המעבדה לעיבוד גיאומטרי של תמונות
Geometric Image Processing Laboratory

Rate-Distortion Optimized Tree-Structures for Image Compression

Project Report

By: Moshe Eliasof , Sefi Fuchs

Supervisor: Mr. Yehuda Dar

Table of Contents

3..... Rate-Distortion Optimized Tree-Structures for Image Compression
4..... 1D Signals - Binary Trees
4..... Binary-Tree Pruning
5..... Leaf-Join Algorithm
7..... Results
12..... Extension to 2D: Quad-Tree Algorithm
12..... Quad-Tree pruning algorithm
13..... Leaf-Join Algorithm
17..... Mathematical Justification
19..... Results
26..... Comparison to previous project results
27..... Summary

Rate-Distortion Optimized Tree-Structures for Image Compression

In this report, we will discuss, analyze and compare methods for image compression.
Our project is based on the paper by Shukla et al. [1].
Shukla et al. [1] proposed an approach addressing a fundamental problem in the structure of trees (binary/quad trees), and utilized it for compression of one-dimensional signals and (two-dimensional) images.

This algorithm is called "Join-Prune" and it manages to exploit the similarities between neighboring blocks by merging them, even if they are represented in the tree by leaves being children of different parent nodes.

The problem with standard trees, is that they limit the possibilities to merge similar neighboring blocks, since they induce a very strict form of squared blocks in a signal.
For example, in a standard quad-tree it is impossible to merge three similar dyadic blocks.
Therefore, we must use their split form (as can be seen in Fig. 1), which costs more bits.

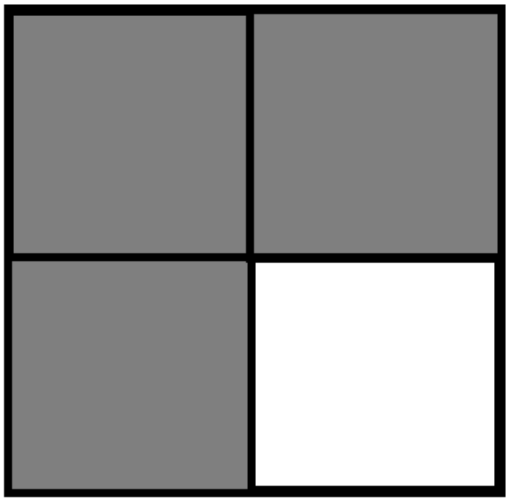


Figure 1: An example in which a quadtree will have to split this block into four blocks.

This exemplifies the motivation for the algorithm suggested in [1].

1D Signals - Binary Trees

A common way of representing 1D signals is by binary trees.

The structure of the binary tree is such that for each inner node (i.e. non-leaf) there are at most two children.

The motivation for using such data structures to represent 1D signal is clear – each level in the tree can be thought of dividing the signal into two halves, and we are doing it recursively, up to some depth.

Obviously, the deeper the tree – the more accurate the signal will be represented, and the more bits it will cost.

To store a signal in an efficient manner while maintaining them as close as possible to the original signal, compression algorithms are introduced.

A popular way of storing a 1D signal is by a Pruned-Binary tree.

Let us discuss this algorithm:

Before introducing the algorithm, we shall define the Lagrangian cost of a node L:

$$L = D + \lambda R \ ,$$

Where D and R are the distortion and bit-rate of the node, respectively.

Many compression techniques optimize the rate-distortion tradeoff by minimizing the Lagrangian cost defined above.

Notice that the Lagrangian optimization provides a solution depending on the value of λ:

For a higher value of λ the bit-rate R is lower, and for a lower λ a larger R will be introduced.

Binary-Tree Pruning

1. Segment the image using the binary-tree decomposition up to a tree depth J.
2. Approximate each node by the mean value of the pixels reside in it.
3. Generate an rate-distortion curve for each node in the tree.
4. *Lagrangian cost-based pruning:* For the given operating slope $-\lambda$ prune the children if the sum of their Lagrangian costs is greater or equal to the Lagrangian cost of the parent node. That means:

Prune the children nodes if:

$$D_{c1} + D_{c2} + \lambda(R_{c1} + R_{c2}) \geq D_p + \lambda R_p$$

The pruning steps is done recursively until no more prunes can be done.

Notice, that this is one iteration of the algorithm, for an operating slope – λ.

For compressing an image for a given bit-rate R_0 , we need to search for the operating slope $-\lambda^*$ which corresponds to the desired bit-rate R_0.

The search algorithm is as follows:

1. Select two starting extreme values for $\lambda_{\min}$ and $\lambda_{\max}$.(Usually , min will be close or equal to zero, and max will be $\sim 10^6$
   The condition that should hold when choosing lambdas is:
   $$R_{\lambda_{\max}} \leq R_0 \leq R_{\lambda_{\min}}$$
   Upon equality for either one of the lambdas, stop.
2. Run the QT pruning algorithm for $\lambda_{\min}$ and $\lambda_{\max}$.

Denote the results from the algorithm as follows:

$$D_{\lambda_{\min}}, R_{\lambda_{\min}}, D_{\lambda_{\max}}, R_{\lambda_{\max}}$$

3. Like in step 1, if one of the sides hold equality:
$$R_{\lambda_{\max}} \leq R_0 \leq R_{\lambda_{\min}}$$
Stop, we found an exact solution.
4. Otherwise, compute :
$$\lambda_{new} = \frac{D_{\lambda_{\min}} - D_{\lambda_{\max}}}{R_{\lambda_{\max}} - R_{\lambda_{\min}}}$$
5. Run the pruning algorithm for λ_{new}. Denote the distortion and rate by $D_{\lambda_{new}}$ and $R_{\lambda_{new}}$.
6. If $R_0 = R_{\lambda_{new}}$ then the optimum is found - stop.
If $R_0 < R_{\lambda_{new}}$ then set $\lambda_{\min} = \lambda_{new}$ and go to line 5.
Else, set $\lambda_{\max} = \lambda_{new}$ and go to line 5.

As discussed before, this method fails to exploit the similarities among neighbors in the pruned-tree.

The paper this project is based on, suggests the following algorithm for solving the disadvantage of the prune-algorithm:

Leaf-Join Algorithm

1. Scan all the leaves from left to right
2. For the current leaf denoted by n_1 and its neighbor (notice there could be more than one neighbor) check if the Lagrangian condition holds:
If $D_1 + \lambda R_1 + D_2 + \lambda R_2 > D_{12} + \lambda R_{12}$ where D_{12} and R_{12} denote the distortion and bit-cost of the joint-leaves, respectively, then perform a join between these leaves.
If a join was performed – an indicator bit will be set to 1 and we will treat the joint-leaf a new leaf.
Otherwise, the indicator bit will be set to 0 and nothing changes.
3. As long as there are unchecked leaves in the quadtree, go to step 2.

Let us describe the algorithm for finding neighbors in the trees:

Firstly, suppose that n_j^i represents the i^{th} leaf in the j^{th} level of the binary tree.

Assume that the current leaf we are scanning is n_j^i , then its neighbor indices i_0 at level j_0 are given by:

$$Left\ neighbor:\ i_0 = 2^{(j_0 - j)} i - 1$$
$$Right\ neighbor: i_0 = 2^{j_0 - j}(i + 1)$$

When joining two leafs, we consider them as a new leaf in the tree. However, we don't change the topology of the tree.

We indicate a joint of two leaves by an indicator function telling if two blocks are to be merged.

The cost of representing each joint is 2 bits – one bit for the merging indication, and one bit for describing to which neighbor (left/right) the current block was attached.

Notice that the joining procedure comes as an addition to the prune algorithm as a postprocessing stage.

Therefore, the full algorithm is as follows:

1. Select two extreme starting values for $\lambda_{\min}$ and $\lambda_{\max}$.(Usually, min will be close or equal to zero, and max will be $\sim 10^6$
   The condition that should hold when choosing lambdas is:
   $$R_{\lambda_{\max}} \leq R_0 \leq R_{\lambda_{\min}}$$
   Upon equality for either one of the lambdas, stop.
2. Run the pruning algorithm for $\lambda_{\min}$ and $\lambda_{\max}$.
3. Run the leaf-join algorithm for the pruned-trees corresponding to for $\lambda_{\min}$ and $\lambda_{\max}$.
   Denote the results from the algorithm as follows:
   $$D_{\lambda_{\min}}, R_{\lambda_{\min}}, D\lambda_{\max}, R_{\lambda_{\max}}$$
4. Like in step 1, if one of the sides hold equality:
   $$R_{\lambda_{\max}} \leq R_0 \leq R_{\lambda_{\min}}$$
   Stop, we found an exact solution.
5. Otherwise, compute:
   $$\lambda_{new} = \frac{D_{\lambda_{\min}} - D_{\lambda_{\max}}}{R_{\lambda_{\max}} - R_{\lambda_{\min}}}$$
6. Run the pruning followed by the leaf-join algorithm for λ_{new}. Denote the distortion and rate by $D_{\lambda_{new}}$ and $R_{\lambda_{new}}$.
7. If $R_0 = R_{\lambda_{new}}$ then the optimum is found - stop.
   If $R_0 < R_{\lambda_{new}}$ then set $\lambda_{\min} = \lambda_{new}$ and go to line 5.

Else, set $\lambda_{\max} = \lambda_{new}$ and go to line 5.

Results

In this section we will show the results we achieved in our implementation of this algorithm.
Example 1: Chirp function:

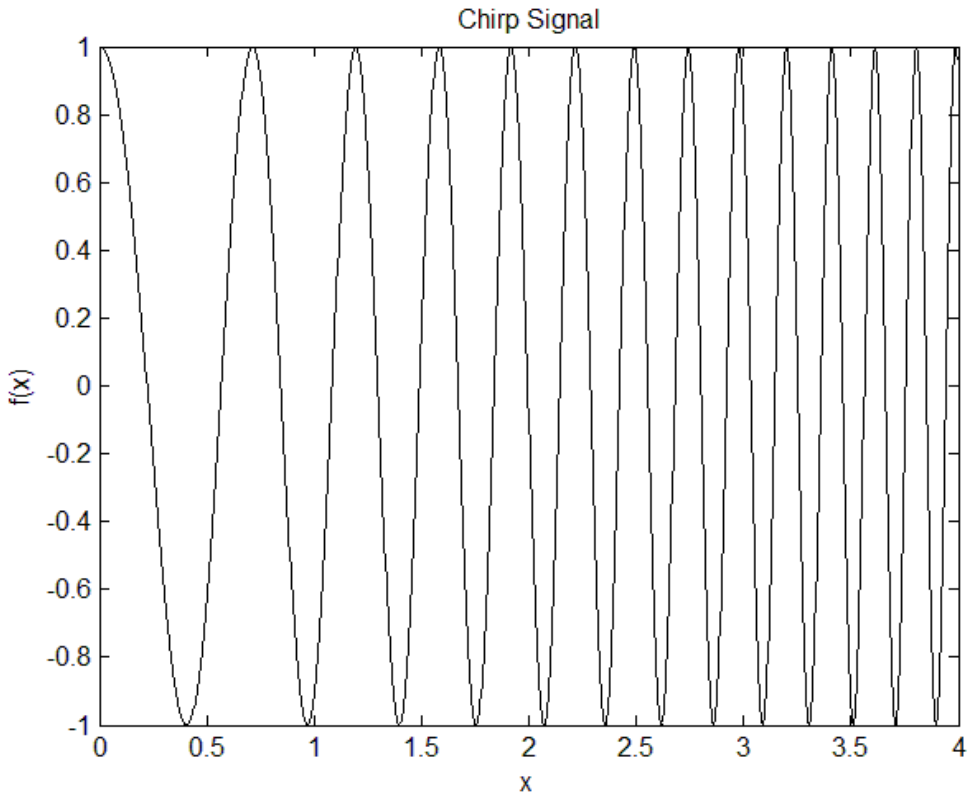


Figure 2: Original Signal

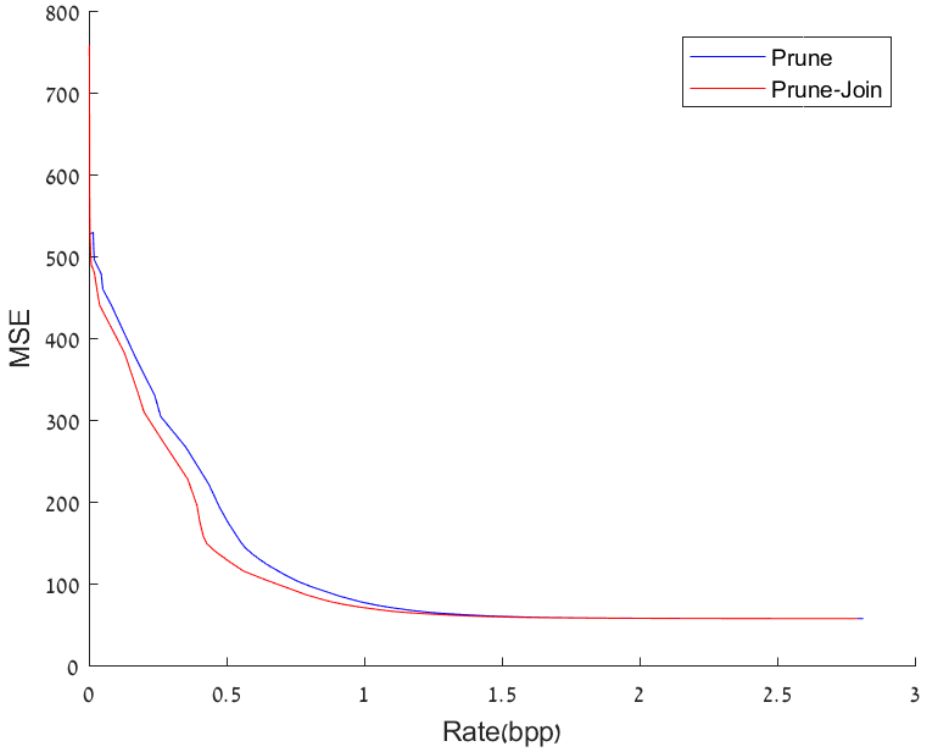
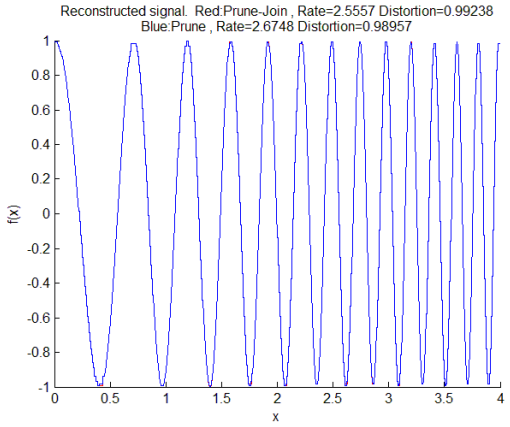
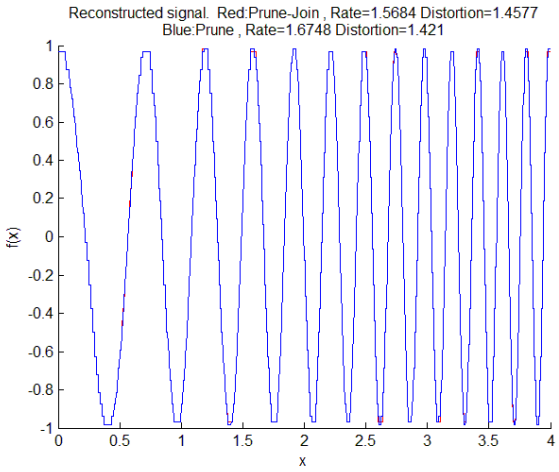
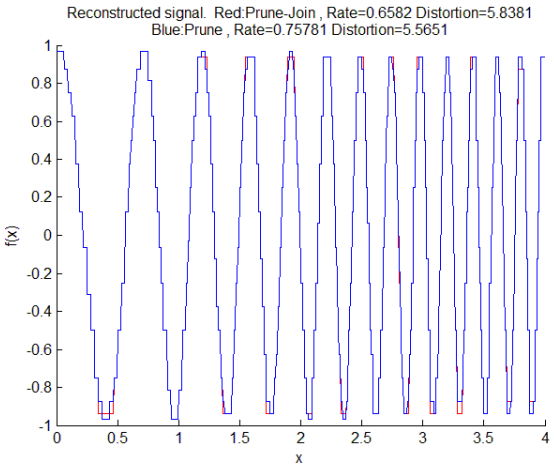
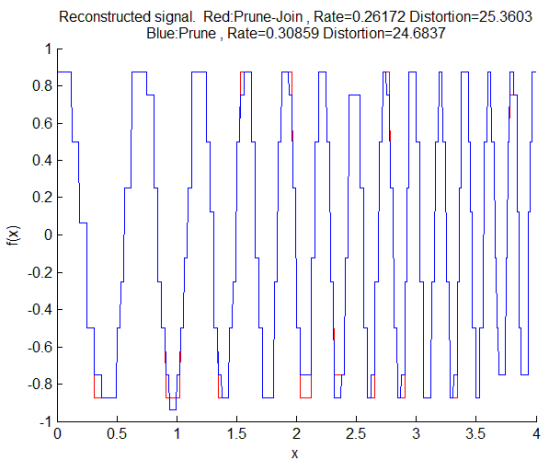


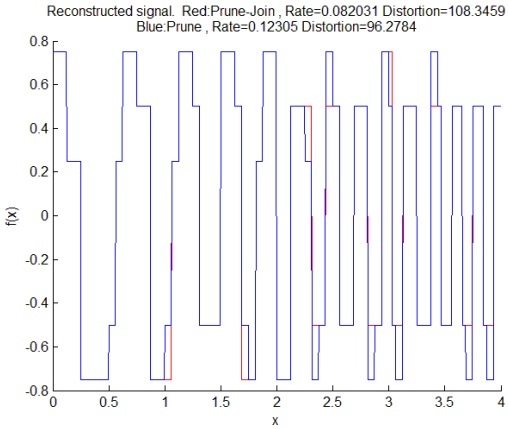
Figure 3: Rate Distortion Graph for Chirp

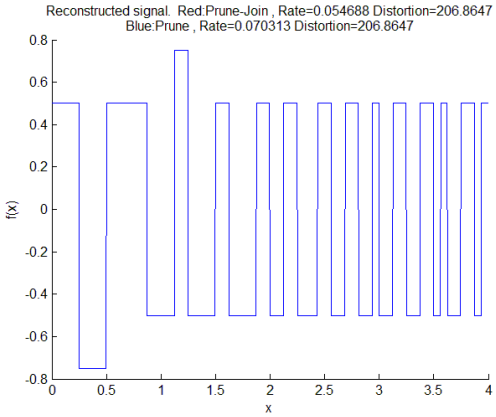












We can see, by the RD graph and the reconstruction graph that indeed the leaf-join is able to reduce the MSE for a given bit-cost.
We can also see that the join procedure is done only when the Lagrangian can be minimized. For instance, we see that for extremely low/high bit-rates the no joins were done, since it wouldn't have reduced the Lagrangian.

Example 2: Sinus function:

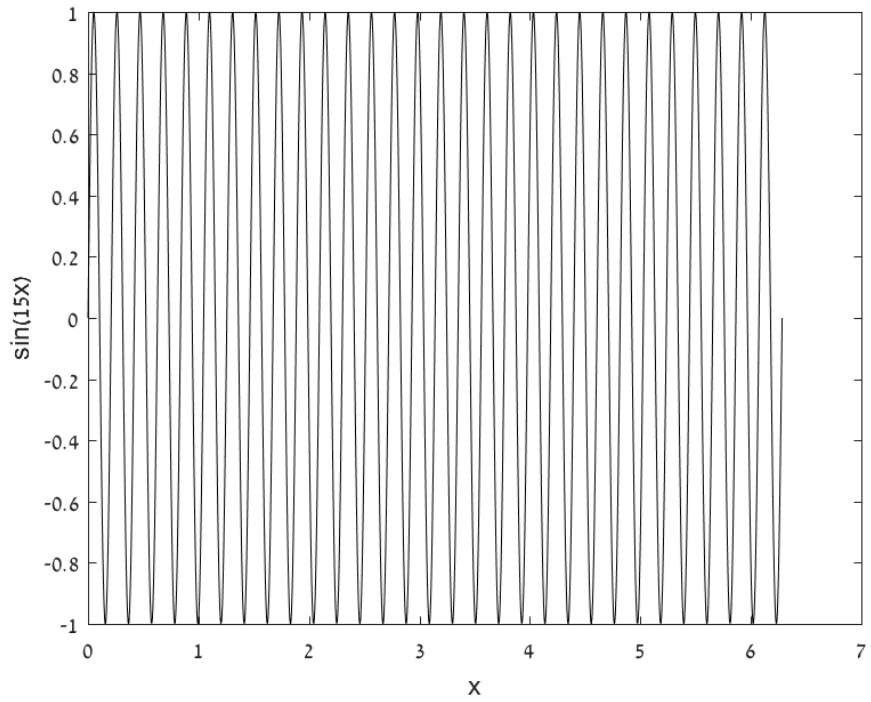


Figure 4: Original Signal

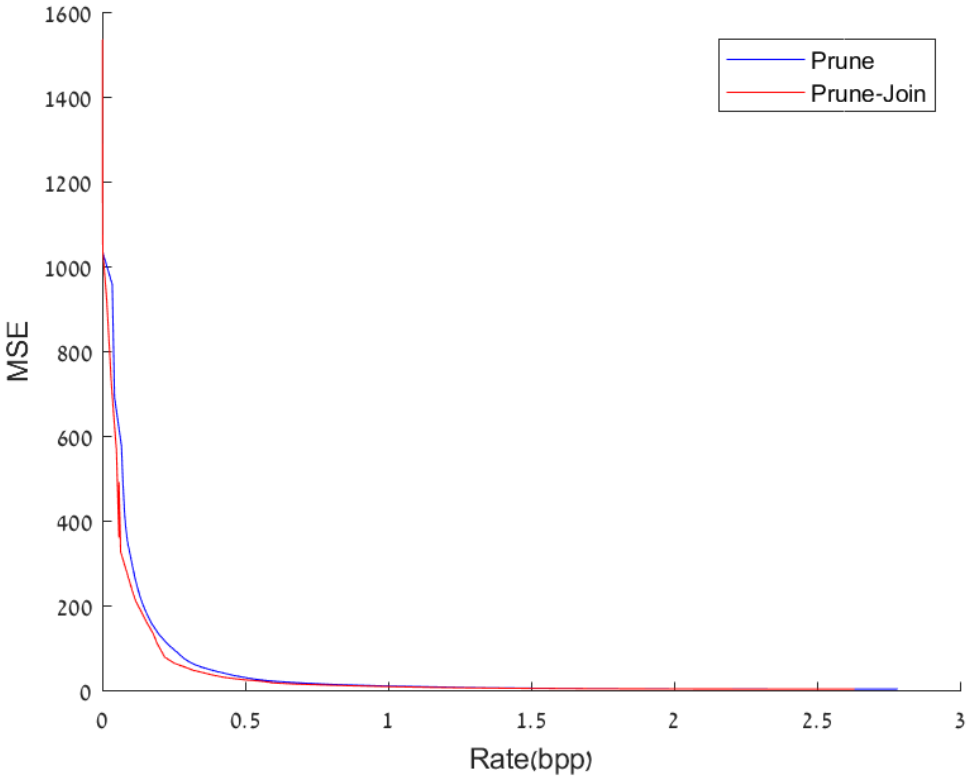
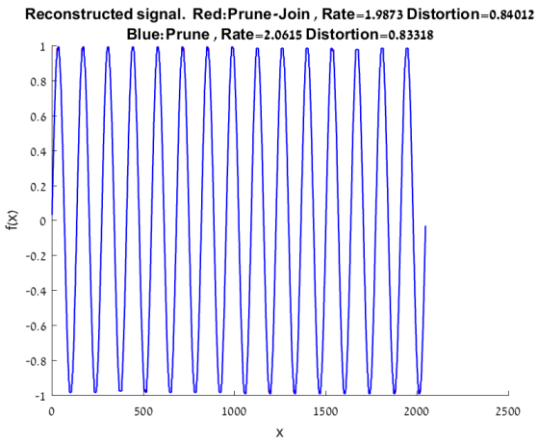
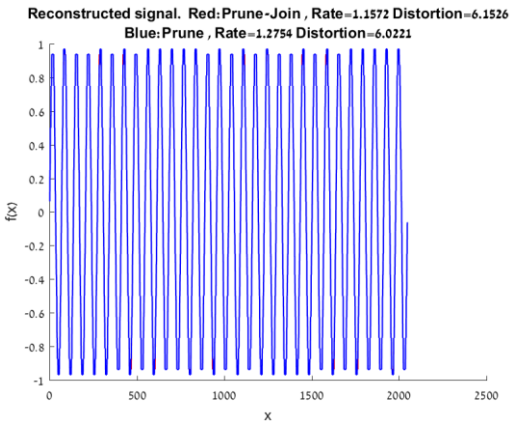
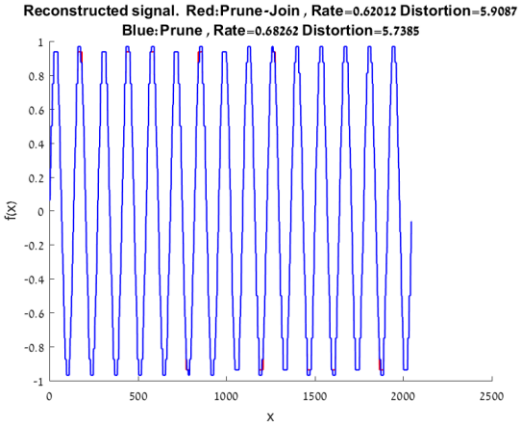
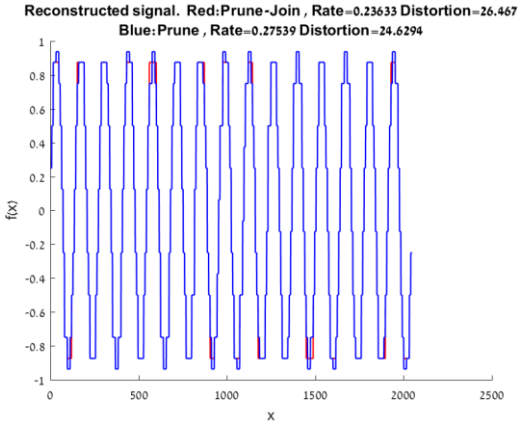


Figure 5: Rate Distortion Graph









As expected, we can see that in this example the leaf-join reduced the MSE for a given bit-rate from the RD graph.
However, we can also see that in this case, since the signal a simple signal and improvement over the prune algorithm is marginal since there aren't many "details" in the signal, as opposed to the chirp signal.

Extension to 2D: Quad-Tree Algorithm

All the algorithms discussed for 1D signals also have extensions for 2D signals.
We will show and discuss them in this section.
First, the signal will be represented using a Quad-tree instead of a Binary-tree.
The structure of the tree consists of 4 children for each internal node.
Each block in the Quad-tree based segmentation has four neighboring blocks, in contrast to two neighbors in the 1D case.

Quad-Tree pruning algorithm:

1. Segment the image using the Quad-tree decomposition up to a tree depth J.
2. Approximate each node by the mean value of the pixels reside in it.
3. Generate an RD curve for each node in the Quad-tree.
4. *Lagrangian cost-based pruning:* For the given operating slope $-\lambda$ prune the children if the sum of their Lagrangians cost is greater or equal to the Lagrangian cost of the parent node. That means:

Prune the children nodes if:

$$D_{c1} + D_{c2} + D_{c3} + D_{c4} + \lambda(R_{c1} + R_{c2} + R_{c3} + R_{c4}) \geq D_p + \lambda R_p$$

The pruning steps is done recursively until no more prunes can be done.

Notice, that this is one iteration of the algorithm, for an operating slope – λ.
For compressing an image for a given bit-rate R_0 , we need to search for the operating slope $-\lambda^*$ which corresponds to the desired bit-rate R_0.
The search algorithm is as follows:

7. Select values for $\lambda_{\min}$ and $\lambda_{\max}$.(Usually , min will be close or equal to zero, and max will be $\sim 10^6$
   The condition that should hold when choosing lambdas is:
   $$R_{\lambda_{\max}} \leq R_0 \leq R_{\lambda_{\min}}$$
   Upon equality for either one of the lambdas, stop.
8. Run the QT pruning algorithm for $\lambda_{\min}$ and $\lambda_{\max}$.
   Denote the results from the algorithm as follows:
   $$D_{\lambda_{\min}}, R_{\lambda_{\min}}, D_{\lambda_{\max}}, R_{\lambda_{\max}}$$
9. Like in step 1, if one of the sides hold equality:
   $$R_{\lambda_{\max}} \leq R_0 \leq R_{\lambda_{\min}}$$
   Stop, we found an exact solution.
10. Otherwise, compute :
    $$\lambda_{new} = \frac{D_{\lambda_{\min}} - D_{\lambda_{\max}}}{R_{\lambda_{\max}} - R_{\lambda_{\min}}}$$
11. Run the QT pruning algorithm for λ_{new}. Denote the distortion and rate by $D_{\lambda_{new}}$ and $R_{\lambda_{new}}$.
12. If $R_0 = R_{\lambda_{new}}$ then the optimum is found - stop.
    If $R_0 < R_{\lambda_{new}}$ then set $\lambda_{\min} = \lambda_{new}$ and go to line 5.
    Else, set $\lambda_{\max} = \lambda_{new}$ and go to line 5.

Like in the 1D case, this method fails to exploit the similarities among neighbors in the pruned-tree.
In the next section, we discuss the suggestion for solving this sub-optimality problem.

Leaf-Join Algorithm:

As discussed before, the novelty of this scheme is in its ability to exploit dependencies between neighbors in the pruned tree.
Clearly, just like in the 1D case, this scheme allows to join similar neighboring blocks (leaves) even if they have different parents in the pruned-tree.
Furthermore, in the 2D case it allows joining two or three blocks.
This scheme allows for many possibilities regarding blocks shape, which in many cases yield far better results than the restricting square blocks.
Therefore, it leads to much better results in terms of compression.
The algorithm is like the 1D algorithm. However, it is a bit more complicated in the sense of finding the neighbors since now there are 4 possible neighbors for each leaf, as can be seen in the following figure.

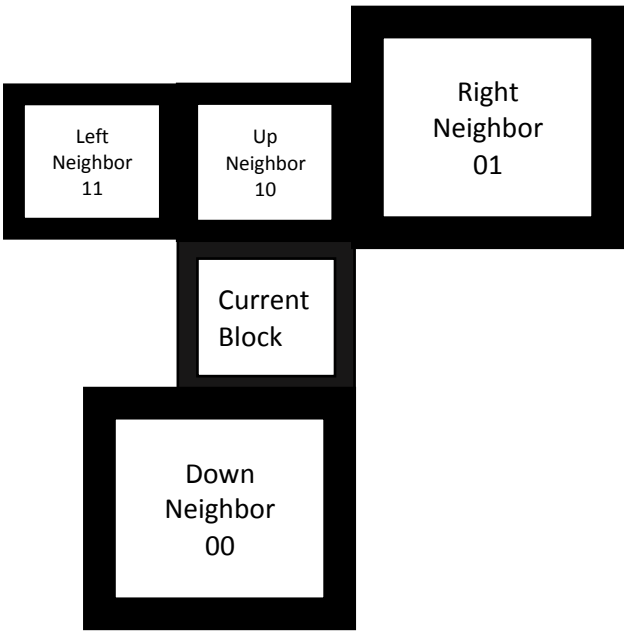


Figure 6: 4-connected neighbors in the image

The algorithm for finding the neighbors of a leaf is as follows:

Denote the current leaf n_1 , and the leaf we want to check if it is a neighbor of n_1 will be denoted as n_2^{i+j} and suppose their origin (bottom-left points) in the original signal are located at:

(x_1, y_1) , (x_2, y_2).

Also, denote their sizes by s_1, s_2 , respectively.

The position of the neighbor is coded by the tuple (i,j) as follows:

Down Neighbor : (0,0) , Right Neighbor : (0,1) , Up Neighbor : (1,0) , Left Neighbor : (1,1).

$if\ (y_2 + s_2 = y_1)$

$if\ (x_2 < x_1)$

$if\ (x_2 + s_2 > x_1)$, $then\ n_2\ is\ the\ down\ neighbor. else -$

$n_2\ is\ not\ the\ neighbor.$

$else, if\ (x_2 < x_1 + s_1), then\ n_2\ is\ the\ down\ neighbor. else -$

$n_2\ is\ not\ the\ neighbor$

$elseif\ (y_2 = y_1 + s_1)$

$if (x_2 < x_1)$

$if (x_2 + s_2 > x_1), then\ n_2\ is\ the\ up\ neighbor. else -$

$n_2\ is\ not\ the\ neighbor.$

$else, if\ (x_2 < x_1 + s_1), then\ n_2 is\ the\ up\ neighbor. else -$

$n_2\ is\ not\ the\ neighbor.$

$elseif\ (x_2 + s_2 = s_1)$

$if (y_2 < y_1)$

$if (y_2 + s_2 > y_1), then\ n_2\ is\ the\ left\ neighbor. else -$

$n_2\ is\ not\ the\ neighbor.$

$else, if\ (y_2 < y_1 + s_1), then\ n_2\ is\ the\ left\ neighbor. else -$

$n_2\ is\ not\ the\ neighbor.$

$elseif\ (x_2 = x_1 + s_1)$

$if (y_2 < y_1)$

$if (y_2 + s_2 > y_1), then\ n_2\ is\ the\ right\ neighbor. else -$

$n_2\ is\ not\ the\ neighbor.$

$else, if\ (y_2 < y_1 + s_1), then\ n_2\ is\ the\ right\ neighbor. else -$

$n_2\ is\ not\ the\ neighbor.$

The join phase is defined as follows:

1. Scan all the leaves from left to right
2. For the current leaf denoted by n_1 and its neighbor n_2^{i+j} $where\ \ i,j = 0,1$ (notice there could be more than one neighbor) check if the Lagrangian condition holds:
   If $D_1 + \lambda R_1 + D_2 + \lambda R_2 > D_{12} + \lambda R_{12}$ where $D_{12}\ and\ R_{12}$ denote the distortion and bit-cost of the joint-leaves, respectively, then perform a join between these leaves.
   If a join was performed – an indicator bit will be set to 1 and we will treat the joint-leaf a new leaf. Describing to which of the 4 neighbors the current leaf was attached to, we use 2 bits (i,j) and the their meaning can be found in the previous page.
   Otherwise, the indicator bit will be set to 0 and nothing changes.
3. As long as there are unchecked leaves in the quadtree, go to step 2.

The join-phase is done after the prune-phase in the full algorithm.
Therefore, we Prune-Join algorithm can be described as follows:
Given a desired bit-rate $R_0\ and\ initial\ values\ for\ \lambda_{\max}\ \ and\ \lambda_{\min}$ we perform the search algorithm described above for the prune algorithm, but this time we also add the join phase right after pruning the quadtree.
Thus, the full Prune-Join algorithm is as follows:

1. Select values for $\lambda_{\min}$ and $\lambda_{\max}$.(Usually, min will be close or equal to zero, and max will be $\sim 10^6$
   The condition that should hold when choosing lambdas is:
   $$R_{\lambda_{\max}} \leq R_0 \leq R_{\lambda_{\min}}$$
   Upon equality for either one of the lambdas, stop.
2. Run the QT pruning algorithm for $\lambda_{\min}$ and $\lambda_{\max}$.
3. Run the QT leaf-join algorithm for the pruned-trees corresponding to for $\lambda_{\min}$ and $\lambda_{\max}$.
   Denote the results from the algorithm as follows:
   $$D_{\lambda_{\min}}, R_{\lambda_{\min}}, D_{\lambda_{\max}}, R_{\lambda_{\max}}$$
4. Like in step 1, if one of the sides hold equality:
   $$R_{\lambda_{\max}} \leq R_0 \leq R_{\lambda_{\min}}$$
   Stop, we found an exact solution.
5. Otherwise, compute:
   $$\lambda_{new} = \frac{D_{\lambda_{\min}} - D_{\lambda_{\max}}}{R_{\lambda_{\max}} - R_{\lambda_{\min}}}$$
6. Run the QT pruning algorithm followed by the leaf-join algorithm for λ_{new}. Denote the distortion and rate by $D_{\lambda_{new}}\ and\ R_{\lambda_{new}}$.
7. If $R_0 = R_{\lambda_{new}}$ then the optimum is found - stop.
   If $R_0 < R_{\lambda_{new}}$ then set $\lambda_{\min} = \lambda_{new}$ and go to line 5.
   Else, set $\lambda_{\max} = \lambda_{new}$ and go to line 5.

Let us show an example for the outcome of this algorithm:

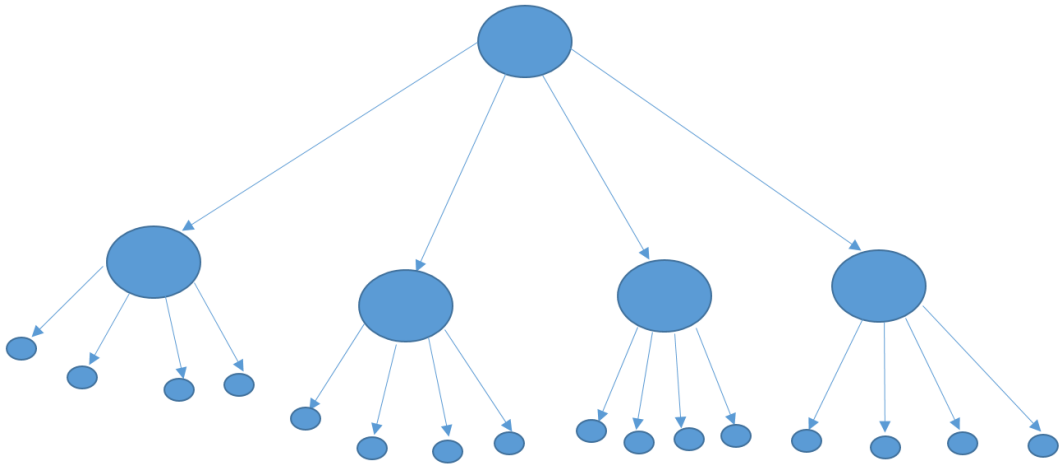


Figure 7:Standard Quad Tree .

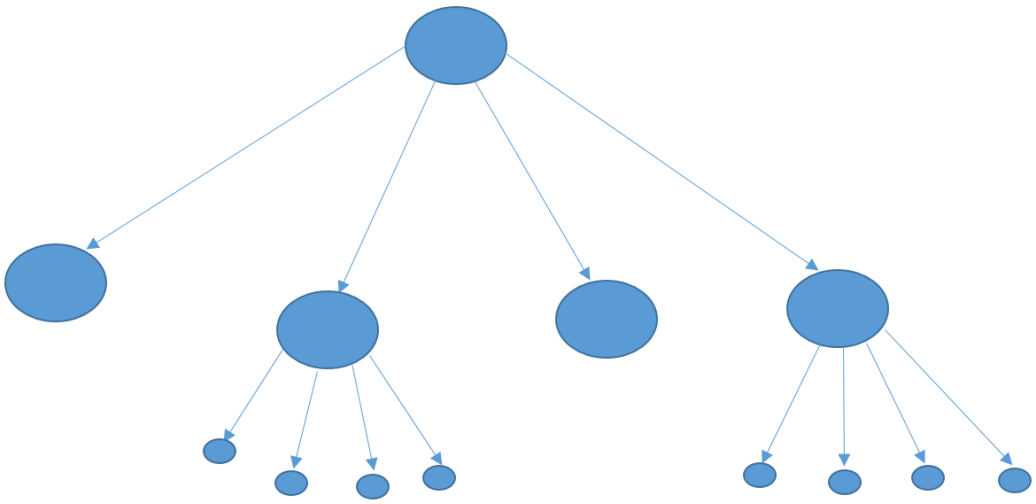


Figure 8: Pruned Quad Tree . Notice that some of the leaves were pruned.

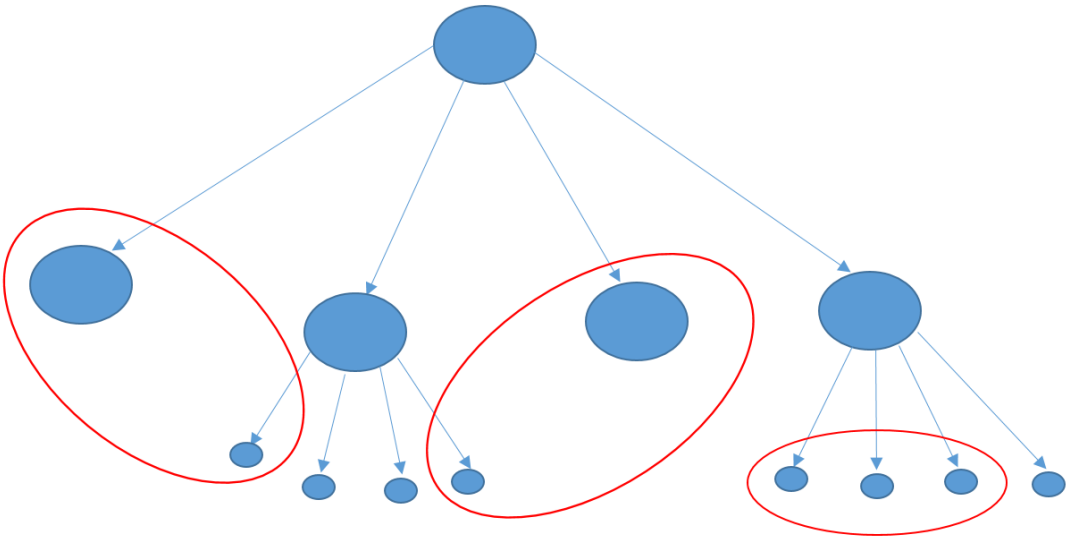


Figure 9: Prune-Join Quadtree . Leafs to be joint are surronded by red circles.

Mathematical Justification

To get a more comprehensive understanding of the algorithms and why they work we would like to mathematically analyze and justify this method.
First, we would like to show that as stated – this method provides an oracle R-D performance. (for polygonal model)
Consider the following polygon model: a white polygon shape with V vertices is placed on a uniform black background.
A possible oracle method would code the polygon's vertices position. Assuming R total bits given to the oracle, it means $\frac{R}{V}$ for each vertex on a regular grid on the unit square will provide us with quantized vertices positions with a distance of: $\Delta = \left(\frac{1}{\sqrt{2}}\right) 2^{-\frac{R}{2V}}$
Notice that since we are working on the unit square, each side length of the polygon is bounded by $\sqrt{2}$ and therefore the total length of the boundary is bounded by $\sqrt{2}V$.
Therefore, we conclude that the distortion created by this oracle method is bounded by $D(R) = \sqrt{2}V\Delta \leq \sqrt{2}V\left(\frac{1}{\sqrt{2}}\right)2^{-\frac{R}{2V}} = V2^{-\frac{R}{2V}}$.
Notice that for the polygon model, it is an exponentially decaying R-D function.
Now we will focus on showing that the suggested Prune-Join scheme also yields an exponentially decaying R-D function.

In the following paragraph we will show why the suggested algorithm in [1] holds the claims in the previous paragraph.
Note that the derivation and proof of the claims can be found in [1].

Using the lemmas from [1], an upper bound for the bit-cost of the prune-join quadtree algorithm is formed as:

$$R = R_{Leaves} + R_{Tree} + R_{LeafJointCoding} \leq 17\left(J + \frac{1}{2}\right)V \,, for\ V > 0$$

The net distortion is simply the sum of the distortion of all the leaves.
This leads to the following:

$$D_{Prune-Join} = V(2^{-2J}) + V\left(\frac{1}{4}2^{-2J}\right)$$

Combining the last two equations gives:

$$D_{Prune-Join}(R) = 2.5V * 2^{\frac{-2R}{17V}}$$

Therefore, the prune-join algorithm achieves an oracle-like exponentially decaying R-D function.

So far we have seen the R-D performance analysis assuming the polygonal model.
Notice that for real images this isn't the case. However, it can be easily modified to fit natural images. For example – instead of allocating r bits for representing the vertices for a polygon, we could also use these bits to quantize a constant value for a block, or polynom (by quantizing its coefficients using these bits). Hence, the proof is also valid for other models, and it is discussed for polygonal model for the sake of simplicity.

Another aspect we would like to stress in this report is the fact that the optimization problem we wish to minimize in the pruning algorithm is separable.
Using separability of the problem – for an operating slope – λ we can optimize each block independently.
Notice, that at each stage we perform the following:

- Given two neighboring leaves n_1, n_2 - check:
  If $D_1 + \lambda R_1 + D_2 + \lambda R_2 > D_{12} + \lambda R_{12}$ where $D_{12}\ and\ R_{12}$ denote the distortion and bit-cost of the joint-leaves, respectively, then perform a join between these leaves.
  If a join was performed – an indicator bit will be set to 1 and we will treat the joint-leaf a new leaf.
  Otherwise, the indicator bit will be set to 0 and nothing changes.

That means, every time we perform a join, we reduce the Lagrangian block/leaf wise, but all together this reduction achieves a minimization of the Lagrangian for the whole signal.
Let us look at the problem:
Our initial wish is to get to solve the following problem:

$$\arg\min_{\lambda} L_{image} = \arg\min_{\lambda} \sum_{i=1}^{N} (D_i + \lambda R_i)$$

Where N the number of is leaves in the quadtree, $D_i\ and\ R_i$ are the distortion and bit-cost of the i^{th} leaf, respectively and L_{image} is the Lagrangian of the whole image.

This problem is separable – if we minimize it for each block i then we minimize the total cost.
This shows that although at each step we are solving to find the local minimum considering two dyadic leaves, we end up finding the global minimum.

Results

In this section we will show the results we achieved in our implementation of this algorithm.

First, we would like to give an of the joint-trees in order to illustrate how the leaf-join algorithm exploits similarities between neighboring leafs (blocks) in the image:

Lena:

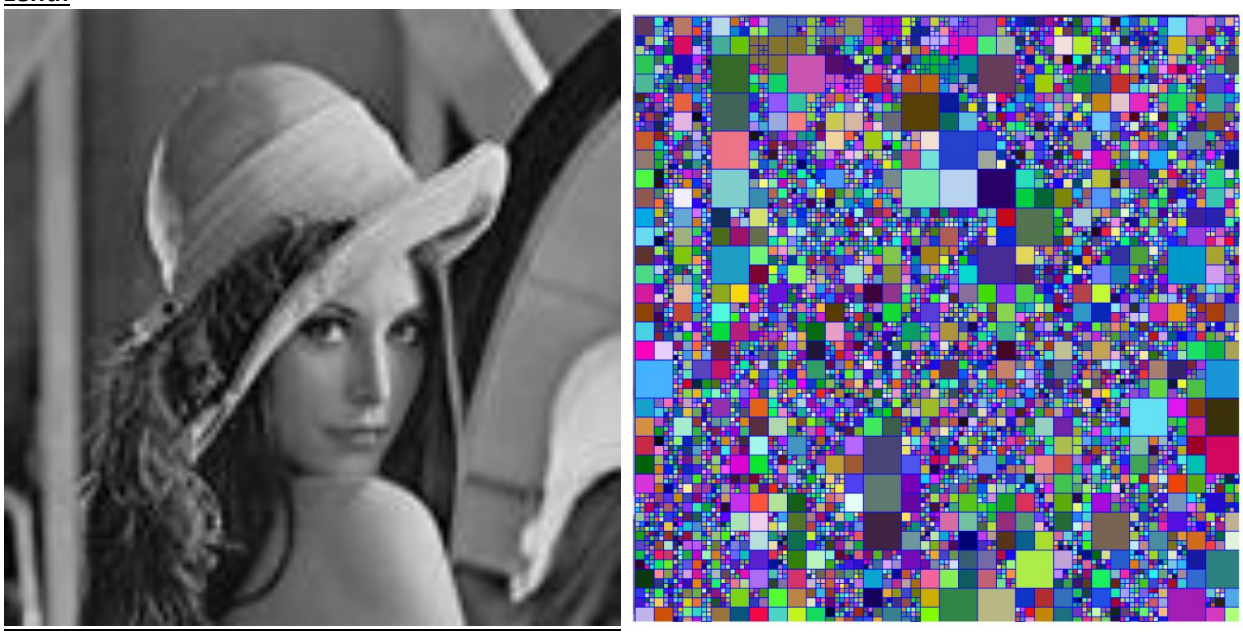


Figure 10: Prune-Join Quadtree representation of Lena

Joint-leaves are colored the same.
Notice how the structure of the image is preserved in the Quadtree representation, and the join of similar blocks.

Example 1: Boat

Figure 11: Original Image

Let us take a look at a sequence of results from the Prune-Join algorithm, followed by a R-D graph comparing between the two algorithms.

Lambda: 0.01 R: 5.5052 D: 29.3101

Lambda: 0.29764 R: 2.9847 D: 66.828

Lambda: 20.6914 R: 1.6311 D: 71.6202

Lambda: 112.8838 R: 1.1093 D: 92.8255

Lambda: 615.8482 R: 0.68916 D: 171.9686

Lambda: 3359.8183 R: 0.28935 D: 478.2814

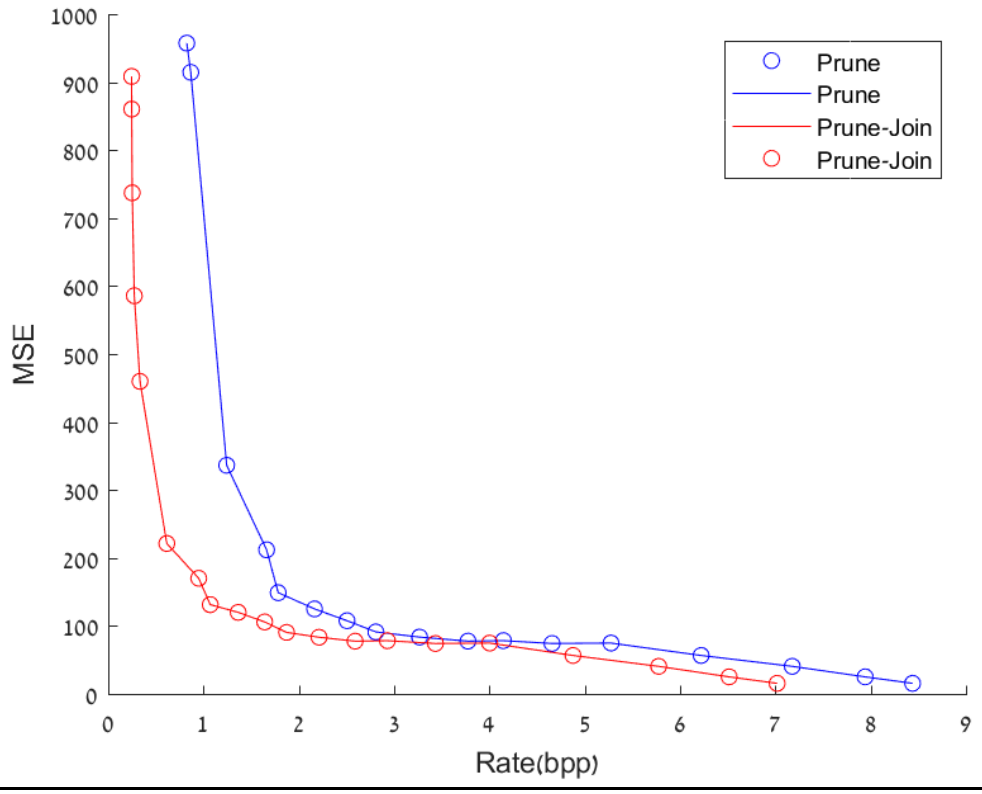


Figure 12: Rate Distortion Graph

We can see that the Prune-Join scheme improves the compression of the Prune algorithm. We can also see, that for points in which bit-rate is high, the prune-join algorithm yields similar or same results as the prune algorithm. This is due to the target of the algorithm – it is designed to minimize the Lagrangian for a given λ.
Thus, if it finds that the overhead for the join-phase in the algorithm increasing the Lagrangian to be greater than the prune-only Lagrangian, no joins are being done.

Example 2:

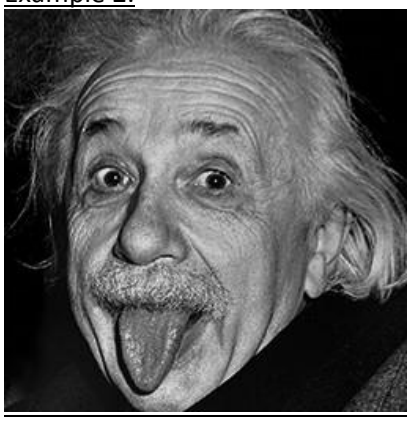
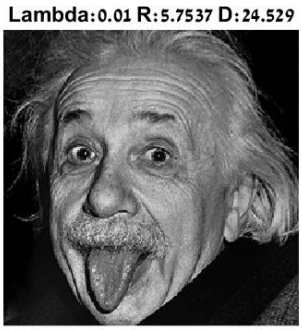
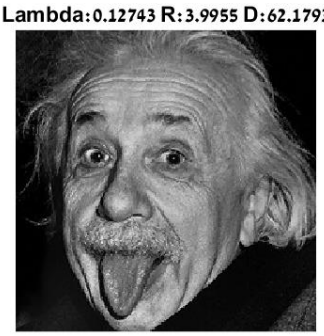
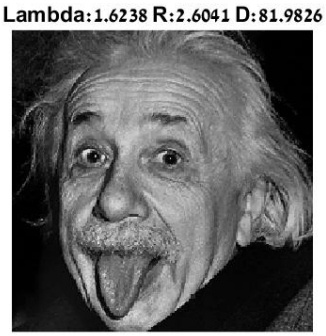
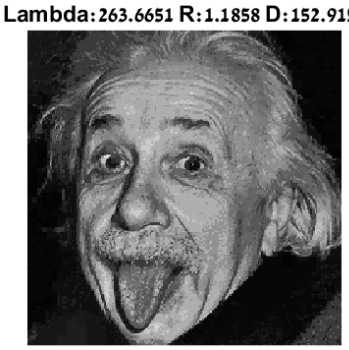


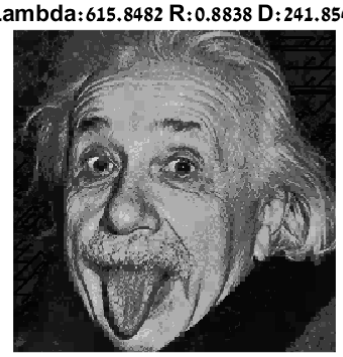
Figure 13: Original Image

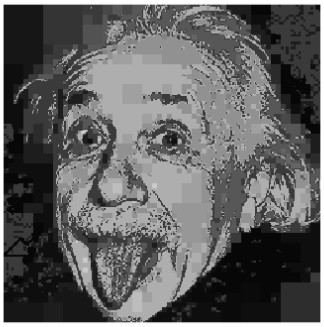












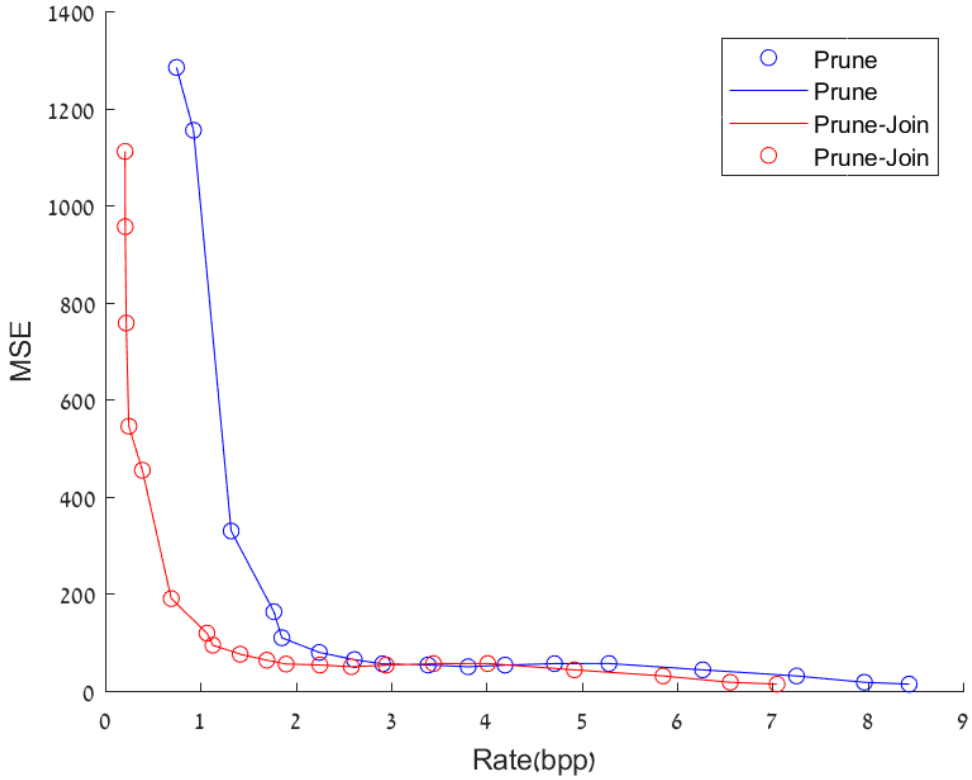


Figure 14: Rate Distortion Graph

In this example we can notice the same observations like in previous example.
Moreover, since this image includes less "details", we can see that for the same bit-rate, it achieves lower MSE, which is what we would have expected to get.

Example 3:

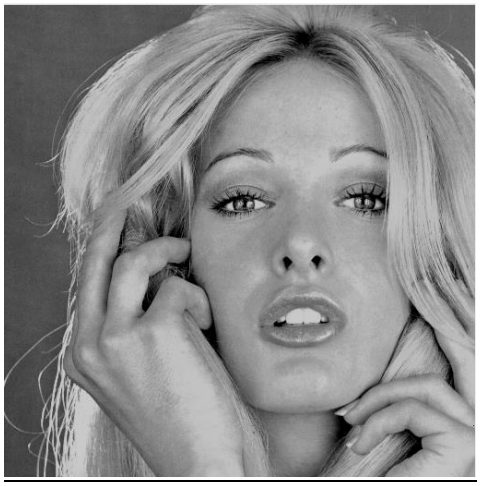


Figure 15:Original Image

Lambda: 0.001 R: 7.082 D: 11.0268

Lambda: 0.01833 R: 4.9104 D: 37.6583

Lambda: 0.33598 R: 2.9092 D: 58.445

Lambda: 6.1585 R: 1.8204 D: 59.4429

.ambda: 112.8838 R: 0.93872 D: 102.820

ambda: 5455.5948 R: 0.16248 D: 439.432

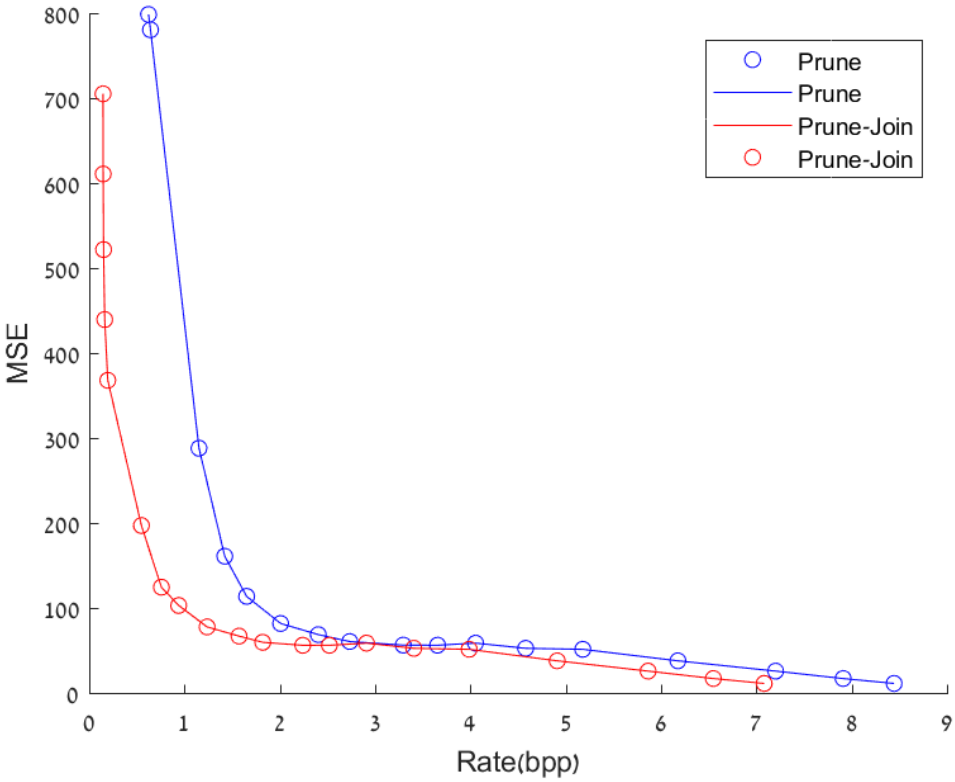


Figure 16:Rate Distortion Graph

Just like in the previous two example, we can see that the join-phase added to the prune algorithm improves performance.

Comparison to previous project results:

In our previous signal compression project we studied the paper by Shusterman and Feder [2].
In the analysis of our results in this project, we wanted to see how and if the paper we study this time suggests a more efficient algorithm.

The expected conclusion is that indeed the Prune-Join algorithm (and also the prune algorithm) is better than the algorithm in [2].
The methods presented in [1] are much better than the approach in [2]. This improved performance of the method from [1] is due to the utilization of Lagrangian rate-distortion optimization over the tree-structured decompositions, and the ability to join segments in a more flexible way.

This can be seen in the following graph:

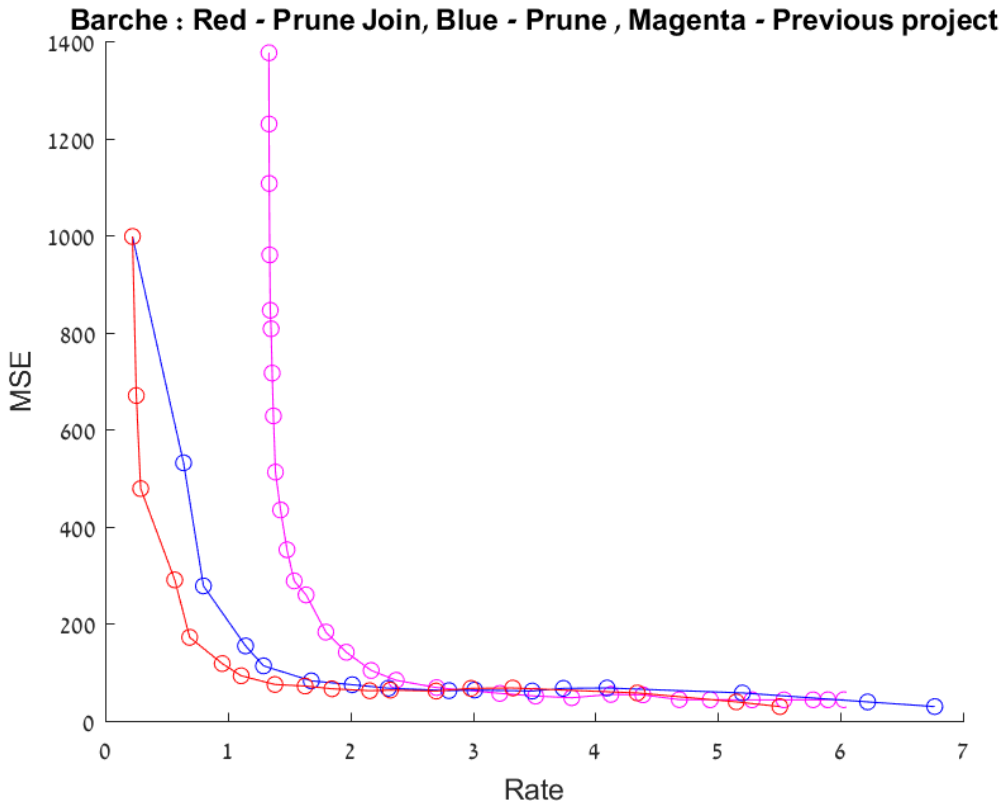


Figure 17: Rate Distortion Graph

Summary

We have reviewed the tree-based Prune algorithm and discussed its advantages and disadvantages.
Then we moved to the Prune-Join algorithm which views the join-scheme as an additional step to the prune algorithm.
We discussed the necessity of such algorithm – to exploit neighboring blocks in the decomposed signal which share similarity and their joint cannot be done in the original algorithm, due to the structure of the tree.

Finally, we implemented both the algorithms for 1D and 2D signals and compared between the two. The comparison was based on the R-D performance of the methods.

In our previous project, we also dealt with compression, and, more specifically quadtrees. One of our main issues with this data-structure was the lack of ability to exploit similarities between neighboring blocks which could not be joint due to their geometric shape that could not be represented in a standard quadtree.
Therefore, this project settled down this issue for us.

References:

[1] R. Shukla, P. L. Dragotti, M. N. Do, and M. Vetterli, "Rate-distortion optimized tree-structured compression algorithms for piecewise polynomial images," IEEE Transactions on Image Processing, 2005.
[2] E. Shusterman and M. Feder, "Image compression via improved quadtree decomposition algorithms," IEEE Transactions on Image Processing, 1994. by E. Shusterman and M. Feder