Using Ordered Binary-Decision Diagrams for Compressing Images and Image Sequences

Mike Starkey and Randy Bryant

January 1995
CMU-CS-95-105

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

The Ordered Binary-Decision Diagram (OBDD) has been used to reduce the amount of space and computation required for verifying digital circuits by removing redundant copies of subfunctions. Similarly, image compression algorithms attempt to reduce their space requirements by finding replicated patterns or features in images. OBDDs would therefore appear to be a good candidate as a data structure for representing images. We show how images can be encoded using Ordered Binary-Decision Diagrams and compare our results to quadtrees. We also show how using this method extends naturally to compressing sequences of related images such those that comprise movies.

This research is sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant F33615-93-1-1330.

Keywords: binary decision diagrams, image compression

1 Introduction

The introduction of Ordered Binary-Decision Diagrams [2] to the field of VLSI circuit verification has a large impact on the size and complexity of circuits that could be verified. The boolean functions for digital circuits were found to contain many equivalent subfunctions. By building a data structure such that these subfunctions could be shared in some way not only reduces the space requirements of storing these circuits but also reduces the computation required when manipulations are performed on the reduced representation.

The OBDD data structure is similar to the binary analog of the quadtree data structure [6]. Both of these data structures attempt to reduce the amount of storage required to store data with replicated patterns (or subfunctions). The traditional quadtree has nodes that reference each of the four quadrants when subdividing the area into equal parts. Each of these quadrants is subdivided into four parts and so on until the pixel values are reached. If a quadtree node references four pixel values of the same color then it is removed and the pixel value is promoted to the parent node. For images with blocks of the same color, this can produce a reduction in the number of nodes required to represent the image. The binary analog to the quadtree, the bintree, has binary nodes which reference the two halves of the image. The division into halves is alternated between dividing in the x direction and the y direction. Nodes are eliminated if the two children have the same pixel value.

The Ordered Binary-Decision Diagram also has binary nodes. In addition, all of the nodes can be shared in the entire OBDD. Therefore, any pattern that occurs multiple times will have one instance in the OBDD that is referenced by nodes that have that pattern as one of their children. Also, a node is eliminated if the two children are identical regardless of whether they are pixel values or sub-OBDDs representing more complex patterns.

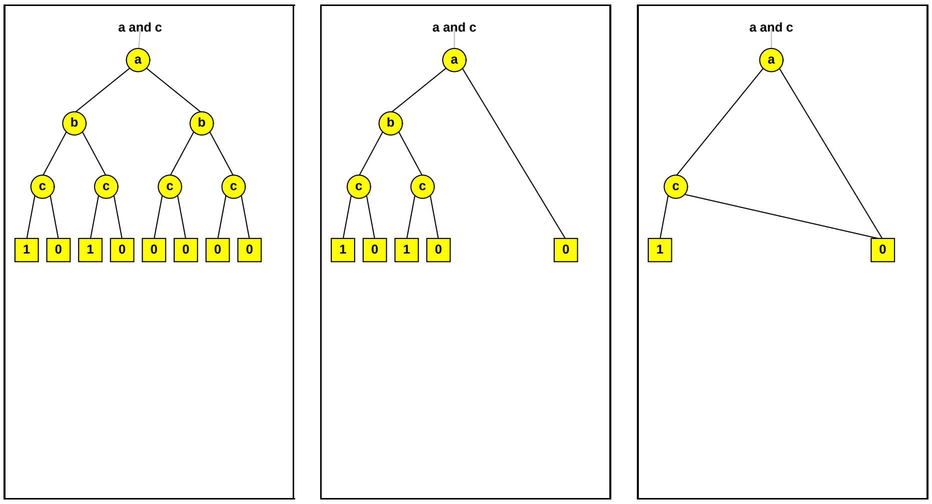
We describe how we build Ordered Binary-Decision Diagrams from images. The resulting OBDDs are compared in terms of the number of nodes to the binary version of the quadtree, the bintree [4, 6, 7]. We compare the number of nodes required for a set of examples. We also show the worst cases for the size of each data structure given images of certain sizes with varying numbers of colors. Our comparison continues with calculating the binary file sizes required to store images for each representation.

We then proceed to describe how the Ordered Binary-Decision Diagram lends itself to be used for storing sequences of related images such as found in movies.

Our results show that storing images as Ordered Binary-Decision Diagrams results in a large savings in memory requirements at only a slight increased cost in file storage. Images represented as bintrees require 4 times as many nodes as required by Ordered Binary-Decision Diagrams. The size of files required to store images as OBDDs is only 25% larger than the files required for storing bintrees.

2 Images as Ordered Binary-Decision Diagrams

Ordered Binary-Decision Diagrams are a natural way of describing binary circuits. Each node represents the value of a binary variable. If the value is *1* then one branch is taken, a value of *0* causes the other branch to be taken. There are no other possible values so by knowing the binary value of the variables along the path, the resulting binary value can be found. For example, suppose we have three variables **a**, **b** and **c** and we want to represent **a and c**. If **a** has a value of *0* then the result is *0* regardless of the values of **b** or **c**. If **a** is *1* then the result depends on the value of **c**. If **c** is *1* then **a and c** is *1* otherwise **a and c** is *0*. Figure 1a shows the complete binary tree required to represent this binary function. In this figure a variable of value *1* indicates that the left branch should be traversed and the value of *0* the right branch. The bintree representation of this circuit would remove the nodes that reference the same leaf value. The bintree representation is shown in figure 1b. The OBDD representation not only removes the nodes with identical leaf values but also removes the nodes that have identical children. Figure 1c shows the resulting



a) b) c)

Figure 1: Representations of the binary function **a and c** as a binary tree (a), binary quadtree (bintree) (b) and OBDD (c). The left and right branches correspond to values of 1 and 0, respectively.

OBDD for this simple binary function, where the node labeled **b** has been eliminated.

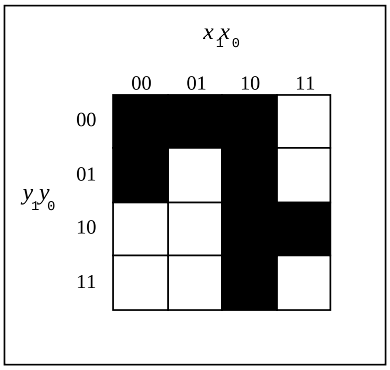
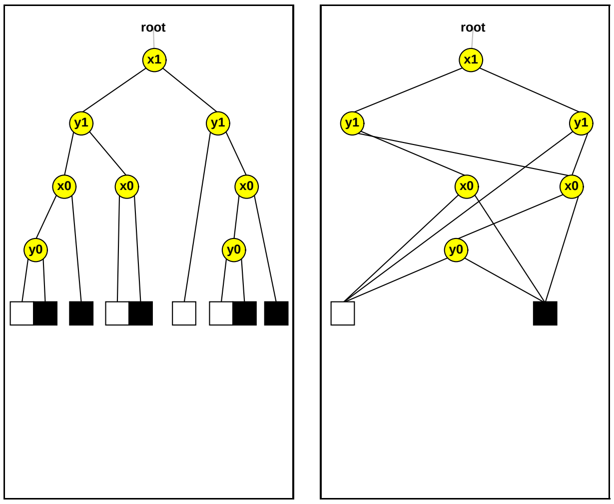


Figure 2: A Simple black and white image and the variables and leaf values required to form a OBDD

For images there is no simple function of a set of variables that describes an arbitrary image. Therefore, we make the image a function of its x and y coordinates. To keep with the OBDD convention of taking left or right branches depending on the bit value of the variable, we use the binary representation of the coordinates to build the OBDD. We use big endian bit numbering to represent the coordinate values and we alternate between x and y values. Figure 2a shows a sample black and white image and the coordinates of the pixels. To build the OBDD, we choose the ordering of variables for this 4 by 4 image to be x_1, y_1, x_0, y_0. The resulting OBDD is shown in figure 3b and for comparison the bintree representation using the same ordering of nodes is shown in figure 3a.

This example shows how a OBDD representing a black and white image can be created. This conforms naturally to the use of OBDDs for circuit verification where leaves have binary values of *0* or *1*. OBDDs have been extended to allow a range of leaf values [1, 3] and we use that extension to represent images with many colors or grey levels.

This method for dividing up the image works best for images that are 2^n by 2^n, 2^n by 2^{n+1} or 2^{n+1} by 2^n. However, a OBDD or bintree can be constructed from any image by finding the largest rectangle $\lceil \log_2 width \rceil$ by $\lceil \log_2 height \rceil$ and assigning a "background" value for any points external to the image.



a) b)

Figure 3: The bintree (a) and OBDD (b) representations of the image in figure 2

3 Ordered Binary-Decision Diagrams versus Bintrees

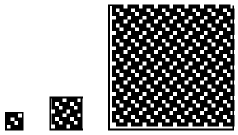
There are many ways to compare the efficiency of data structures. There are usually tradeoffs between space and speed for different operations. We concentrate primarily on the saving of space. There are numerous operations that are performed on quadtrees that we believe will perform as well if not better on OBDDs. In circuit verification OBDDs increase performance of operations by caching results. If an operation is performed twice on an identical subpart of the OBDD the cached result will be used the second time.

We will compare the space required by OBDDs and bintrees in two ways. The first is to compare the number of nodes required. Figures 2b and 2c have already provided a comparison of the number of nodes required for some simple cases. We believe that a reduced number of nodes is important since the data structures must be present in memory to be manipulated. The second comparison is of file sizes. This compares the number of bits required to store each of the representations and permit the internal representation to be reconstructed.

3.1 Comparison of Nodes

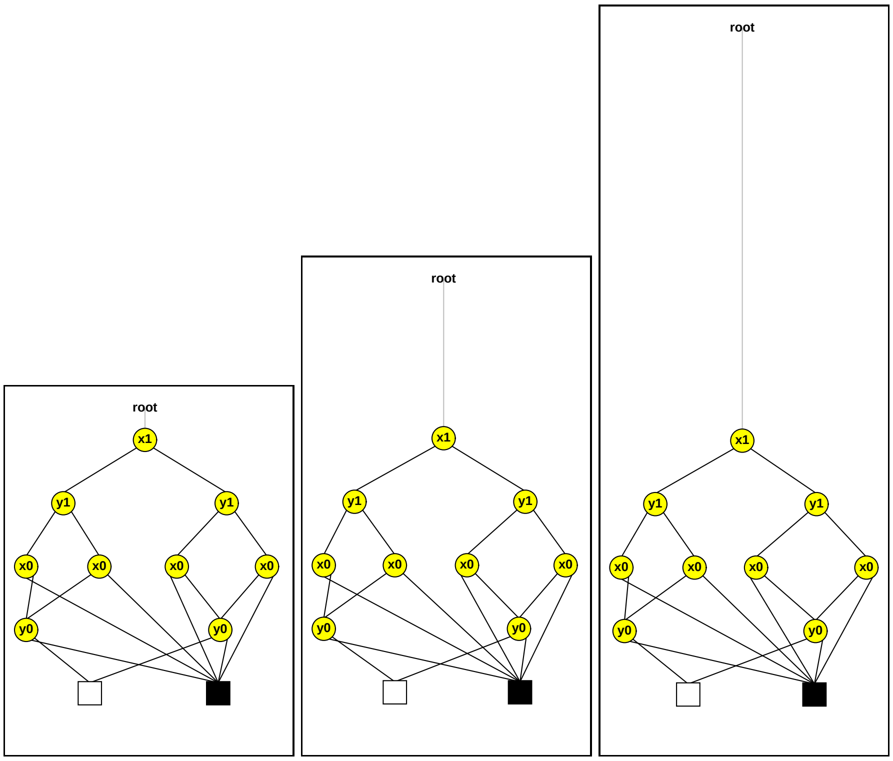
The number of nodes required for OBDDs and bintrees is dependent on a number of factors. Image size plays a role in how many variables are required. However, a small square tile with dimensions 2^n by 2^n replicated 2^m by 2^m times can produce identical instances of the data structure. The number of colors is also important to the number of nodes. With only 2 colors it is more likely that patterns will be replicated thereby reducing the sizes of the data structures. And finally, the amount of redundancy is the most important factor determining the size of the data structure and hence the number of nodes.

Figure 4 shows a number of images formed from the base tile in figure 4a. The corresponding OBDDs are shown in figure 5. Since the basic tile is identical in all of the images, the OBDD that is constructed is that of the basic tile. For demonstration we have shown the root being higher the larger the image but the OBDDs are identical. The bintree representation that corresponds to figure 4b appears in figure 6. This can be compared to the OBDD in figure 5b. These figures show quite clearly that OBDDs can produce a much smaller data structure.



a) b) c)

Figure 4: Repeating pattern in a) 4x4 pixels, b) 8x8 pixels, c) 32x32 pixels



a) b) c)

Figure 5: OBDD representations corresponding to the images in figure 4

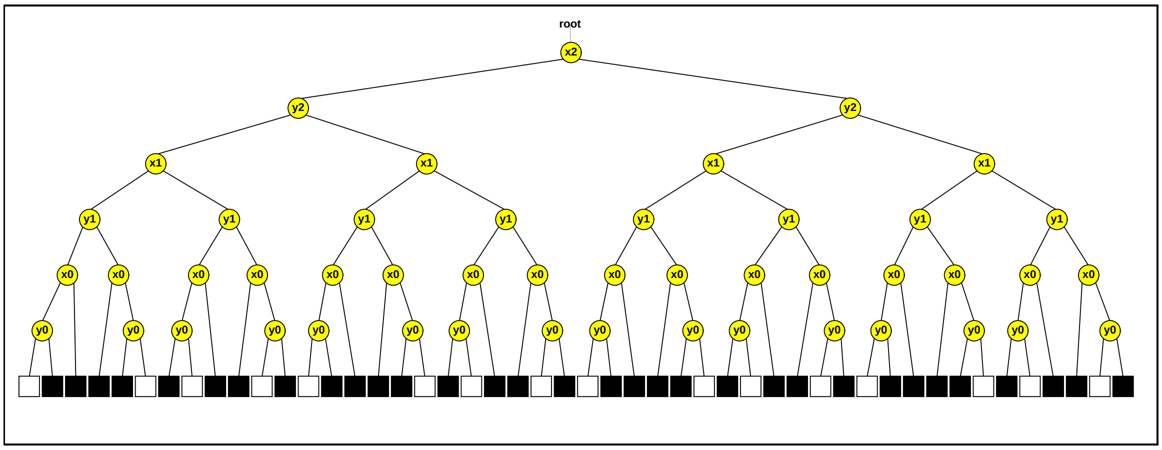


Figure 6: Bintree representation of the 8x8 pixel image of the repeating pattern in figure 4b

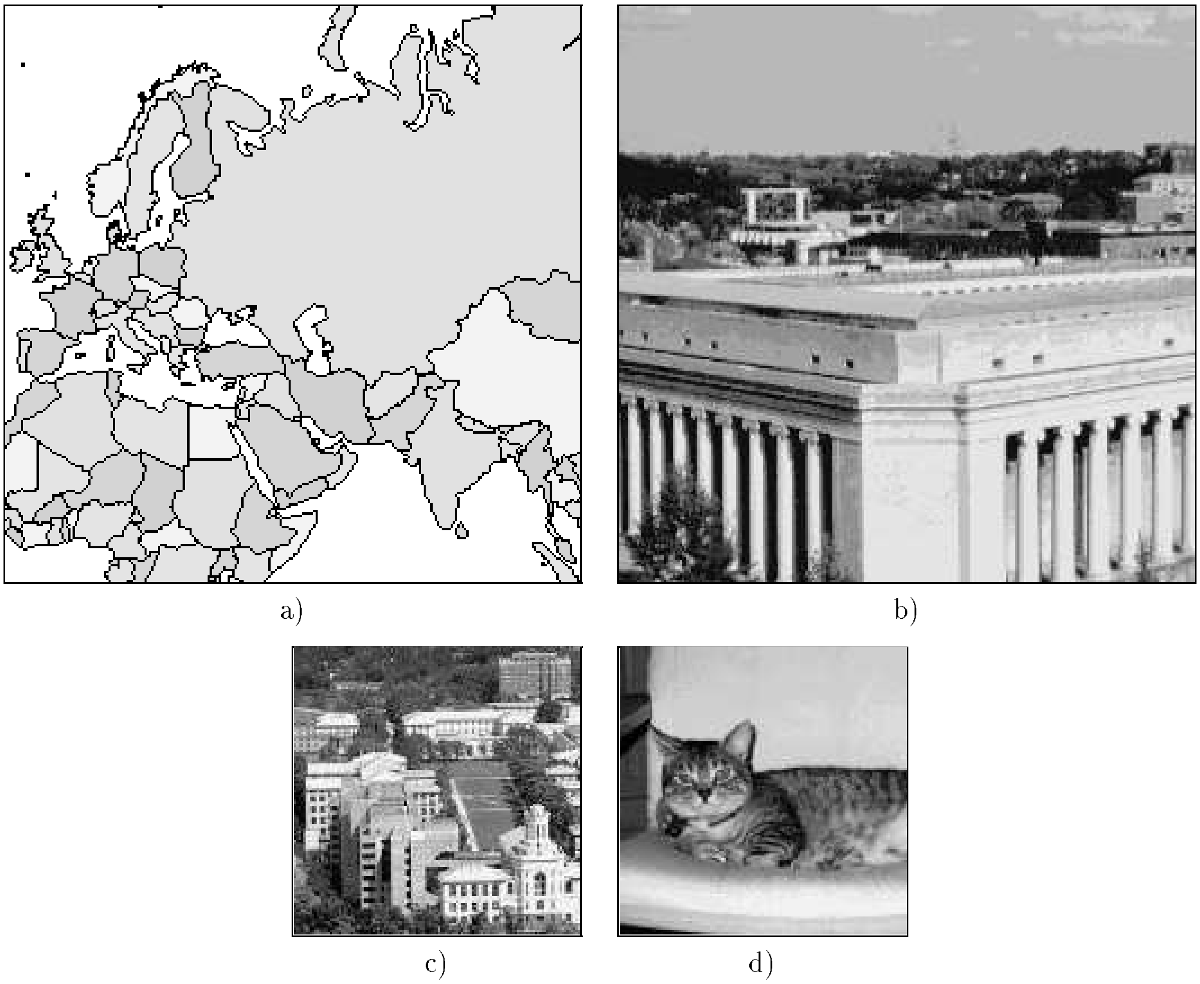


Figure 7: Random selection of Pictures. a) world map, b) Mellon Institute, c) Carnegie Mellon University campus, d) cat.

Figure 7 shows a sampling of images of different sizes and complexity. There are two different sizes represented here and varying number of colors. Table 1 shows these parameters as well as the number of nodes and leaves required to represent the images as OBDDs and as bintrees. The total number of nodes required by bintrees is approximately four times the total number required by OBDDs. The simplest way to implement these nodes is to use two pointers, one to each child. This makes traversal easy. OBDD nodes, unlike bintree nodes must also contain information indicating which level of the OBDD they are on. This is usually much smaller than the two pointers, requiring $\lceil \log_2(width + height) \rceil$ bits.

It is interesting to compare the bounds on the number of nodes for both data structures. We can calculate the worst cases for images based on their dimensions and the number of colors. The best case is trivial, representing degenerate images of 1 leaf node.

3.1.1 Worst Case Analysis

Our analysis of the worst case for the OBDD and bintree provides an upper bound on the amount of storage required by the data structures. Our analysis is based on the shape and size of the resulting data structures and not the actual characteristics of the worst case image.

For the bintree the largest number of nodes required is when it forms a complete binary tree of depth

Image	colors	OBDD nodes	OBDD leaves	OBDD total	Bintree nodes	Bintree leaves	Bintree total	OBDD/bintree
world (fig. 7a)	7	3701	7	3708	10610	10611	21221	0.175
Mellon (fig. 7b)	50	15826	50	15876	32891	32892	65783	0.241
CMU (fig. 7c)	40	7805	40	7845	14016	14017	28033	0.280
cat (fig. 7d)	50	6779	50	6829	12659	12660	25319	0.270

Table 1: Comparison of the nodes required for representing images as OBDDs and as bintrees.

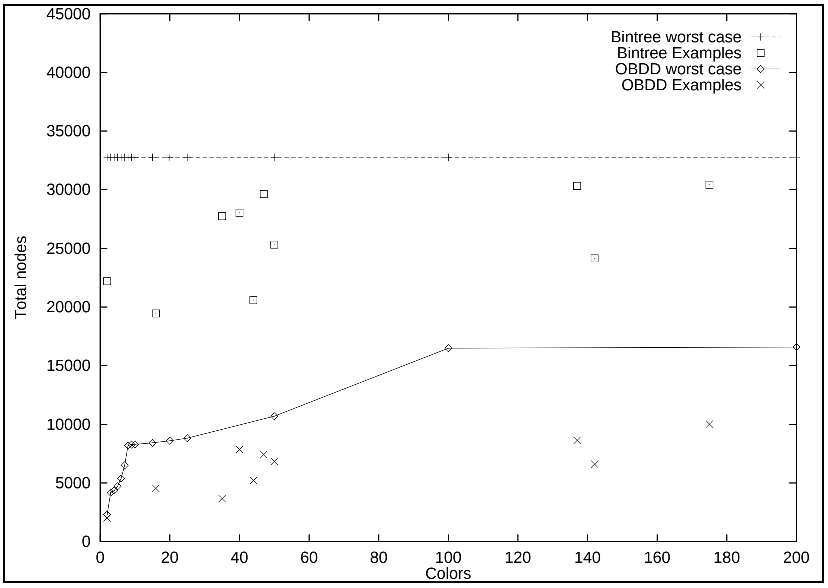
n, where n is based on the width and height of the image, $n = \lceil \log_2 width \rceil + \lceil \log_2 height \rceil$. Therefore, the number of nodes and leaves can be calculated as follows.

$$\begin{aligned} nodes_{bintree} &= 2^{\lceil \log_2 width \rceil + \lceil \log_2 height \rceil} - 1 \\ leaves_{bintree} &= 2^{\lceil \log_2 width \rceil + \lceil \log_2 height \rceil} \end{aligned}$$

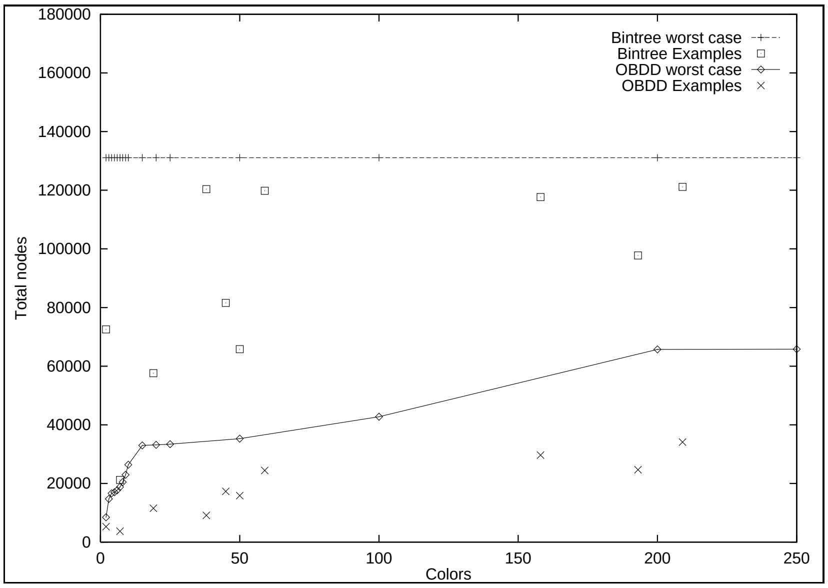
The Ordered Binary-Decision Diagram is more difficult to analyze. We use the findings in [5] to find the number of nodes. At each level of the OBDD the number of nodes are constrained by either the number of possible nodes for a full binary tree or the number of nodes given the possible combinations of the descendent nodes. For a large n, the top part of the OBDD will resemble a binary tree with $n - k$ levels. The bottom of the OBDD will be constrained by the number of colors, c and their combinations. The division between the top part and the bottom part of the tree is where the number of combinations exceeds the number of possible nodes formed by a binary tree. The number of nodes in the bottom part of the tree is bounded by c^{2^k}.

$$\begin{aligned} nodes_{OBDD} &= min_k(2^{n-k} - 1) + (c^{2^k}) \\ leaves_{OBDD} &= c \end{aligned}$$

The results of these calculations are presented in Figures 8a and b. Figure 8a shows how the number of nodes in the worst case changes depending on the number of colors in a 128 by 128 pixel image. The figure also shows the number of nodes for each encoding of ten sample images including the two figures mentioned in table 1 with these dimensions. These real data show that there are typically a number of patterns found in images. Figure 8b shows the data for 256 by 256 pixel images again with ten sample image data points including two from table 1.



a)



b)

Figure 8: Comparison of 14 (a) and 16 (b) level bintree and OBDD representations for images

3.2 Comparison of File Storage

To compare the amount of file storage required by these two methods we take a rather simple approach. In both cases we wish to incorporate enough information to enable reconstructing the same data structure from the data. For both we will assume bit level input/output streams which we have implemented on character based streams.

Bintrees can be stored by introducing an additional character to the set of colors, c, as suggested by [7]. Therefore, the number of bits per symbol is equal to $\lceil \log_2 (c+1) \rceil$. Multiplying this by the total number of nodes gives the size of the file. We also need to store the number of colors which we assume is stored in a 32 bit unsigned integer. We need to list the symbol and the actual color value it represents which is $c * \lceil \log_2 (c+1) \rceil * 32$ bits, assuming that the description of the colors requires 32 bits.

$$\begin{aligned} file\ size_{bintree} &= 32 \\ &+ (\lceil \log_2 (c+1) \rceil + 32) * c \\ &+ \lceil \log_2 (c+1) \rceil * (nodes_{bintree} + leaves_{bintree}) \end{aligned}$$

OBDDs require a little more information. We need to store the number of colors, the total number of nodes and leaves and the number of levels. We need a mapping of the node representing the color to the color value. We also need to store the level number before each set of nodes for a level along with the number of nodes at that level. Then we list the nodes in numerical order specifying only their two child nodes.

$$\begin{aligned} file\ size_{OBDD} &= 32 + 32 + 32 \\ &+ (\lceil \log_2 (nodes_{OBDD} + leaves_{OBDD}) \rceil + 32) * c \\ &+ (\log_2 width + \log_2 height) * (\lceil \log_2 (nodes_{OBDD} + leaves_{OBDD}) + 32) \\ &+ 2 * \lceil \log_2 (nodes_{OBDD} + leaves_{OBDD}) \rceil * nodes_{OBDD} \end{aligned}$$

		OBDD			Bintree			
Image	colors	nodes	leaves	bits	nodes	leaves	bits	OBDD/bintree
world (fig. 7a)	7	3701	7	88932	10610	10611	63716	1.396
Mellon (fig. 7b)	50	15826	50	446260	32891	32892	395030	1.130
CMU (fig. 7c)	40	7805	40	205456	14016	14017	168470	1.219
cat (fig. 7d)	50	6779	50	179230	12659	12660	152246	1.177

Table 2: Comparison of the file size in bits required for representing images as OBDDs and as bintrees.

Table 2 shows the number of bits required for the sample images stored using these schemes. On average OBDDs produce files apprximately 25 percent larger than bintrees.

Ordered Binary-Decision Diagrams appear to provide more compressed data structures for manipulating images in memory. The number of nodes required to represent the images is the important factor. Although the nodes are slightly larger in OBDDs, the memory requirements are much less than that for binary quadtrees. When creating a file for storing the data structures, the binary quadtree produces slightly smaller files.

4 Compressing Movies Using Ordered Binary-Decision Diagrams

We have shown how images can be compressed using Ordered Binary-Decision Diagrams. One of the prime features of OBDDs that permits less data storage for storing images is their sharing of commom sub-OBDDs representing subimages. We use this infrastructure for sharing common subimages to investigate storing sequences of related images using OBDDs.

Movies are composed of sequences of related images. Typically these images do not change greatly from frame to frame to give the impression of continuity when viewed in sequence. Using OBDDs we can exploit the similarities between successive frames.

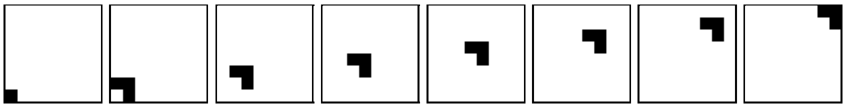


Figure 9: A simple 8 frame "movie"

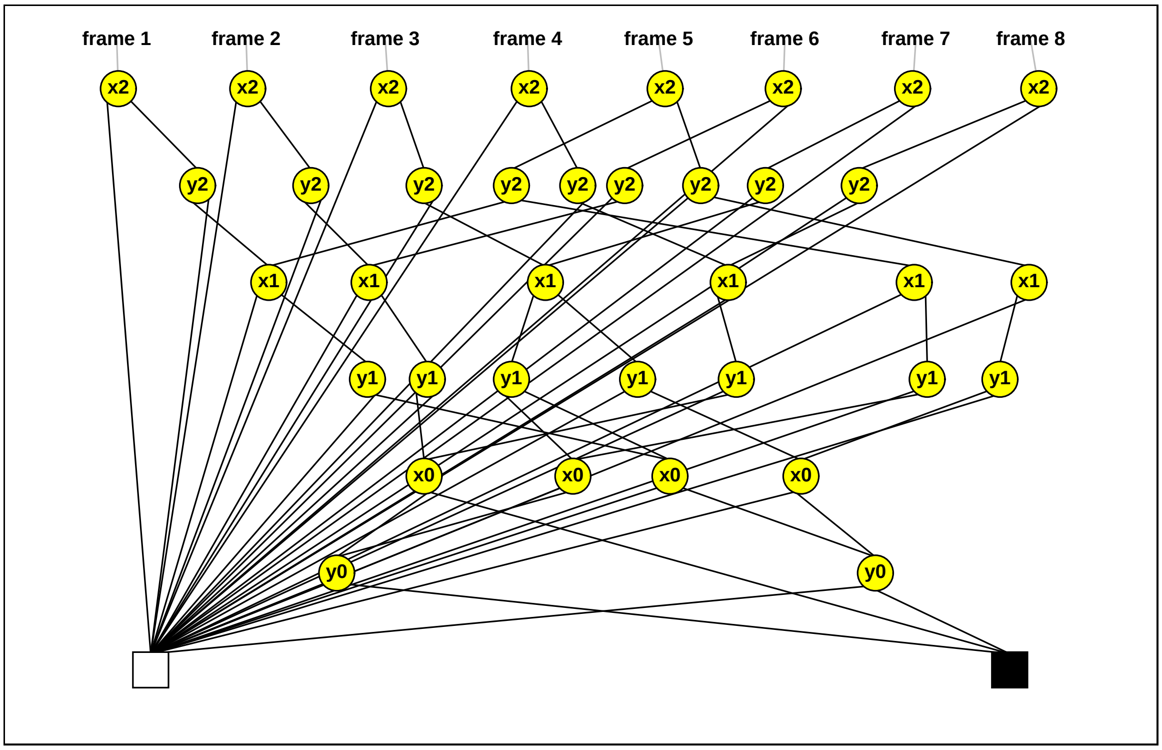


Figure 10: The OBDD representation of the entire "movie"

Figure 9 shows a very simple example of a "movie". Each of the frames is related in that the "background" is remaining constant and the "arrow" in the foreground is moving from the bottom left corner to the top right corner. If we build a OBDD for each of these frames the resulting set of individual representations requires a sum of 80 nodes. By combining the OBDDs as shown in figure 10 we get a OBDD containing 38 nodes.

This is a very simple example but illustrates how movie OBDDs can be constructed. Once the OBDD has been built, each frame can be individually accessed just by traversing the OBDD from the root representing that frame.

5 Conclusions

We have shown how images can be compressed using Ordered Binary-Decision Diagrams. The number of nodes required to represent an image is one quarter the number required when using binary quadtrees. We have shown the number of nodes required in the worst case as well as some sample images. Storing the OBDDs to files requires slightly more space than storing binary quadtrees.

The mechanism of sharing identical sub-OBDDs representing subimages is also useful when compressing related image sequences such as movies. We therefore use the same compression technique for storing inter-frame and intra-frame differences and similarities.

We believe that operations on OBDDs can be as efficient if not more efficient as those on binary quadtrees. This remains as future work as we continue our research into using this versatile data structure for representing images.

References

[1] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *International Conference on Computer-Aided Design*, pages 188–191, November 1993.

[2] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[3] E. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. C.-Y. Yang. Spectral transforms for large boolean functions with applications to technology mapping. In *30th ACM/IEEE Design Automation Conference*, pages 54–60, June 1993.

[4] K. Knowlton. Progressive transmission of grey-scale and binary pictures by simple, efficient, and lossless encoding schemes. *Proceedings of the IEEE*, 68(7):885–896, July 1980.

[5] Heh-Tyan Liaw and Chen-Shang Lin. On the OBDD-representation of general boolean functions. *IEEE Transactions on Computers*, 41(6):661–664, June 1992.

[6] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

[7] M. Tamminen. Comment on quad- and oct-trees. *Communications of the ACM*, 27(3):248–249, March 1984.