AN ALGORITHM DESIGN ENVIRONMENT FOR SIGNAL PROCESSING

Michele Covell

SRI International, 333 Ravenswood Ave, Menlo Park CA 94025
(work completed at M.I.T., Digital Signal Processing Group, Cambridge MA 02139)

ABSTRACT

Classically, the computer has been used in signal processing simply for numerical calculations, leaving the majority of the design process has been unsupported: in particular, the analysis of the properties of the selected algorithm and the rearrangement of the algorithm to find efficient, input/output equivalent implementations. This paper explores some of the issues involved in supporting the symbolic rearrangement of algorithms. The paper demonstrates the potential of automatic rearrangement through the design of an incoherent detector for sonar FSK-code signals. An innovative implementation, discovered by the computer environment, for the frequency-chip matched filters is presented: this implementation reduces the computational requirements from order $N \log N$ to order N.

I. INTRODUCTION

Development of digital signal processing operations or programs often involves algorithm manipulation in addition to the familiar data pressing and algorithm definition. Algorithm manipulation can involve the analysis of a given signal processing expression or algorithm to determine its properties or its rearrangement to provide an alternate implementation. Examples of property analysis include the determination of the linearity and time-invariance of a subband analysis/synthesis system. An example of algorithm rearrangement is provided by the manipulation of the subband analysis stage to interchange to order of filtering and downsampling.

Although the conceptual distinction between data processing, algorithm definition and algorithm manipulation is clear, most real situations involve all three activities. The focus of the research reported here has been to explore the possibility of a signal processing workstation which provides an integrated environment for data processing, algorithm definition and algorithm manipulation. This research has been the result of a sequence of research efforts[1,2,3,4]. This article focuses on the design environment, ADE, developed in [4]. Like many other signal-processing software environments, ADE can be thought of as a block-diagram environment. A large number of low-level signal-processing "blocks" or systems are defined in ADE: there are more than 210 hierarchal classes of signals and systems in ADE. By stringing systems together, the user of the environment is often able to quickly describe the signal-processing algorithm which is of interest. If some desired signal-processing operation is not available, the user of the environment has the option of defining a new class of signals or systems.

In addition to these basic capabilities, ADE provides facilities for supporting the analysis and manipulation of signal-processing properties. For example, ADE will propagate information about the range and periodicity of signals through signal-processing systems, using information included in the system-class definitions.

Finally, ADE provides the user with a high-level, signal-processing "compiler". This "compilation" facility accepts a signal-processing algorithm and returns a list of the alternate, input/output equivalent implementations which were uncovered using the transformation rules embedded in the environment. While these rules provide the actual transformations which are used in finding the equivalent implementations, the search process which is used to find all the appropriate transformations is comparatively complex.

The remainder of this article focuses on the search process used for finding equivalent algorithms and the results have been achieved using the described approach.

II. MULTIPLE-BEAM SONAR IMAGING USING FSK CODES

The problem of FSK-code detection and discrimination is used within this article as an application area to illustrate the potential and the difficulties of algorithm rearrangement. This section introduces this sonar application of FSK codes.

The use of multiple-beam, code-division sonar[5] avoids the drawbacks of conventional sonar imaging systems. The transmitter is a set of N transducers, each illuminating a different direction and each transmitting a distinct FSK-code signal, S_i for $i = 1, ..., N$. The FSK codes are created from permutations of N individual, uniformly-spaced frequency chips. One wide-beam hydrophone is used as a receiver.

The received signal is a superposition of the reflected energy from each of the illuminated scattering centers. An unknown propagation delay from the combined forward and return paths is associated with each scattering center. In addition, a nonuniform phase distortion between frequency chips is introduced by the scattering characteristics of the object and the fluctuations in the propagating medium.

Using this model, the discrete-time approximation to the optimal detectors uses matched filters to detect the individual frequency chips followed by incoherent addition to create the detectors for the full set of FSK-codes. Since there is an unknown time delay in the return, the output from these detectors is desired at each point in time.

III. UNCONSTRAINED SEARCH FOR EQUIVALENT ALGORITHMS

As mentioned in the introduction, ADE provides partial compilation of high-level signal-processing descriptions. This section provides an initial description of the search space which is explored in finding the equivalent implementations of an algorithm.

The task of finding alternate implementations of a signal-processing expression is the same as finding all the identity transformations which are applicable to the signal-processing expression or one of its subexpressions. For example, to find the equivalent implementations of the matched filter bank used in the FSK-code detector, all the applicable identity transformations for the filter bank should be completed as should the transformations on the frequency-chip sequences used in the matched filters and the input sequence to the matched filters. In addition, once an alternate description is uncovered, all of the identity transformations which are applicable to this new description or to one of its subexpressions must also be applied. Thus, equivalent implementations of a signal-processing expression can be obtained in any of a variety of ways: a transformation can be applied to the original signal-processing expression itself; a subexpression of the original expression can be replaced by an equivalent implementation of the subexpression; or either of these approaches can be applied to one of the newly uncovered equivalent implementations of the signal-processing expression.

A graphical representation of the search process is presented in Figure 1. Using this representation, the problem of finding the equivalent forms of a signal-processing expression, without consideration of its subexpressions, is represented graphically as a net, as shown in Figure 1-a. One of the nodes of this net represents the starting signal-processing expression. The remaining nodes of the net represent equivalent implementations of the original expression. Some of these nodes are connected directly to the original node via simple transformation rules. These newly obtained nodes can themselves be used as seeds for other transformations. This search for additional nodes stops when no new nodes remain to be considered.

Any of the nodes of this net can also be viewed as a combination of subexpressions which can also be manipulated. A subexpression can be replaced by any of its equivalent implementations in an enclosing expression. Graphically, requesting the equivalent forms of a subexpression drops the problem down to another net and again tries to find connected nodes (Figure 1-b). The set of nodes found on this lower net is then projected back up into the original net by replacing the subexpression in the enclosing expression with its equivalent forms, as shown in Figure 1-c. This projection can generate new nodes in the original net. These new nodes are also used as seeds for finding additional nodes through recursive transformation and through expression decomposition.

The same strategies for finding the equivalent implementations of an expression are obviously also applicable to finding the equivalent implementations of any of the subexpressions. Thus, each of the searches for the equivalent forms of the subexpressions can also give rise to subsearches, using some even lower net. The downward progression stops when there are no more subexpressions which are signals or systems.

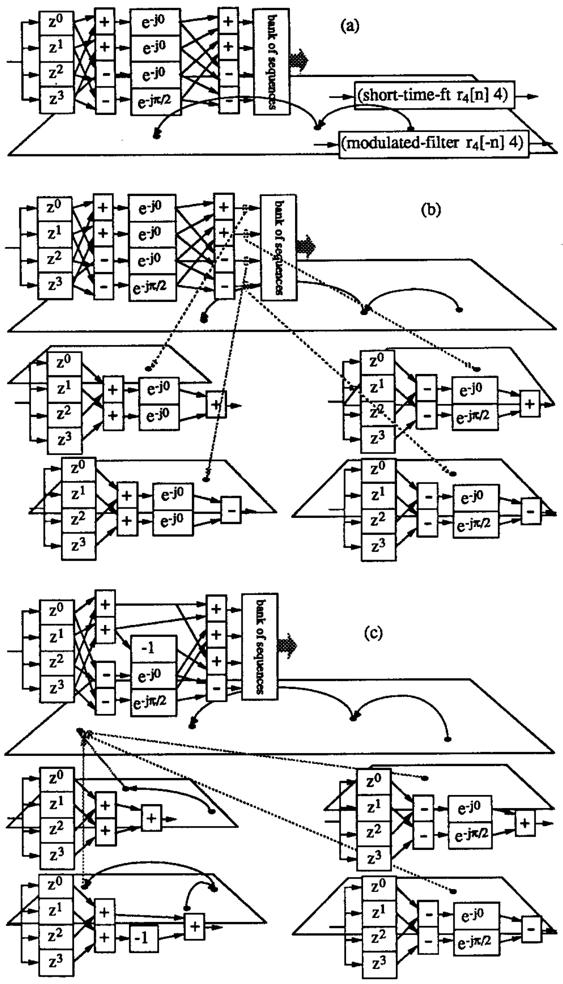

Figure 1: A net representation of the search for equivalent forms

IV. CONSTRAINED SEARCH FOR EQUIVALENT ALGORITHMS

As described, the search for equivalent implementations of a signal-processing expression must consider the equivalent implementations of the subexpressions as well as the complete expression itself. Since each of the subexpressions are independently manipulated and independently recombined to form new equivalent expressions, the size of the search space under consideration grows combinatorially with the number of subexpressions. To illustrate, consider the problem of implementing the full FSK-code detector described in Section II for sixteen channels. Five independent descriptions of a simple, finite-length convolution are embedded in ADE[4]. Thus, using these subexpressions as inputs into the incoherent summation, there will be $5^{16} \approx 10^{11}$ equivalent forms to consider.[1] Each of these implementations would then be reconsidered to see if any additional equivalent forms could be found by exploiting interactions between the implementations of the matched filters and the implementation of the incoherent processing. As illustrated by the projected size of the design space, some set of constraints must be imposed on the search process to avoid this combinatorial growth.

The internal regularity of signal-processing algorithms can be exploited to limit the search space. Signal processing algorithms are often described at different levels of detail: for example, the four-point, rectangularly-windowed, short-time Fourier transform (STFT) of a sequence can be described by **(output-of (stft (rectangular-window 4)) x)** or by the structure shown in Figure 2.[2] From the high-level description of the algorithm, the regularity in the low-level computational structure can often be asserted: from the high-level description given by the **stft** system, the underlying regularity inherent in Figure 2 can be asserted. By enforcing these internal correspondences in the low-level descriptions, the space of equivalent forms which is explored can be drastically reduced.

This approach to pruning the search is heuristic. However, the regularity of the computation suggests that efficient implementations will reflect the same regularity: if separate sections of an algorithm are similar, then the efficient implementations of these separate sections are likely to coincide.

To illustrate, consider the description of the STFT given in Figure 2. This implementation of the STFT is provided explicitly by one of the transformation rules included in the definition of **stft**. The regularity of this implementation is also explicitly pointed out by the rule. In particular, the similarity of the sequences feeding into the **bank-of-sequences** is pointed out using a "correspondence constraint". The effect of applying a correspondance constraint on a set of subexpressions is that the rearrangement of these subexpressions is constrained to coincide: that is, if some transformation is applied to one of the subexpressions, it will also be applied to the remaining subexpressions. In addition, the subexpressions feeding into each of the **add/subtract** systems are similar: the first addend into the k'th system is similar to the second addend into the k'th system. Thus, more correspondence constraints are placed on the inputs into each of the **add/subtract** systems. Through these *correspondence constraints*, the manipulation of the corresponding subexpressions of the STFT is constrained to occur in synchrony, resulting in the manipulation of $S_k[n]$, $R_{l,i}[n]$, $P_{l,i}[n]$ and $x[n+2m+i]$ where $S_k[n]$, $R_{l,i}[n]$ and $P_{l,i}[n]$ are

[1]None of these implementations exploit the special structure of the modulated filter bank. The actual number of equivalent implementations which have to be considered is more than 10^{19}.

[2]The structure shown in Figure 2 is the STFT using an FFT structure. The familiar butterflies are not immediately obvious since a tree graph is used, to avoid the implication that multiple uses of intermediate results will always use coincident implementations.

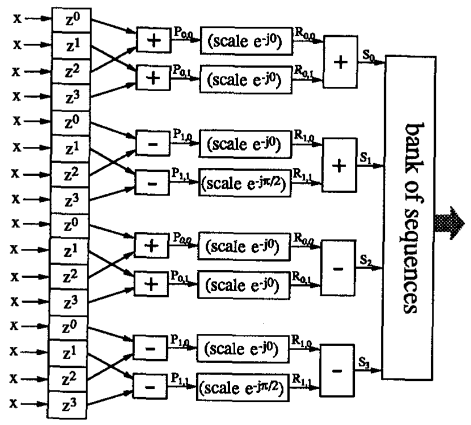

Figure 2: An example of a "low-level" computational description with a highly regular internal structure

as labelled in Figure 2 and $x[n+2m+i]$ are shifted versions of the input sequence. By enforcing these constraints, the number of independently manipulated subexpressions is reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(\log N)$, where N is the size of the STFT.

Once a correspondence constraint is imposed, that constraint should propagate inward until a mismatch is detected, affecting the manipulations of all the intervening expressions. Recursive search, in which newly uncovered equivalent forms act as seeds for recursive requests for additional equivalent forms, can be used without modification. However, subexpression manipulation must be modified for constrained equivalent-form searches. In particular, if a correspondence constraint is imposed on a set of subexpressions, the equivalent forms of the subexpressions can not be generated separately. Instead, the equivalent forms of the subexpressions are found by manipulating the corresponding subexpressions identically. The alternate implementations which result from these manipulations of the subexpressions are then used to replace the subexpressions in the enclosing expression.

To illustrate this process, consider the task of finding the constrained equivalent forms of the four-point STFT structure shown in Figure 2. Some of these equivalent forms will be found by manipulating the input sequences to the system **bank-of-sequences** in synchrony: thus, the set of sequences labelled $S_k[n]$, $k = 1, \ldots, 4$ are considered simultaneously. Under this parallel manipulation, each of the equivalent-form transformations which could be applicable to the individual expressions is considered. Only the equivalent-form transformations which are applicable to *all* these parallel sequences will be completed and these transformations will be done on all of the sequences, simultaneously. The point of manipulation progresses inward by considering the inputs to the constrained expressions. Thus, the set of sequences labelled $R_{l,i}[n]$, $P_{l,i}[n]$ and $x[n+2m+i]$ are also examined in turn.

The equivalent forms found through the inner constrained manipulations propagate outward by combining the discovered equivalent forms with the operators which were dropped.

As before, each new set of parallel equivalent expressions is then used to as a seed to find other constrained equivalent forms. This recursive search continues until no new, constrained equivalent forms are found. This outward progress of constrained equivalent forms continues until all the equivalent forms of the original STFT are found.

V. ADE FOR DESIGN OF THE FSK-CODE DETECTOR

The FSK-code detector separates naturally into three subproblems: the recovery of the in-phase and quadrature samples of the sonar return; the modulated filter bank for matched-filter detection of the individual frequency chips; and the incoherent combination of the filter-bank outputs to create the detectors for the full FSK code set. These subsections describe the transfer characteristics which are desired for the the detector but they need not describe the actual algorithm which is used to compute the output signals. Instead, any input/output equivalent implementation can be used in place of these descriptions. The set of all input/output equivalent implementations represents a design space which can be explored using an algorithm design environment, like ADE.

An impressive demonstration of the potential of constrained, automatic algorithm manipulation is provided by the design of the matched filter bank used in the FSK-code detector. In particular, when the problem of finding efficient implementations of the matched filter bank was submitted to ADE, the implementation shown in Figure 3 was uncovered. This implementation, which will be referred to as a "pruned FFT structure," has the same underlying structure as the general FFT implementation of the modulated filter bank. The difference lies in the number of butterflies that are computed at each stage: the pruned FFT structure has only one butterfly in the first stage, two in the second, four in the third and so on, while the general FFT structure has $\frac{N}{2}$ butterflies in each stage.

It is interesting to note that, with the pruned FFT, the order of the computational complexity is actually reduced as well as the number of computations themselves. The order is reduced from $\mathcal{O}(N^2)$ for the direct-form implementation or from $\mathcal{O}(N \log N)$ for the general FFT implementation to $\mathcal{O}(N)$ for the pruned FFT implementation.

The design of this matched filter bank indicates the potential of constrained algorithm manipulation: the pruned FFT structure is an innovative implementation for a modulated filter bank or a STFT. Furthermore, due to the high internal branching factor of this algorithm, unconstrained manipulation is untenable. Using simple combinatoric analysis, the number of equivalent forms which would be found by unconstrained manipulations on these structures can be estimated: this number is more than 10^{19}.

VI. CONTRIBUTIONS

This article has focused on the process by which ADE finds equivalent implementations of a signal processing system. Myers[3] was the first to consider the symbolic manipulation to support and expedite the design of signal-processing algorithms. One of the difficulties with the unconstrained manipulation of signal-processing expressions used in [3] is the combinatorial growth of the design space, due to the independent manipulation of subexpressions. ADE introduces the idea of a regularity constraint to limit the combinatoric growth of the search for efficient implementations. Regularity constraints exploit the internal regularity of signal-processing algorithms to limit the size of the explored search space. This regularity in the low-level signal-processing descriptions is pointed out using information provided by the high-level description of the same operation. Without these constraints, many FFT-based and polyphase-based algorithms are beyond the scope of reasonable consideration, due to the combinatoric expansion of these design spaces.

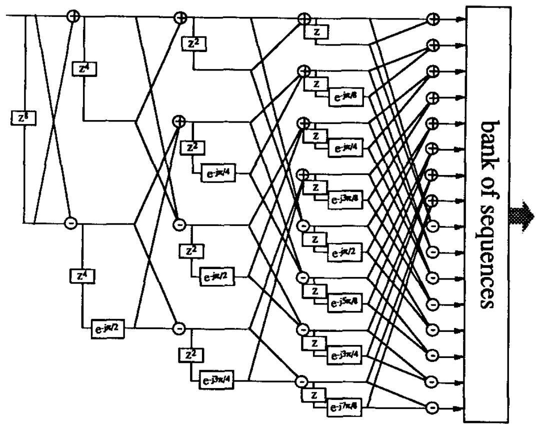

Figure 3: An innovative implementation, uncovered by ADE, of the matched filter banks using a rectangular frequency-chip window

[1] Kopec, G., *The Representation of Discrete-Time Signals and Systems in Programs*, PhD thesis, MIT, 1980.

[2] Dove, W., Myers, C., and Milios, E., *An Object-Oriented Signal Processing Environment: The Knowledge-Based Signal Processing Package*, R.L.E. Technical Report 502, MIT, 1984.

[3] Myers, C., *Signal Representation for Symbolic and Numerical Processing*, PhD thesis, MIT, 1986.

[4] Covell, M., *An Algorithm Design Environment for Signal Processing*, PhD thesis, MIT, 1989.

[5] Jaffe, J., and Richardson, J., "A Code-Division Multiple Beam Imaging System," *OCEANS '89*, 1989.