The Wavelet Packet Transform[1]
Extending the wavelet transform

Mac A. Cody[2]

The wavelet transform enables analysis of data at multiple levels of resolution (also known as "scale"). In addition, transient events in the data are preserved by the analysis. When the wavelet transform (WT) is applied to a signal in the time domain, the result is a two-dimensional, time-scale domain analysis of the signal. The transform has proven useful for the compression and analysis of signals and images.

The fast wavelet transform (FWT) is an efficient implementation of the discrete wavelet transform (DWT). The DWT is the WT as applied to a regularly sampled data sequence. The transform of the data exhibits discrete steps in time on one axis, and discrete steps of resolution on the another. The algorithm and a C language implementation of the FWT were presented in my article "The Fast Wavelet Transform" (*DDJ*, April 1992).

As demonstrated in that article, the superiority of the DWT over the discrete Fourier transform (DFT) is in the DWT's simultaneous localization of frequency and time, something that DFTs can't do. As a trade-off, the frequency divisions in the DWT are not in integral steps. Instead, the divisions are in octave bands. Each level of the transform represents a frequency range half as wide as the level above it and twice as wide as that of the level below it; see Figure 1(a).

Conversely, the time scale on each level is twice that of the level below it and half that of the level above it; see Figure 1(b). This characteristic of the DWT poses problems when attempting to localize higher frequencies. Discrimination of frequency is sacrificed for time localization at the higher levels in the transform.

It turns out that the DWT is actually a subset of a far more versatile transform, the wavelet packet transform (WPT). Developed by Dr. Ronald A. Coifman of Yale University, the WPT generalizes the time-frequency analysis of the wavelet transform. It yields a family of orthonormal transform bases of which the wavelet transform basis is but one member.

In this article is I'll develop the wavelet packet transform algorithm from it's roots in the wavelet transform algorithm. After that, I'll present the C code to implement the algorithm.

From a Humble Root a Tree Shall Grow

In the fast wavelet transform algorithm, the sampled data set is passed through the scaling and wavelet filters (the convolution operation). They are, respectively, low-pass and high-pass filters with complementary bandwidths, also known as a quadriture mirror filter (QMF) pair. The outputs of both filters are decimated (desampled) by a factor of two. The high-pass filtered data set is the wavelet transform detail coefficients at that level of scale of the transform. The low-pass filtered data set is the approximation coefficients at that level of scale. Due to the decimation, both sets of coefficients have half as many elements as the original data set.

The approximation coefficients can now be used as the sampled data input for another pair of wavelet filters, identical to the first pair, generating another set of detail and approximation coefficients at the next lower level of scale. This process can continue until the limit for the unit interval is reached. For example, if it is desired that the transform have six levels (5 through 0), then the unit interval must be 64 (2^6) samples long. The data set can be of any length as long as it has an integral number of unit intervals. The resulting algorithm is the forward fast wavelet transform tree algorithm; see Figure 2(a).

You can turn the tree algorithm on end, with the initial data input at the top and the detail and approximation coefficients fanning out towards the bottom. The fast wavelet transform algorithm can now be viewed as a partial graph of a binary tree (the significance of this will be seen shortly); see Figure 2(b). The flow of the algorithm moves down and to the left, forming new levels of the transform from the approximation coefficients at higher levels. The detail "branches" are not used for further calculations.

Observe that the wavelet transform operation can be stopped at any level while working down the tree. The resulting "partial" transform is still a valid orthonormal transform. For example, if the unit interval for a

[1] Originally appeared in Dr. Dobb's Journal, April, 1994. Copyright ©1994, M&T Publishing Inc., Miller Freeman Publishing, Inc.; preprint created by permission. Corrections made by the author.

[2] Mac is the owner of Mac A. Cody Associates, Richardson, TX. He has published Tsunami Plus, a wavelet software library written in C. He can be at contacted by phone at 214.437.5761 and on the Internet at mcody@dfw.net.

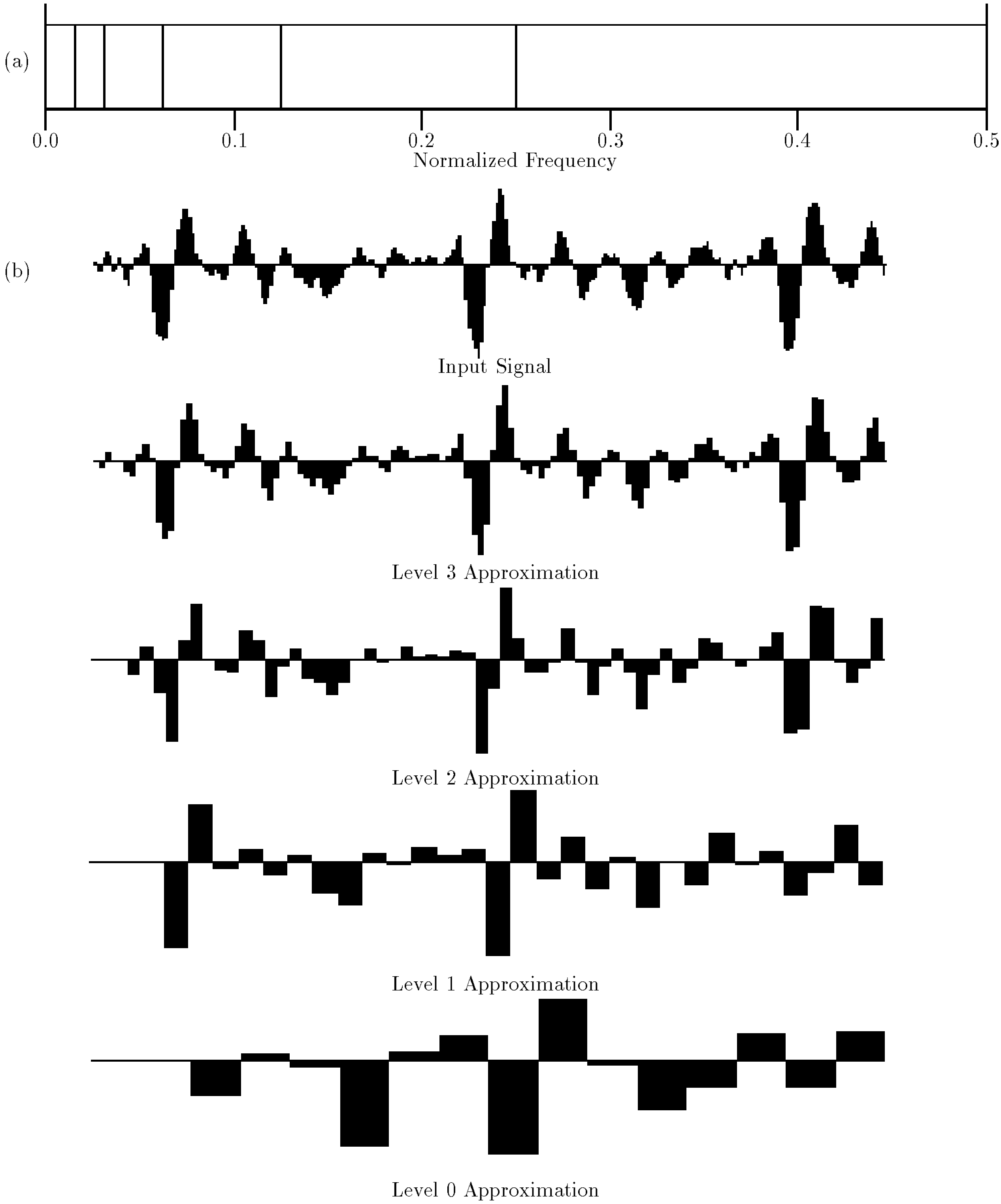
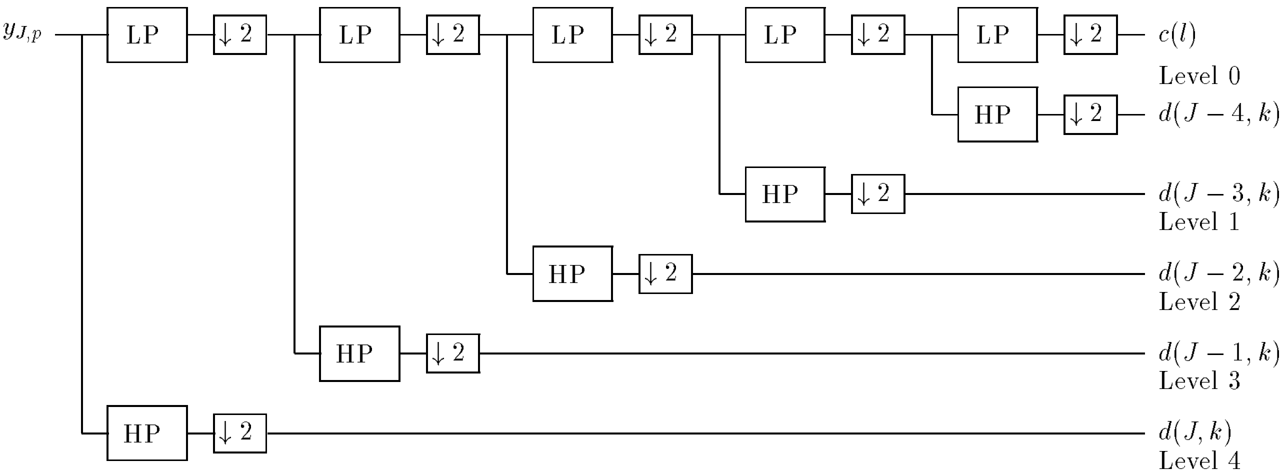
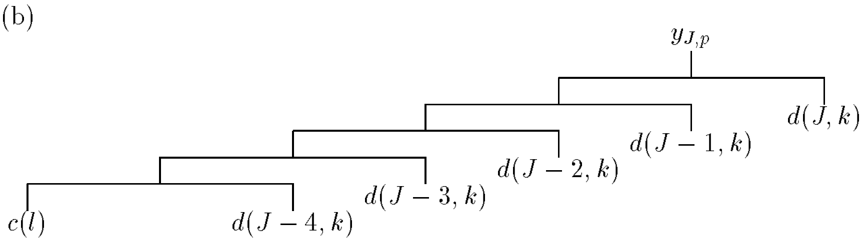


Figure 1: (a) The Discrete Wavelet Transform (DWT) divides the spectrum of the sampled data into octave bands; (b) the resolution at each level of the DWT is half that of the level above it and twice that of the level below. At lower levels, time resolution is sacrificed for frequency localization.

(a)



(b)



		J		
5	4	3	2	1
4	3	2	1	0
3	2	1	0	-
2	1	0	-	-
1	0	-	-	-
0	-	-	-	-

Figure 2: The tree or pyramid algorithm of the forward fast wavelet transform (a) can be viewed as a partial graph of a binary tree (b). For a particular unit interval (2^J samples), a maximum of J levels of transform data can be formed.

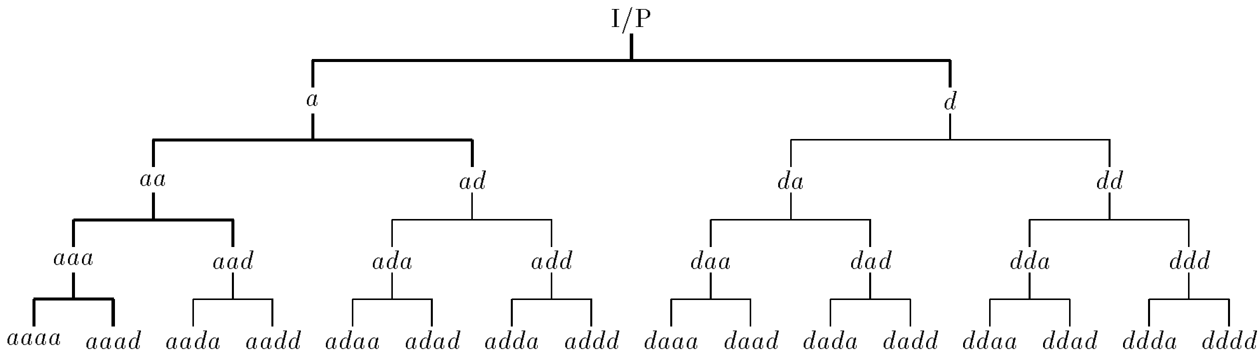


Figure 3: The wavelet packet transform viewed as a complete binary graph. Each a and d in each sequence represents the filtering operations performed to yield the particular subspace of the original signal. The bold lines represent the disjoint cover known as the "wavelet basis."

data set is 32 points (2^5), the corresponding transform would have five levels (4 through 0). If the transform operation were stopped at level 2, the transform would have only three levels, but the approximation and detail coefficients of the transform would correspond exactly to a wavelet transform with a unit interval of 8 (2^3) samples, see Figure 2(b).

The implication of this observation is that the QMF pair is an orthonormal transform kernel, like the butterfly operation is the kernel of the FFT. As long as the filters are designed to be orthonormal wavelet filters and the original data set meets the unit-interval requirement described above, repeated applications of the kernel will always yield an orthonormal transform.

Now, the set of detail and approximation coefficients at each level of the transform forms a pair of subspaces of the approximation coefficients of the next higher level of scale and, ultimately, the original data set. The subspaces created by the wavelet transform roughly correspond to the frequency subbands shown in Figure 1(a). These subspaces form a disjoint cover of the frequency space of the original data set. In other words, the subspaces have no elements in common and the union of the frequency subbands span the frequency range of the original data set.

What Dr. Coifman proposed is that any set of subspaces which are a disjoint cover of the original data set is an orthonormal basis. The wavelet transform basis is then but one of a family of orthonormal bases with different subband intervals. As with the wavelet transform basis, each disjoint cover roughly corresponds to a covering of the frequency space of the original signal. Coifman dubbed this family a "wavelet packet library." The various orthonormal bases are formed by arbitrary applications of the orthonormal transform kernel upon the detail coefficients as well as the approximation coefficients of higher transform levels.

The application of the transform kernel to both the detail and approximation coefficients results in an expansion of the structure of the fast wavelet transform tree algorithm. The tree algorithm for the wavelet packet transform can be represented as a full binary tree; see Figure 3. The a and d symbols at each node indicate the order of orthonormal transform kernel filter operations performed which yield each particular subspace of the original data set. Each node in the transform tree is also representative of a particular wavelet packet. The transform coefficients computed at each node are a correlation of the original data set and a waveform function representing the wavelet packet.

For example, the sequence $aaad$ in Figure 3 represents four operations of the orthonormal transform kernel representing one of 48 possible wavelet packets. (The combination of all possible translations in time and dilation in scale for the wavelet packets is $J * 2^J$; in this instance J equals 4.). The first three represent low-pass filter/decimation operations performed by the transform kernel. The fourth represents a high-pass filter/decimation operation performed by the transform kernel. This subspace should be recognizable as exactly the level 0 detail coefficients of the wavelet transform. The operations of the orthonormal transform kernel correspond to the wavelet function of the wavelet transform. Likewise, the wavelet packet represented by $aaaa$ is the scaling function of the wavelet transform.

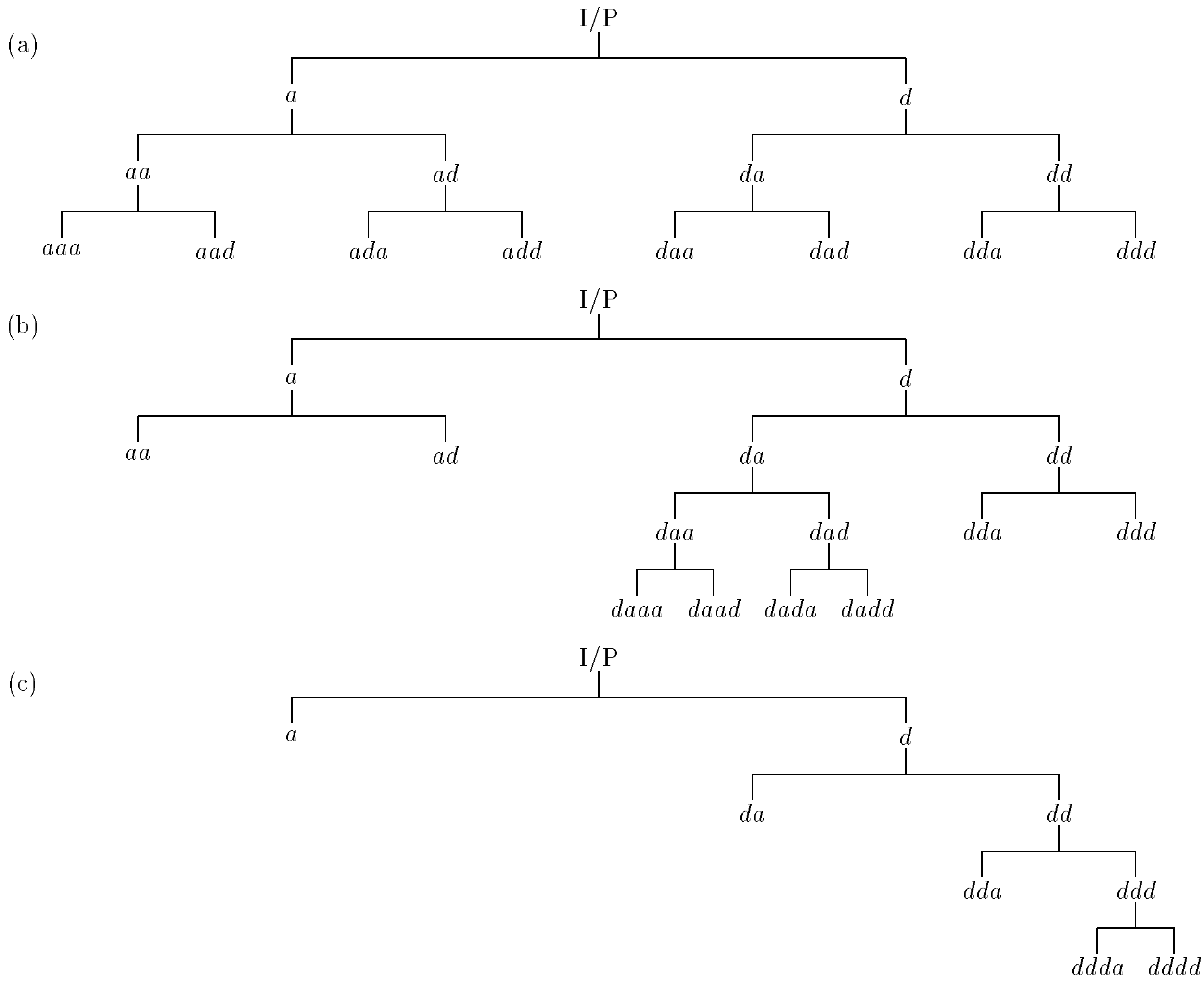


Figure 4: Different disjoint covers formed from the WPT binary tree (Figure 3) yield different wavelet packet bases. (a) A subband basis; (b) an othonormal basis subset; (c) a basis which is the opposite of thw wavelet basis (better frequency localization at higher frequencies).

Packets, Graphs, and Bases

The wavelet transform basis is actually a subset of a family of bases formed by the wavelet packet transform. The heavy lines in Figure 3 indicate the graph forming the wavelet basis. Note that the wavelet basis consists of the subspaces d, ad, aad, $aaad$, and $aaaa$. The sequences a, aa, and aaa are intermediate steps leading to the generation of the subspaces of the wavelet basis at the lower levels. Since the orthonormal transform kernel can be arbitrarily applied to either approximation or detail branches on the tree, $J * 2^J$ graphs representing different orthonormal bases can be created; see Figure 4.

The variety of orthonormal bases which can be formed by the WPT algorithm, coupled with the infinite number of wavelet and scaling functions which can be created, yields a very flexible analysis tool. The flexibility of WPT versus the FWT can be compared to that of having a complete set of sockets for a ratchet rather than a single socket to attach to it. The ratchet (algorithm) works the same regardless of the socket (basis) that is chosen. The flexibility of the tool is in choosing the appropriate socket (basis) for the particular nut (problem). The choice of wavelet and scaling functions is then analogous to selecting from English, metric, or Torx socket sets for use with the ratchet. The WPT allows tailoring of the wavelet analysis to selectively localize spectral bands in the input data as well as to correlate the signal to the wavelet. Not only can the best wavelet be chosen to analyze a particular signal but the best orthonormal basis can as well. In signal-processing terminology, the various bases of the wavelet packet transform can be used as arbitrary adaptive tree-structured filter banks.

Piece-wise Convolutions and Traversing the Tree

The implementation of the WPT is itself a generalization of the FWT routine presented in my previous article. As with the FWT, the kernel operations are the decimating and interpolating convolutions, as presented in Listing One (page 101). The convolutions performed are actually piece-wise convolutions, due to the discrete nature of the data. The routine *DualConvDec2* replaces the routines *ConvolveDec2* and *Dotp* in the FWT code. The routine *DualConvInt2Sum* replaces *ConvolveInt2*, *DotpOdd*, and *DotpEven* in the inverse FWT code.

Both convolution routines are designed to operate upon aperiodic data of finite length. The data does not represent an infinitely repeating pattern and is assumed to be surrounded by zero-valued data; see Figure 5. To support this data model, each data array is appended with additional data storage equal to the length of the wavelet filter minus one (the shaded elements). The extra data is filled with the terminating convolution values as the wavelet filters "slides off" the end of the data set. Extending the convolution data by this additional amount during the decomposition process of the forward transform ensures perfect reconstruction of the original input data by the inverse transform. *DualConvDec2* also uses partial dot products at both ends of the convolution to simulate the implied zero-valued data (the dotted lines) outside of the data array. The dotted lines on the filter elements indicate coefficients not used in the partial dot product calculations. *DualConvInt2Sum* does not need to do this, since all information necessary for reconstruction is contained within the extended data arrays. Note that *DualConvInt2Sum* performs only the odd-valued dot product at the beginning of the convolution for the initial reconstruction data point.

The WPT data is stored in the structure *WPTstruct*, defined in Figure 6(a). The structure contains storage for the number of levels in the transform, the length of the original, untransformed data array, and a pointer to a two-dimensional matrix of data arrays. The size of the matrix in dependent upon three factors. These are the number of levels in the transform, the length of the original data array, and the length of the transform filters. The length of the data array is itself affected by the length of the transform filters; see Figure 6(b).

Figure 6(c) shows the matrix structure as the wavelet packet binary tree. The data-array pointers are allocated memory from the heap as necessary to form the disjoint cover for the chosen orthonormal basis. Those array pointers not required for the disjoint cover are set to zero. The example shown in the figure represents a three-level wavelet basis for an input of 40 data points with transform filters containing six coefficients.

The routine *DualConvDec2* is used by *AWPT*, the forward wavelet packet transform routine; see Listing Two (page 101). *APWT* accepts pointers to the input data array, the WPT data structure, and the transform filter arrays, and the length of the filters. The transform routine works down the levels of the binary tree performing convolutions on the data arrays. Each level of the binary tree is traversed by taking adjacent pairs of array pointers as the destination nodes for the low-pass and high-pass convolutions. If the array pointer for the low-pass convolution is zero, the convolutions are not perform since the destinations are not part of the current disjoint cover.

At the highest level (where i equals 0) the input data array is the source for the convolutions. On each subsequent level, the sources are the arrays on the previous level. The appropriate array in the binary tree is determined by the dividing the current j index by 2. After each convolution operation, the data length is divided by two, in order to keep track of the affect of the decimation operation during the convolutions.

DualConvInt2Sum is used by *IAWPT*, the inverse wavelet packet transform routine. *IAWPT* accepts pointers to the source WPT data structure, the output data array, and the transform filter arrays, and the length of the filters. The inverse transform routine works up the levels of the binary tree performing convolutions on the data arrays and reconstructing the higher resolution data on each level. Each level of the binary tree is traversed in the same fashion as was *AFWT*. The destination arrays are on the next-higher level of the matrix, and they are selected by dividing the j index by 2. At the highest level (i equals 0), the destination array is the output data array. After each convolution operation, the data length is doubled to keep track of the effect of the interpolation operation during the convolutions.

The code listings presented here are written in ANSI C and have been tested with Borland Turbo C

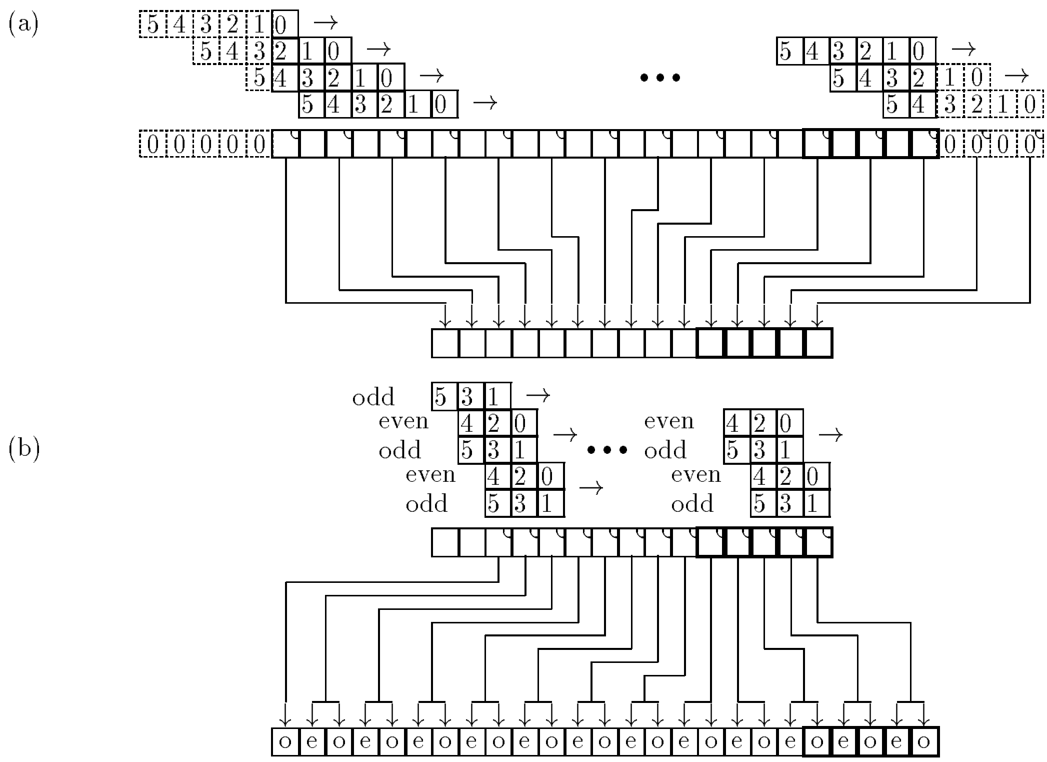
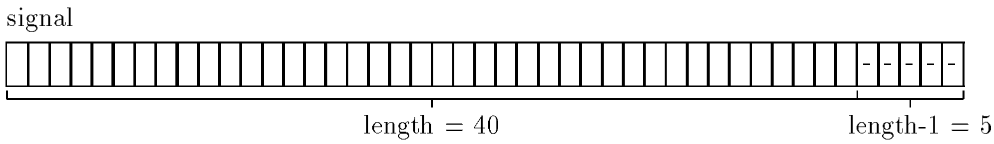


Figure 5: (a) The convolution operation in DualConvDec2 employs partial dot products at both ends of the data array to simulate zero-valued data surrounding it; (b) the convolution operation in DualConvInt2Sum performs dot products with alernating odd and even filter components to simulate interpolation. The additional convolution data generated by DualConvDec2 is used to accomplish perfect reconstruction. Marked data elements indicate starting points of dot-product calculations.

(a)

```
typedef struct
{
   short levels, length;
   REAL_TYPE ***data;
} WPTstruct;
```

(b)



(c)

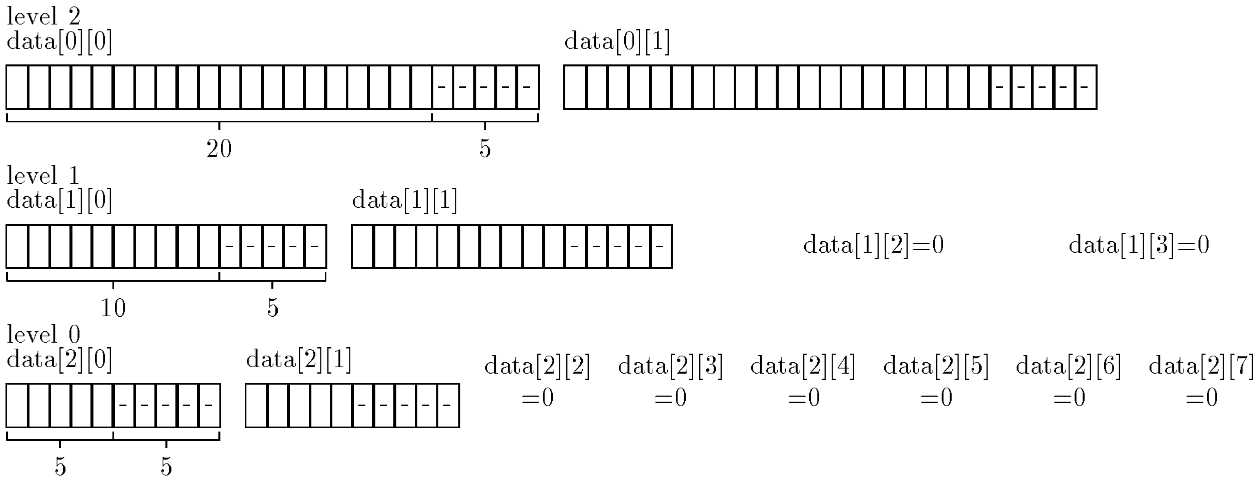


Figure 6: (a) The data structure for the WPT. The REAL_TYPE declaration can be defined either as float or double; (b) the storage structure for the original signal consists of the data, plus appended storage egual to the filter length, minus 1; (c) the data component of WPTstruct is a two-dimensional matrix of data arrays forming the wavelet packet binary tree. Data-array pointers set to 0 indicate parts of the tree that aren't part of the disjoint cover (the wavelet basis).

2.0. They should compile on any compiler that is compliant with ANSI C. I've also written a wavelet packet transform demonstration program which is available electronically; see "Availability" on page 3[3]. The electronic version includes the demo program, sample data files, support drivers, and documentation.

Conclusion

The wavelet packet transform generalizes the discrete wavelet transform and provides a more flexible tool for the time-scale analysis of data. All of the advantages of the fast wavelet transform are retained since the wavelet basis is in the repertoire of bases available with the wavelet packet transform. Given this, the wavelet packet transform may eventually become a standard tool in signal processing.

References

Cody, Mac A. "The Fast Wavelet Transform." *Dr. Dobb's Journal* (April, 1992).

Coifman, Ronald R., Yves Meyer, and Victor Wickerhauser. *Wavelet Analysis and Signal Processing.* New Haven, CT; Yale University, 1991, preprint.

[3] The listings and programs described in this article are available via anonymous ftp from ftp.mv.com in the /pub/ddj/1994.04 directory. The file to download is wave3.zip.