Some Complexity Issues in Digital Signal Processing

PETER R. CAPPELLO, MEMBER, IEEE, AND KENNETH STEIGLITZ, FELLOW, IEEE

Abstract—Over the past decade a large class of problems, called *NP-complete* [5], have been shown to be equivalent in the sense that if a fast algorithm can be found for one, fast algorithms can be found for all. At the same time, despite much effort, no fast algorithms have been found for any, and these problems are widely regarded as intractable. This class includes such notoriously difficult problems as the traveling salesman problem, graph coloring, and satisfiability of Boolean expressions.

Using FIR filter implementation as an illustration, we describe some problems in digital signal processing that are NP-complete. These include: 1) minimize the number of additions needed to implement a fixed FIR filter; 2) minimize the number of registers needed to implement a fixed FIR filter; and 3) minimize the time to perform the additions of such an FIR filter using *P* adders. Large-scale instances of such problems may become important with the use of programmable chips to implement signal processing.

Our main purpose in this paper is to illustrate the usefulness of asymptotic complexity theory in the field of digital signal processing. The theory discriminates between tractable and intractable problems, sometimes identifies fast algorithms for the former, and justifies heuristics for the latter.

I. Introduction

LOOKING for ways to improve or optimize the use of programmable digital signal processing chips can be a tricky business, and the throughput obtainable often determines whether or not a custom design is required. Using the implementation of an FIR filter as an example, we will formulate some combinatorial optimization problems that are computationally tractable, and some that are likely to be intractable (that is, they are equivalent to problems that are NP-complete).

One purpose of this paper is to identify some problems in digital signal processing that are NP-complete. NP-complete problems are all as hard as any in the large class NP, and are generally regarded as intractable (their best known algorithms require exponential running time). But the main purpose of this paper is not so much to identify specific problems that are intractable as much as it is to illustrate the usefulness of the theory of NP-completeness to digital signal processing. Demonstrating the NP-completeness of a problem is a practical yet theoretically convincing way to discourage attempts at exact, efficient solutions of the problem, and to justify the development of heuristics for its approximate solution. (For details of the theory, see [5].)

Manuscript received August 10, 1982; revised December 27, 1983. This work was supported by the National Science Foundation under Grants ECS-8307955 and ECS-8120037, by the U.S. Army Research Office under Grant DAAG29-82-K-0095, by the Defense Advanced Research Projects Agency under Contract N00014-82-K-0549, and by the Office of Naval Research under Grant N00014-83-K-0275.

P. R. Cappello is with the Department of Computer Science, University of California, Santa Barbara, CA 93106.

K. Steiglitz is with the Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ 08544.

The methodology used to show that a problem is NP-complete is as follows.

1) Construct an algorithm that, given an instance of the problem and a proposed solution, verifies the correctness of the solution in a polynomial amount of time. (This shows that the problem is "easy enough" to be in NP.)

2) Select a problem that has been shown to be NP-complete, and transform it to the problem under consideration. The transformation must be performed by an algorithm that uses only a polynomial amount of time. (This shows that the problem is as "hard" as any in the class NP.)

In this paper we will deal with problems related to the implementation of an FIR filter on a programmable digital signal processing chip whose architecture is roughly that of a random access machine. That is, the chip has a random access memory and an arithmetic/logic unit (ALU).

In Section II we look at the problem of minimizing the number of additions in a shift-and-add realization of a fixed FIR filter. Such a realization precludes the need for multiplication hardware. This leads to Section III, which deals with minimizing the storage requirements of an FIR filter computation. Section IV similarly takes up the problem of minimizing the number of instructions needed to implement an FIR filter. Section V considers a processor scheduling problem arising from the assumption that more than one ALU can be fabricated on the chip and that they can be used concurrently to compute filter outputs.

II. Optimizing a Shift-and-Add Implementation of an FIR Filter

Before stating the optimization problem, we formulate a shift-and-add implementation of an FIR filter. Let us say we are given the following 15-tap symmetric filter:

$$y_n \leftarrow 7(x_n + x_{n-14}) + 5(x_{n-1} + x_{n-13}) + (x_{n-2} + x_{n-12}) - 2(x_{n-3} + x_{n-11}) + (x_{n-5} + x_{n-9}) + (x_{n-6} + x_{n-8}). \tag{1}$$

For notational convenience we let

$$w_0 \equiv x_n + x_{n-14}$$

$$w_1 \equiv x_{n-1} + x_{n-13}$$

$$w_2 \equiv x_{n-2} + x_{n-12}$$
$$w_3 \equiv x_{n-3} + x_{n-11}$$
$$w_4 \equiv x_{n-5} + x_{n-9}$$
$$w_5 \equiv x_{n-6} + x_{n-8}.$$

Then we may rewrite the computation of y_n as follows:

$$y_n \leftarrow 7w_0 + 5w_1 + w_2 - 2w_3 + w_4 + w_5. \quad (2)$$

We rewrite (2), replacing the decimal notation for coefficient values with 5 bit two's-complement notation:

$$y_n \leftarrow 00111 \cdot w_0 + 00101 \cdot w_1 + 00001 \cdot w_2 + 11110 \cdot w_3 + 00001 \cdot w_4 + 00001 \cdot w_5. \quad (3)$$

Collecting like powers of 2 we obtain

$$y_n \leftarrow 2^4(w_3) + 2^3(w_3) + 2^2(w_0 + w_1 + w_3) + 2^1(w_0 + w_3) + 2^0(w_0 + w_1 + w_2 + w_4 + w_5). \quad (4)$$

That is,

$$y_n \leftarrow \sum_{i=0}^{4} 2^i \cdot sum_i. \quad (5)$$

We now are in a position to realize some computational savings. Notice that $w_0 + w_3$ appears in both sum_2 and sum_1. We can save additions by remembering this partial sum and using it wherever it is needed. The question arises as to how much can be saved this way: can we produce the set of sum_i's using only, say, five additions?

We state this question more formally.

Collection of Sums

Input: A collection C of expressions of the form $a_{i1} + a_{i2} + \cdots + a_{ij}$, all $a_{ik} \in A$, a finite set, where an element of A appears at most once in any expression, but may appear in any number of expressions, and a positive integer J.

Question: Is there a sequence S of J or fewer additions that computes all the expressions in C?

Complexity: NP-complete, see [5, "Ensemble Computation"]. This problem remains NP-complete even when expresions are restricted to contain no more than three elements of A. Note also that computing a collection of expressions involving both addition and subtraction is a more general problem and, therefore, at least as hard.

III. Register Allocation (Storage Conservation)

Suppose that we obtain a sequence of add instructions that produces the needed *sums*, provided that all the addition results are held in some registers (we model storage by registers, although one can view these as memory locations). Fig. 1 illustrates graphically the partial order induced by one such add sequence for our example FIR filter. Although it is likely that each partial result is needed (it is certain if the sequence of additions is minimal) and so needs to be placed in *some* register, there remains the question as to how many *distinct* registers are needed. We may reuse registers. Different orderings of add instructions will in general require a different number of registers. Is there a sequence of add instructions that obtains the needed sums and which uses only, say, five registers? This question is formalized by the following game. (Note that a partial order can be represented by a directed acyclic graph (DAG), as can any computable signal flow graph with its delay arcs removed.)

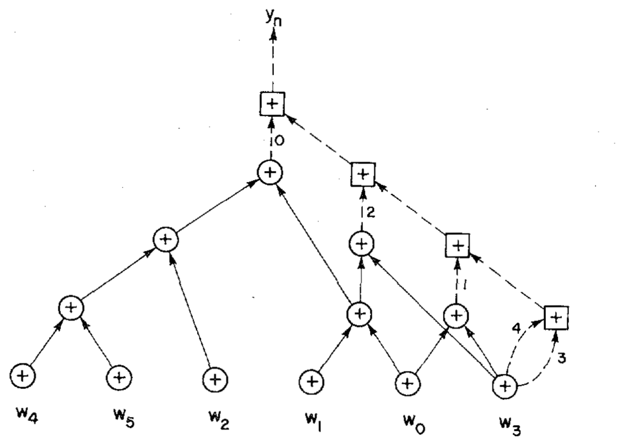


Fig. 1. The directed acyclic graph (DAG) represented by the circular nodes (adds) and the solid arcs illustrates the *sums* part of the FIR filter computation. It is a partial order induced by a sequence of additions. (The dashed arcs and square nodes represent the shift-and-add part of the computation. Dashed arc labels show the shift amounts.)

Definition: Let $G = (N, A)$ be a directed acyclic graph (DAG) such that its nodes have at most two incoming arcs. Let there be an infinite supply of registers. A *move* in the *register allocation game* is one of the following:

a) place a register on a node with no incoming arcs (that is, load a register)

b) pick up a register from a node (that is, declare a register free).

If there are registers on every node coming into $node_i$, then either

c) place a new register on $node_i$ ($r_i \leftarrow r_j + r_k$), or

d) move a register to $node_i$ from one of $node_i$'s incoming nodes ($r_i \leftarrow r_i + r_k$).

Definition: A DAG *register allocation computation* is a sequence of moves in the register allocation game that starts with no registers on any node, and places a register on every node *exactly* once.

The register allocation problems we consider are NP-complete even for this restricted class of DAG's, those with at most two incoming arcs at any node.

Definition: A register allocation computation of a DAG is said to *use k registers* if during some move in the computation there are k registers on nodes of the DAG, and during every other move there are no more than k registers being used.

It is natural to ask how many registers are required to compute a given DAG.

Register Sufficiency

Input: A DAG, and an integer K.

Question: Is there a computation for the DAG that uses K or fewer registers?

Complexity: NP-complete [10].

The related problem that permits a register to be placed on a node *more than once* is PSPACE-complete [6]. That is, finding the fewest number of registers needed when recomputation is allowed is essentially as hard as any problem whose algorithms require a polynomial amount of space. Such problems are at least as hard as NP-complete problems.

Any DAG computation resulting from the register allocation game defines a function from nodes (N) to registers; that is, there is exactly one register associated with each node (but not conversely). One may ask, given a proposed function from nodes to registers, how hard is it to determine if there is a DAG computation that defines that function?

Feasible Register Assignment

Input: A DAG, positive integer K, and a register assignment $f: N \rightarrow \{r_1, r_2, \cdots, r_K\}$.

Question: Is there a computation of the DAG that uses K or fewer registers and that is compatible with the function f?

Complexity: NP-complete [10].

Again, both of these problems remain NP-complete even when each node has no more than two incoming arcs.

IV. Code Generation (Time Conservation)

What if we are more interested in saving instructions than registers? Suppose, for example, that we have, as in Section II, a computation DAG for computing the necessary sums (or a computable signal flow graph with its delay arcs removed). From this DAG we wish to generate succinct straight line code. As in register allocation for DAG's, we model DAG code generation by a game.

Definition: Let $G = (N, A)$ be a DAG. Let there be an infinite supply of registers. A *move* in the *instruction game* is one of the following:

a) if $node_i$ has only one incoming node, $node_j$, and $node_j$ is covered by register r_j, then either

i) place r_j on $node_i$ (a no-op), or
ii) place a new register r_i on $node_i$ ($r_i \leftarrow r_j$)

b) if $node_i$ has left and right incoming nodes and they are covered by registers r_j and r_k, respectively, then either

i) place r_j on $node_i$ ($r_j \leftarrow r_j + r_k$), or
ii) place r_k on $node_i$ ($r_k \leftarrow r_j + r_k$), or
iii) place a new register r_i on $node_i$ ($r_i \leftarrow r_j + r_k$).

Note that the class of DAG's dealt with here is the same as in the register allocation problems.

Definition: A DAG *code generation* is a sequence of moves in the instruction game that starts with registers on all nodes that have no incoming arcs (that is, all inputs are in registers), and places a register on every node.

Definition: A DAG code generation *produces K instructions* if it uses exactly K of the moves above.

It is natural to ask how many instructions are required to compute, in this sense, a given DAG.

Code Generation with Unlimited Registers

Input: A DAG $G = (N, A)$ in which no node has more than two incoming arcs (each arc is considered either a left arc or a right arc), and an integer K.

Question: Is there a code generation that produces K or fewer instructions?

Complexity: If moves b-ii) and b-iii) are forbidden, the problem is NP-complete [11]. If only move b-iii) is forbidden (the commutative variant of the problem), the problem remains NP-complete [11]. If all moves are allowed, the problem enjoys a polynomial-time solution, a fact that may have implications for architects or programmers of digital signal processing chips. In this case a linear-time algorithm for producing a minimum-length instruction sequence is to evaluate the signal flow graph in a bottom-up fashion (that is, starting with nodes that have no incoming arcs), level by level, assigning a distinct register to each node (that is, using only instructions of the form $r_k \leftarrow r_i + r_j$).

TABLE I
PRECEDENCE-CONSTRAINED SCHEDULING PROBLEMS

Processors	Process lengths	Partial order	Complexity
Fixed (=n) Parameter (P)	Identical (I) Different (D)	Forest (F) DAG (D)	
= 2	I	D	$O(n^2)$
= 2	D	F	NP-complete
= 3,4,5,...	I	D	Open
P	I	F	$O(n\,\alpha(n))$†

† $\alpha(n)$ grows slower than log (n), but is not a constant. See [15].

V. Processor Scheduling

In this section we consider programmable digital signal processing chips with more than one ALU. In this scenario the chip can be performing more than one addition at a time. Continuing with our FIR filter example, we now want to see how many ALU's are needed to compute our sums in a specific amount of time. Equivalently, we may wish to know if a given number of processors is sufficient to compute our sums in a desired time frame. By processors (processes) we mean, for example, adders (additions). This context permits us to assume that the schedule

1) is *deterministic* (that is, the filter's requirements are known in advance),

2) is *nonpreemptive* (that is, requires a processor to finish a process, once started), and

3) involves *unit execution time* processes (for example, all additions will be assumed to consume the same amount of a processor's time).

Under these assumptions we would like to know how hard it is to schedule the operations indicated by a partial order (or those indicated by a computable signal flow graph with its delay arcs removed).

Precedence-Constrained Scheduling

Input: A DAG G, P processors, and a deadline D.

Question: Is there a P-processor schedule for G that meets deadline D?

Complexity: NP-complete [12].

We summarize variations of this problem in Table I. See [13] for details.

As we did in the feasible register assignment problem, we consider the feasible precedence-constrained schedule problem. In this problem we are given a set of *individual* process deadlines and ask if there is some schedule that can achieve them. This situation might arise, for example, when one first obtains a schedule for a filter's multiplications, and then seeks a compatible schedule for the required additions. An algorithm for this problem also might be used as the basis of a heuristic for the overall deadline problem.

Feasible Precedence-Constrained Schedule

Input: A DAG G, P processors, and a set D of deadlines, one for each node in the DAG.

Question: Is there a P-processor schedule for G that meets all the deadlines?

Complexity: NP-complete [1]. The problem enjoys a polynomial-time solution if P is fixed at 2, see [14], or if the DAG is such that no node has more than one immediate successor [1].

VI. Conclusions

We have described a variety of FIR filter realization problems in digital signal processing. Many of them are NP-complete. When a problem has been shown to be NP-complete, and we need to solve large instances of it, we are inclined to investigate either

1) restricted versions of the problem (for example, assuming some modularity or regularity) to see if they can be solved efficiently, or

2) fast procedures that give either

   a) suboptimal solutions, or
   b) optimal solutions with probability less than one.

Similarly, when a problem has been shown to have a polynomial-time solution, we may be inclined to investigate more general versions of the problem, again sharpening our understanding of the problem's characteristics vis à vis its computational complexity.

Computational complexity theory is useful, not because it tells us how to solve a particular problem, but because it tells us whether or not a problem is likely to have an efficient solution. It thus guides the direction of future research.

In this paper we have been discussing optimization problems for programmable digital signal processing chips with *random access machine* architectures. Optimized solutions are not well suited for custom implementation on a special purpose FIR filter chip. For example, laying out a directed acyclic graph of adders is unwise, even if the graph has a minimum number of add nodes (for a given FIR filter). This is because the topology of an arbitrary DAG is highly irregular. Moreover, in most digital signal processing applications we wish to minimize, not latency, but *period*. Special purpose architectures for FIR filters (and other functions) should have highly regular topologies (see, for example, [2]-[4], [7]-[9]). As long as there are RAM-architecture digital signal processing chips, however, optimization problems such as the ones described in this paper will need attention. Indeed, such attention may influence the instruction set of such general purpose chips. As was noted in Section IV, for arbitrary computable signal flow graphs with their delay arcs removed, the problem of producing a minimum instruction sequence is *easy* when the chip has three-register instructions, and is *intractable* when the chip only has two-register instructions.

Garey and Johnson [5] provide a comprehensive introduction to the theory of NP-completeness, a catalogue of problems that are known to be NP-complete, and a discussion of some methods for coping with NP-complete problems.

Acknowledgment

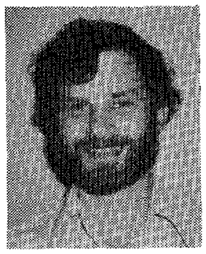
We thank the referees for helpful comments, and for the remark concerning the PSPACE-complete problem.

References

[1] P. Brucker, M. R. Garey, and D. S. Johnson, "Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness," *Math. Oper. Res.*, vol. 2, pp. 275-284, 1977.
[2] P. R. Cappello and K. Steiglitz, "Digital signal processing applications of systolic algorithms," in *VLSI Systems and Computations*, H. T. Kung, B. Sproull, and G. Steele, Eds. Rockville, MD: Comput. Sci. Press, 1981.
[3] ——, "Bit-level fixed-flow architectures for signal processing," in *Proc. IEEE Int. Conf. Circuits Comput.*, New York, Sept. 29-Oct. 1, 1982.
[4] ——, "Completely pipelined architectures for digital signal processing," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-31, pp. 1016-1023, Aug. 1983.
[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA: Freeman, 1979.
[6] J. R. Gilbert, T. Lengauer, and R. E. Tarjan, "The pebbling problem is complete in polynomial space," *SIAM J. Comput.*, pp. 513-524, 1980.
[7] H. T. Kung, L. M. Ruane, and D. W. L. Yen, "A two-level pipelined systolic array for convolutions," in *VLSI Systems and Computations*, H. T. Kung, B. Sproull, and G. Steele, Eds. Rockville, MD: Comput. Sci. Press, 1981.
[8] H. T. Kung, "Let's design algorithms for VLSI systems," in *Proc. Conf. Very Large Scale Integration: Architecture, Design, Fabrication*, California Inst. Technol., Pasadena, Jan. 1979, pp. 65-90.
[9] ——, "Special-purpose devices for signal and image processing: An opportunity in very large scale integration (VLSI)," in *Proc. Soc. Photo-Opt. Instrum. Eng.*, vol. 241, "Real-Time Signal Processing III," July 1980.
[10] R. Sethi, "Complete register allocation problems," *SIAM J. Comput.*, vol. 4, pp. 226-248, 1975.
[11] A. V. Aho, S. C. Johnson, and J. D. Ullman, "Code generation for expressions with common subexpressions," *J. Ass. Comput. Mach.*, vol. 24, pp. 146-160, 1977.
[12] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10. pp. 384-393, 1975.
[13] E. Coffman, *Computer and Job-Shop Scheduling Theory.* New York: Wiley, 1976.
[14] M. R. Garey and D. S. Johnson, "Scheduling tasks with nonuniform deadlines on two processors," *J. Ass. Comput. Mach.*, vol. 23, pp. 461-467, 1976.
[15] R. E. Tarjan, "Efficiency of a good but not linear set union algorithm," *J. Ass. Comput. Mach.*, vol. 25, pp. 215-225, 1975.

Peter R. Cappello (M'83), for a photograph and biography, see p. 33 of the February 1984 issue of this Transactions.

Kenneth Steiglitz (S'57-M'64-SM'79-F'81), for a photograph and biography, see p. 33 of the February 1984 issue of this Transactions.



Kenneth Steiglitz (S'57-M'64-SM'79-F'81) was born in Weehawken, NJ, on January 30, 1939. He received the B.E.E., M.E.E., and Eng.Sc.D. degrees from New York University, New York, NY, in 1959, 1960, and 1963, respectively.

Since September 1963 he has been with the Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, where he is now Professor, teaching and conducting research in the computer and systems areas. He is the author of *Introduction to Discrete Systems* (New York: Wiley, 1974), and coauthor, with C. H. Papadimitriou, of *Combinatorial Optimization: Algorithms and Complexity* (Englewood Cliffs, NJ: Prentice-Hall, 1982).

Dr. Steiglitz is a member of the VLSI Committee of the IEEE ASSP Society, and has served as a member of the Digital Signal Processing Committee, a member of the Administrative Committee, and Awards Chairman of the Society. He is an Associate Editor of the journal *Networks*, and is a former Associate Editor of the *Journal of the Association for Computing Machinery*. A member of Eta Kappa Nu, Tau Beta Pi, and Sigma Xi, he received the Technical Achievement Award of the ASSP Society in 1981, and an IEEE Centennial Medal in 1984.

Resolution Enhancement of Digital Beamformers

HONG FAN, STUDENT MEMBER, IEEE, EZZ I. EL-MASRY, SENIOR MEMBER, IEEE, AND W. KENNETH JENKINS, SENIOR MEMBER, IEEE

***Abstract*–Large array extent is usually required in order to adequately enhance the resolution of a beamformer and to improve its beam pattern. Due to physical constraints this requirement may not be met. This paper presents the results of an analytical and experimental study in the use of a signal extrapolation method to enhance the beamformer resolution for small array extent. A real-time implementation scheme is proposed. It is shown that the beam pattern and the resolution of the beamformer can be improved. Examples for various situations are provided.**

I. Introduction

IN the area of signal processing, considerable attention has been devoted to digital array processing in recent years. This attention is due to the increasingly wide use of array processing for both civilian and military purposes. Digital beamforming, for example, is an active area in digital array processing. Many techniques in digital beamforming have been well established [1]-[8]. It has been shown [1], [8] that the beam pattern, signal-to-noise ratio (SNR), etc., depends on the array length or the array extent and the number of sensors used, i.e., the larger array extent and the more sensors, the better the beam pattern and the SNR become. In a practical situation, however, the array extent may be restricted for economical or physical reasons. In this situation, techniques for improving beamformer performance through signal processing may be useful.

A totally different concept, signal extrapolation, has also been drawing a great amount of interest recently, largely in the area of signal/image restoration. It has been shown that a known portion of a signal can be extrapolated outside of the observation interval if the signal possesses certain properties [9], [12]. Many algorithms, both iterative and noniterative, have been proposed for continuous and discrete cases. References [9]-[13] serve as a good review of this subject.

The purpose of this paper is to use spatial extrapolation in conventional digital beamforming to improve the beam pattern without extending the array length. Effectively, the array length is extended through signal processing. Other currently used techniques such as interpolation beamforming, weighting, and the widely used spectral estimation techniques: the maximum entropy method (MEM), the maximum likelihood method (MLM), etc., can be combined with extrapolation to achieve an overall better performance.

Manuscript received November 22, 1982; revised May 19, 1983 and December 15, 1983. This work was supported in part by the Joint Service Electronics Program (JSEP) under Contract N00014-79-C-0424.

H. Fan and W. K. Jenkins are with the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801.

E. I. El-Masry is with the Department of Electrical Engineering, Technical University of Nova Scotia, Halifax, N. S., Canada B3J 2X4.

II. Background

The following discussion assumes a uniform linear array of omnidirectional sensors as shown in Fig. 1. For convenience, assume that there are a total $2M + 1$ sensors in the array, indexed from $-M$ to M. Also, assume that the signal is band limited, and may be corrupted by additive noise. Furthermore, assume that the beam steering specifications are met by either sampling the sensor outputs at a sufficiently high frequency, or by using digital interpolation beamforming techniques, as described in [2]. These assumptions will be used throughout this paper.

A. Beamforming

The task of detecting a signal and determining its direction can be accomplished by conventional digital beamforming, i.e., by delaying and summing the corresponding sensor signals.