntesa Schm*, Izv. VUS'ov Radiofizika, Vol. 1, 1958, pp. 120-140.
10. O. B. Lupanov, *One Approach to Synthesis of Control Systems-Principle of Local Coding*, Prob. of Cybernetics, Vol. 14, 1965, pp. 31-110.
11. A. A. Markov, *Normal Algorithms for Computation of Boolean Functions*, Izv. Akad. Nauk, S.S.S.R., Vol. 31, 1967, pp. 161 (in Russian).
12. N. V. Petri, *On Algorithms Related to Predicate and Boolean Functions*, Dokl. Akad Nauk, S.S.S.R., Vol. 185, No. 1, 1969, pp. 37-39 (in Russian).
13. N. Pippenger, *The Realization of Monotone Boolean Functions*, A.C.M. Sym. on Theory of Comp., 1976, pp. 204-211.
14. H. Jr. Rogers, *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, NY, 1967.
15. J. Savage, *The Complexity of Computing*, 1976, Wiley and Sons, NY.
16. C. P. Schnorr, *The Network Complexity and the Turing Machine Complexity of Finite Functions*, Acta Infor., Vol. 7, 1976, pp. 95-107.
17. L. A. Sholomov, *Information Complexity Problems of Minimal Realization of Boolean Functions by Combinational Schemas*, Prob. of Cybernetics, Vol. 26, 1973, pp. 207-256.
18. B. A. Trachtenbrot, *On Problems Solvable by Successive Trials*, Math. Found. of Comp. Sc., 1975 (Lect. Notes in Comp. Science n° 32, Springer-Verlag, pp. 125-137).
19. S. V. Yablonski, *On Algorithmic Difficulties in Synthesis of Minimal Schemas*, Prob. of Cybernetics, Vol. 2, 1959, pp. 75-121.