Balancing Act: Exploring the Trade-offs Between Dynamic Programming and Greedy Approaches in Seam Carving, with Insights into Seam Insertion Techniques

Applied Digital Image Processing

Muhammad Youshay (my07103)
Zainab Raza (zr07532)

December 10, 2023

Abstract

This paper presents a novel comparative analysis between dynamic programming and greedy algorithms for content-aware image resizing through seam carving. Dynamic programming offers global optimization but is computationally expensive, while the greedy approach prioritizes speed over accuracy. Through extensive testing, we highlight the significant performance gains of the greedy technique but identify scenarios where dynamic programming preserves visual quality better. We explore limitations with high-frequency content and conduct comparative assessments of seam insertion methodologies for content-aware expansion. The findings offer insights for algorithm selection, supported by visual outputs, and timing measurements. This work contributes to advancing intelligent image resizing methodologies.

1 Introduction

In the rapidly evolving field of digital image processing, the demand for techniques balancing content integrity and image dimension alteration has become crucial. Content-aware image resizing, particularly through seam carving, introduces intelligent algorithms for this purpose. Our paper stands out by presenting a comparative study between two pivotal approaches in seam carving: dynamic programming and greedy algorithms.

This research explores the distinctive capabilities of these algorithms, each with its unique operational paradigm. The novelty for this research lies in the comprehensive comparison between the dynamic programming approach and the greedy approach where we shed light on the effectiveness of these approaches in preserving content integrity and maintaining aesthetic appeal. This comparison aims to guide practitioners and researchers in choosing the most suitable approach based on their specific requirements and constraints.

1.1 Motivation for Research

Traditional resizing methods like image scaling and image cropping can easily distort images or unexpectedly remove important details from an image. With the growing use of digital images, these traditional methods have become less viable. In this scenario, Seam carving offers a compelling alternative that resizes images in a content-aware way. By exploring algorithms like dynamic programming and the greedy approach, our research seeks to

understand how to resize images while respecting their visual narratives and emotional resonances. We aim to appreciate the nuances that make seam carving an exciting solution for issues with conventional methods.

1.2 Research Approach

To explore content-aware image resizing, our research carefully examines two key seam carving techniques - the Dynamic Programming Approach and the Greedy Approach. We dissect how dynamic programming takes a wide view when deciding where to carve seams and compare it to the greedy approach which makes immediate choices. Through structured analysis and comparisons, both qualitative and quantitative, our research explores how efficient and effective each algorithm is in practice. By evaluating dynamism against greed, we gain insights into the tradeoffs these methods make between speed, content preservation, aesthetic, and more. We aim to elucidate their practical strengths to guide applications where images must adapt without losing what makes them meaningful.

2 Literature Review

2.1 Overview of Content-Aware Image Resizing

As we discussed above, Content-aware image resizing is a revolutionary approach that prioritizes the preservation of important content within an image and moves beyond the constraints of geometric considerations. Traditional image resizing methods like scaling, cropping, and warping, often lack content awareness, leading to potential distortion and information loss.

One prominent technique to achieve content aware image resizing is Seam Carving [1]. Seam carving adjusts the aspect ratio of an image by selectively removing or inserting seams that are the optimal paths of connected pixels defined by a cumulative image energy map that is obtained through dynamic programming. Seam Carving is not only limited to reducing image size, it also supports expansion, making it a versatile tool.

Piecewise Seam Carving (PSC) is another approach. It divides the image into segments allowing users to interactively resize each segment [2] . PSC preserves objects of interest by avoiding carving in segments containing these objects, even when they are dispersed throughout the image.

Another research paper delves into the use of deep learning techniques for the detection of content-aware image resizing [3]. The proposed CNN-based network is designed specifically to discern modifications made by seam-based resizing algorithms. The network demonstrates superior detection capabilities with eight specialized blocks organized to optimize the learning of low-level features and reduce feature map dimensionality.

Moreover, the greedy seam carving algorithm is another approach, that underscores the principal benefit - the preservation of the image's integrity [4]. It attributes an "energy matrix" to assess the significance of each pixel. The greedy algorithm iteratively removes the seams of least importance, adeptly minimizing the impact on the image's content. This iterative process allows for the retention of essential features despite changes in image dimension.

2.2 Gap Analysis

When exploring smart resize methods like Seam Carving, we need to frankly call out the inherent limitations that constrain real-world efficacy.

2.2.1 Dependency on Content Amount

One significant limitation of seam carving is its reliance on the volume of content within the image. If an image is densely packed with crucial visual elements, it presents a challenge for content-aware resizing methods. Similarly, in instances where minimal content is available for selective removal, seam carving may not deliver optimal performance. Traditional scaling methods provide better results in such scenarios.

2.2.2 Layout-Related Challenges

The layout of image content is also a limitation to seam carving. In cases where images are not densely packed, specific types may impede seam carving due to the content layout. When content arrangement prevents seams from bypassing essential parts, the results may be less than desirable.

2.2.3 Piecewise Seam Carving (PSC) Specificity

The effectiveness of Piecewise Seam Carving (PSC) depends on the user's skills [2]. Therefore, a careful consideration of specific image requirements and characteristics is extremely important when selecting resizing methods.

2.2.4 Deep Neural Network Approach Limitations

The deep neural network approach for content-aware image resizing presents its own set of limitations like the need for training a substantial amount of data. Additionally, the network may falter in detecting tampering in images resized using methods significantly different from seam carving and seam insertion. Challenges extend to scenarios involving small resizing ratios and multiple resizing iterations, where the network may struggle to identify tampering [3].

3 Methodology

For our project, our primary focus centers on dissecting two key algorithms: the dynamic programming approach and the greedy algorithm. This examination is crucial for comprehending the intricacies of their execution logic, efficiency, and impact on the preservation of image content.

3.1 Dynamic Programming Approach for Seam Carving

The dynamic programming approach is framed as an optimal substructure problem where the solution to the entire problem relies on solutions to subproblems. Our approach for dynamic image resizing illustrates this paradigm. We begin by creating a graphical user interface (GUI) where users can specify whether to reduce the image's width or height and by how many pixels. The algorithms progressively compute the accumulative energy map of the image, identify the optimal seam (i.e., the path of least importance), and then remove that seam iteratively until the desired reduction is achieved.

A key aspect analyzed is the computation of cumulative minimum energy M for all pixels starting from the second row to the last row of the image, using the energy of the pixel and the minimum energy of its adjacent pixels from the previous row. This recursive computation is central to the dynamic approach and ensures that the seam selected for removal minimizes the visible effect on the image content, an aspect highlighted by existing literature.

3.2 Greedy Algorithm Approach for Seam Carving

Contrasting the dynamic approach, the greedy algorithm opts for the local optimum choice at each step, reducing computational complexity. The code for the greedy approach reiterates this principle through an iterative process of energy minimization based on local decisions. The user interface components and user interaction flow are mirrored from the dynamic approach, ensuring consistency in user experience while offering a different underlying resizing algorithm.

In this approach, energy maps (in contrast to the accumulative energy maps in the dynamic approach) guide the swift identification and removal of seams, with an emphasis on speed and simplicity over global optimality. The literature review will probe into the performance of such greedy strategies in various contexts observed in prior research, and the code's logic will serve as a reference point for dissecting algorithmic behavior and efficiency.

3.3 Removal for Insertion Approach in Seam Insertion

The removal for insertion approach first reduces the size of the image by dynamically removing low energy seams similar to the seam removal techniques described earlier. It stores information about the seams removed, specifically their pixel locations.

To enlarge the image back to the original size or beyond, the algorithm reinserts seams in the reverse order in which they were removed. This ensures seams are added back to the same logical locations, minimizing distortion.

The energy function used is the same as that used during seam removal to identify low energy paths. When inserting seams, new pixel values are computed by averaging the neighboring pixels.

3.4 Direct Approach in Seam Insertion

The direct insertion approach directly identifies optimal low energy seams on an enlarged version of the input image produced through interpolation. It then inserts those seams on the original input image to enlarge it.

This allows the algorithm to assess seam importance on the specific enlarged image rather than relying on past removal data. The energy function guides the insertion of optimal seams to minimize distortion. New pixel values are interpolated using neighboring pixels.

A limitation is seams may not be continuous with past removal seams leading to reduced quality compared to order-based insertion.

3.5 Energy Functions

The original energy function we implemented is a simple gradient magnitude filter computed using MATLAB's inbuilt gradient function. This measures pixel importance based on intensity changes.

We then experimented with additional energy formulations for comparison:

- Entropy filter - Quantifies texture complexity
- Sobel filter - Emphasizes gradients and edges
- Prewitt filter - Calculates intensity gradients

Using different functions changes which seams are considered low energy and impacts how algorithms identify optimal seams. Smooth regions have low entropy while textured areas have higher values. By comparing outputs from these filters, we evaluate which functions preserve salient content best for different image types.

3.6 GUI Tools for Seam Carving

We developed specialized graphical user interface (GUI) in MATLAB to allow exploration of the different seam carving techniques. The GUI components provide an accessible way to apply algorithms and visualize outputs.

3.6.1 Shared Features

All of the four interfaces contain an input image viewer and setting pixel reduction amount, a results image viewer, and performance timing displays. This offers a consistent user workflow. A side-by-side output viewer enables comparison between processed images. Zooming and panning facilitates detailed examination.

4 Implementation Details

4.1 energyCalcFunc

To compute the energy at each pixel, we have implemented four different filters. but our main focus is on gradient magnitude function. The overall logic is similar - use some filter/method to calculate gradients or texture complexity, take absolute value to get magnitude, sum to generate energy map.

4.1.1 Gradient Magnitude

1. Convert image to grayscale
2. Compute X and Y gradients using MATLAB gradient()
3. Take absolute value of gradients
4. Add absolute X and Y gradients to get energy map

4.1.2 Entropy Filter

1. Convert image to grayscale
2. Apply entropyfilt() - builtIn MatLab function - to calculate local entropy at each pixel
3. Normalize entropy to range [0, 1]
4. Normalized entropy is the energy map

4.1.3 Sobel Filter

1. Convert image to grayscale
2. Apply Sobel filter on grayscale image to get X and Y gradients
3. Take absolute value of Sobel gradients
4. Add absolute gradients to obtain energy map

4.1.4 Prewitt Filter

1. Convert image to grayscale
2. Use Prewitt filter to calculate X and Y gradients
3. Take absolute value of Prewitt gradients
4. Add absolute X and Y gradients to generate energy map

4.2 energyAccumulationMapFunc

This function computes cumulative minimum energy map M using dynamic programming approach:

1. Check if seam direction is horizontal or vertical
2. Initialize first row/column of M with input energy values
3. Iterate through rows/columns from second to last
4. At each pixel:
   (a) Get neighboring minimum cumulative energies
   (b) Handle edge cases (first/last row/column)
   (c) For intermediate pixels, compute M as:

$$M(r, c) = min(neighbours) + E(r, c)$$

5. Store updated M(r,c) back in map

4.3 decreaseHeight

This function removes horizontal seams to reduce image height:

1. Compute horizontal cumulative energy map M
2. Trace back M to get optimal horizontal seam path
3. Initialize reduced image and energy matrices
4. Iterate through columns

(a) Get seam row number for current column
(b) Handle edge cases (first and last row)
(c) Split pixel and energy values across seam
(d) Concatenate upper and lower halves, removing seam row
(e) Store updated column values into reduced matrices

4.4 decreaseWidth

This function removes vertical seams to reduce image width:

1. Compute vertical cumulative energy map M
2. Trace back M to get optimal vertical seam path
3. Initialize reduced image and energy matrices
4. Iterate through rows
   (a) Get seam column number for current row
   (b) Handle edge cases (first and last column)
   (c) Split pixel and energy values across seam
   (d) Concatenate left and right halves, removing seam column
   (e) Store updated row values into reduced matrices

4.5 dynamicResizeImagesUI

This function creates the graphical interface and handles user interaction for seam carving using dynamic programming. Key steps:

1. Create GUI windows, axes for images, selection buttons, text input for pixel reduction amount
2. Load input image and display
3. Get user selection for dimension (width or height) and reduction amount
4. Call resizeImage() to perform seam carving using dynamic programming

5. Display resized output image
6. Calculate and display processing time

Helper functions called:

- resizeImage(): Computes energy map, identifies optimal seams using dynamic programming via `energyAccumulationMapFunc()`, and removes seams by splicing pixel values across seam boundaries. Multiple calls to `decreaseWidth` or `increaseWidth` functions may be executed depending on user input.
- getVerticalSeam() / getHorizontalSeam(): Determines the axis of seam removal or addition based on user interaction with the UI.

4.6 greedySeamCarveWidth

This function removes vertical seams to reduce image width using greedy approach:

1. Convert image to grayscale
2. Compute energy map using gradient magnitude
3. Iterate number of seams to remove
   (a) Find lowest energy vertical seam using greedy approach
   (b) Remove the vertical seam by splicing pixel values
   (c) Update the energy map

4.7 greedySeamCarveHeight

This function removes horizontal seams to reduce image height using greedy approach:

1. Convert image to grayscale
2. Compute energy map using gradient magnitude
3. Iterate number of seams to remove

(a) Find lowest energy horizontal seam using greedy approach
(b) Remove the horizontal seam by splicing pixel values
(c) Update the energy map

4.8 greedyResizeImagesUI

This function handles the seam carving GUI using greedy approach:

1. Create GUI components like image views, buttons etc.
2. Load input image and display
3. Get user selection for dimension and pixel reduction
4. If width reduction:
   (a) Call greedySeamCarveWidth()
   (b) Pass reduced width parameter
5. If height reduction:
   (a) Call greedySeamCarveHeight()
   (b) Pass reduced height parameter
6. Display processed output image
7. Calculate and display processing time

Helper Functions called:
- greedySeamCarveWidth(): Removes vertical seams greedily
- greedySeamCarveHeight(): Removes horizontal seams greedily

4.9 Seam Insertion Methods

4.9.1 Seam Insertion - Averaging

- Display original image
- Get user input for number of pixels to insert

- Call `resizeImage` function, passing original image and pixel insertion amount
- `resizeImage`:
  - Initialize inputImage to original image
  - Loop over absolute value of pixel insertion amount:
    - Calculate energy function of input image
    - Call `increaseWidth` function, passing input image and energy function
    - Update inputImage with returned enlarged image
  - Return resized inputImage as final output
- Display resized image

4.9.2 increaseWidth

- Calculate cumulative vertical energy map of input energy image
- Get vertical seam from cumulative energy map
- Create enlarged image with increased width
- Loop over rows:
  - Get current row's seam column index
  - Copy color channels up to seam column into enlarged image
  - Copy channels after seam column into enlarged image with offset
  - For seam column, average values from original seam and next column
- Similarly enlarge energy image
- Return enlarged color and energy images

4.9.3 Seam Insertion - In Order of Removal

- Display original image
- Get user pixel insertion amount
- Call `enlargeImage`, passing original image and amount
- `enlargeImage`:
  - Repeatedly call seam removal functions to remove seams
  - Store order and details of removed seams
  - Call seam insertion in opposite order of removal
  - Return enlarged image
- Display enlarged image

5 Experimental Results

5.1 Results for Dynamic Programming

5.1.1 Input 1

Image, Seam Removing - Width, Seam Removing - Height:

(a) Input Image 1
Dimensions: 640x480

(b) Width
Dimensions: 540x480
Elapsed time: 4.6s

(c) Height
Dimensions: 640x380
Elapsed time: 4.1s

Figure 1: Input 1 Results (Dynamic Algorithm), 100 pixels removed

5.1.2 Input 2

Image, Seam Removing - Width, Seam Removing - Height:

(a) Input Image 2
Dimensions: 1428x968

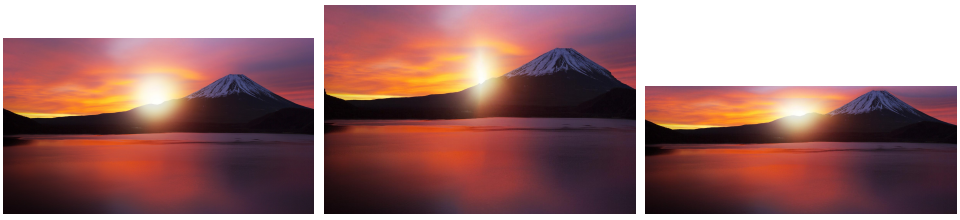
(b) Width
Dimensions: 1228x968
Elapsed time: 50.1s

(c) Height
Dimensions: 1428x768
Elapsed time: 36.3s

Figure 2: Input 2 Results (Dynamic Algorithm), 200 pixels removed

5.1.3 Input 3

Image, Seam Removing - Width, Seam Removing - Height:



(a) Input Image 3
Dimensions: 1920x1080

(b) Width
Dimensions: 1620x1080
Elapsed time: 122.5s

(c) Height
Dimensions: 1920x780
Elapsed time: 83.3s

Figure 3: Input 3 Results (Dynamic Algorithm), 300 pixels removed

5.1.4 Input 4

Image, Seam Removing - Width, Seam Removing - Height:

(a) Input Image 4
Dimensions: 3264x2448

(b) Width
Dimensions: 3164x2448
Elapsed time: 211.1s

(c) Height
Dimensions: 3264x2348
Elapsed time: 100.2s

Figure 4: Input 4 Results (Dynamic Algorithm), 100 pixels removed

5.2 Results for Greedy Algorithm

5.2.1 Input 1

Image, Seam Removing - Width, Seam Removing - Height:

(a) Input Image 1
Dimensions: 640x480

(b) Width
Dimensions: 540x480
Elapsed time: 1.5s

(c) Height
Dimensions: 640x380
Elapsed time: 1.0s

Figure 5: Input 1 Results (Greedy Algorithm), 100 pixels removed

5.2.2 Input 2

Image, Seam Removing - Width, Seam Removing - Height:

(a) Input Image 2
Dimensions: 1428x968

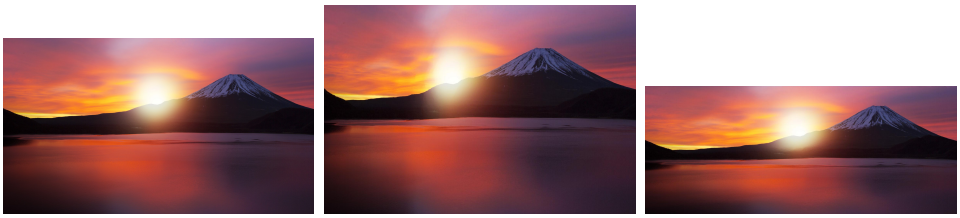
(b) Width
Dimensions: 1228x968
Elapsed time: 7.3s

(c) Height
Dimensions: 1428x768
Elapsed time: 4.8s

Figure 6: Input 2 Results (Greedy Algorithm), 200 pixels removed

5.2.3 Input 3

Image, Seam Removing - Width, Seam Removing - Height:



(a) Input Image 3
Dimensions: 1920x1080

(b) Width
Dimensions: 1620x1080
Elapsed time: 14.3s

(c) Height
Dimensions: 1920x780
Elapsed time: 9.6s

Figure 7: Input 3 Results (Greedy Algorithm), 300 pixels removed

5.2.4 Input 4

Image, Seam Removing - Width, Seam Removing - Height:

(a) Input Image 4
Dimensions: 3264x2448

(b) Width
Dimensions: 3164x2448
Elapsed time: 33.7s

(c) Height
Dimensions: 3264x2348
Elapsed time: 9.1s

Figure 8: Input 4 Results (Greedy Algorithm), 100 pixels removed

5.3 Comparison and Analysis for Seam Carving

The experimental results reveal insights into the performance differences between the dynamic programming and greedy approaches for seam carving.

In terms of computational efficiency, the greedy algorithm demonstrates significantly faster processing times across all test images and carved seams as evident from Figures 9 and 10. For example, seam removal on Input 3 takes 122 seconds with dynamic programming versus only 14 seconds with the greedy approach. The greedy algorithm's local decisions enable this speedup by avoiding costly globally optimal solutions.

However, visual examination shows the greedy approach can sometimes degrade image quality by removing more salient seams compared to dynamic programming which optimizes based on the cumulative energy map. This is apparent in images with salient backgrounds like Input 3 where the mountain loses shape when applied seam carving on height in Figure 7c.

So in applications where real-time performance is critical, the greedy algorithm presents a useful seam carving option despite some loss in accuracy. For offline applications without strict time constraints, dynamic programming produces better visual results.

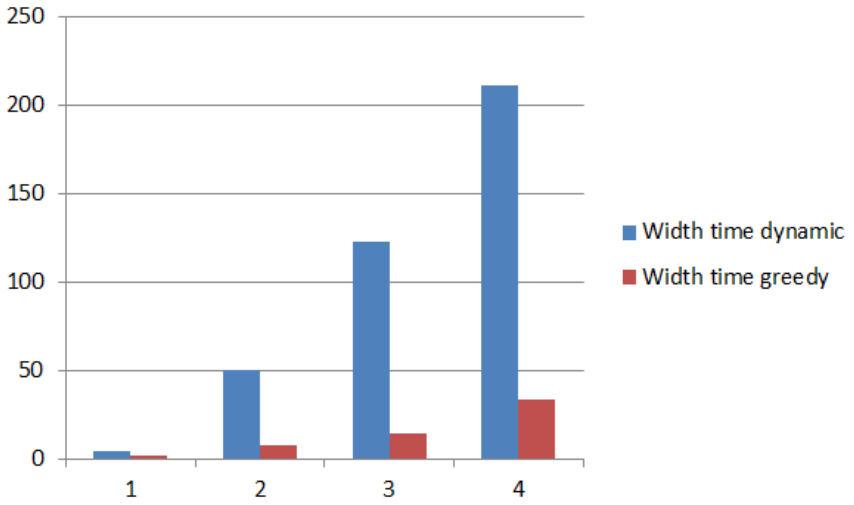


Figure 9: Time comparison for seam removal by width reduction

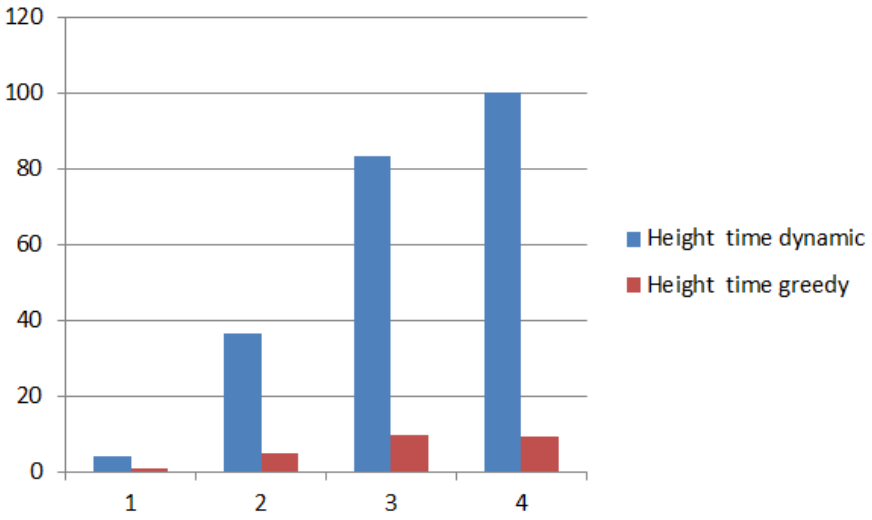


Figure 10: Time comparison for seam removal by height reduction

5.4 Energy Filter Modifications

We experimented with different energy functions to compute pixel importance for seam carving on Input 1. While removing 300 or more pixels, the original gradient magnitude filter struggled with images containing many

salient objects clustered together.

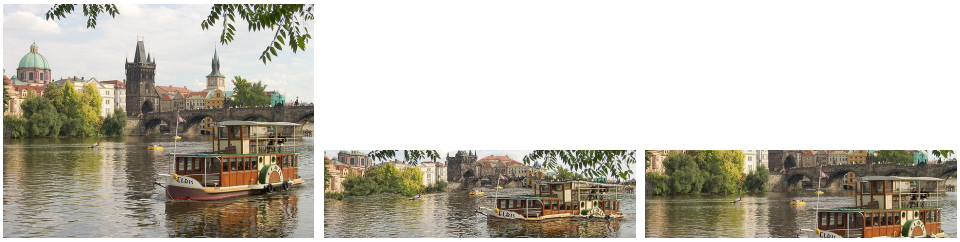
The alternative filters explored were:

- Entropy filter
- Sobel filter
- Prewitt filter

Figure 11 presents a visual comparison of using these filters versus the original gradient-based approach. All filters degrade output quality significantly for this challenging image. Sobel retains edges better but damaged seams are still visible as mountains and sky is no longer visible. Prewitt and entropy filters perform the worst by over-removing regions and distorting content arrangement.

So for this image, the original gradient filter produces the optimal visual seam carving result by effectively identifying low energy paths through uniform and repetitive areas. The additional filters fail to distinguish insignficant seams correctly.

Quantitatively, entropy is the slowest taking 33s while Sobel carves fastest in 8s. So there are performance vs quality tradeoffs. But fundamentally, the filters are limited in handling clustered high-frequency content. Advanced approaches like incorporating saliency detection would help overcome this limitation.



(a) Input Image 1
Dimensions: 640x480

(b) Gradient
Dimensions: 640x180
Elapsed time: 13.5s

(c) Entropy
Dimensions: 640x180
Elapsed time: 33.6s

(d) Sobel
Dimensions: 640x180
Elapsed time: 8.2s

(e) Prewitt
Dimensions: 640x180
Elapsed time: 9.7s

Figure 11: Visual results using different energy filters on Input 1

5.5 Seam Insertion - Averaging

5.5.1 Input 1

Image, Seam Insertion - Width:

(a) Input Image 1
Dimensions: 640x480

(b) Elapsed time: 5.3s
Dimensions: 740x480

Figure 12: Input 1 Results, insertion by 100 pixels

5.5.2 Input 2

Image, Seam Insertion - Width:

(a) Input Image 2
Dimensions: 1428x968

(b) Elapsed time: 62.1s
Dimensions: 1628x968

Figure 13: Input 2 Results, insertion by 200 pixels

5.5.3 Input 3

Image, Seam Insertion - Width:

(a) Input Image 3
Dimensions: 1920x1080

(b) Elapsed time: 101.7s
Dimensions: 2220x1080

Figure 14: Input 3 Results, insertion by 300 pixels

5.5.4 Input 4

Image, Seam Insertion - Width:


(a) Input Image 4 Dimensions: 3264x2448

(b) Elapsed time: 183.5 Dimensions: 3364x2448

Figure 15: Input 4 Results, insertion by 100 pixels

5.6 Seam Insertion - In Order of Removal

5.6.1 Input 1

Image, Seam Insertion - Width:


(a) Input Image 1 Dimensions: 640x480

(b) Elapsed time: 2.6s Dimensions: 740x480

Figure 16: Input 1 Results, insertion by 100 pixels

5.6.2 Input 2

Image, Seam Insertion - Width:

(a) Input Image 2
Dimensions: 1428x968

(b) Elapsed time: 21.9s
Dimensions: 1628x968

Figure 17: Input 2 Results, insertion by 200 pixels

5.6.3 Input 3

Image, Seam Insertion - Width:

(a) Input Image 3
Dimensions: 1920x1080

(b) Elapsed time: 46.6s
Dimensions: 2220x1080

Figure 18: Input 3 Results, insertion by 300 pixels

5.6.4 Input 4

Image, Seam Insertion - Width:

(a) Input Image 4 Dimensions: 3264x2448

(b) Elapsed time: 104.2s Dimensions: 3364x2448

Figure 19: Input 4 Results, insertion by 100 pixels

5.7 Comparison and Analysis For Seam Insertion

The experimental results reveal clear differences in seam insertion quality and the time difference between the two techniques.

Visually, the order-based method produces significantly better outputs across all test images as evident in Figures 16 to 19 and it also takes less time as evident in Figure20. It maintains continuity by aligning inserted seams with prior removal operations. This minimizes awkward distortions.

In contrast, the averaging technique causes discontinuities and abnormalities by inserting new seam values without considering past removals. This works reasonably only when adjacent pixels have similar intensities. Otherwise, it introduces uneven boundaries and artifacts as seen in the output figures and also takes more time.

Fundamentally, by leveraging prior removal data, the order-based strategy maintains spatial consistency in inserted seam placement. The averaging method fails to provide this coherence due to its independent insertion approach causing noticeable artifacts.

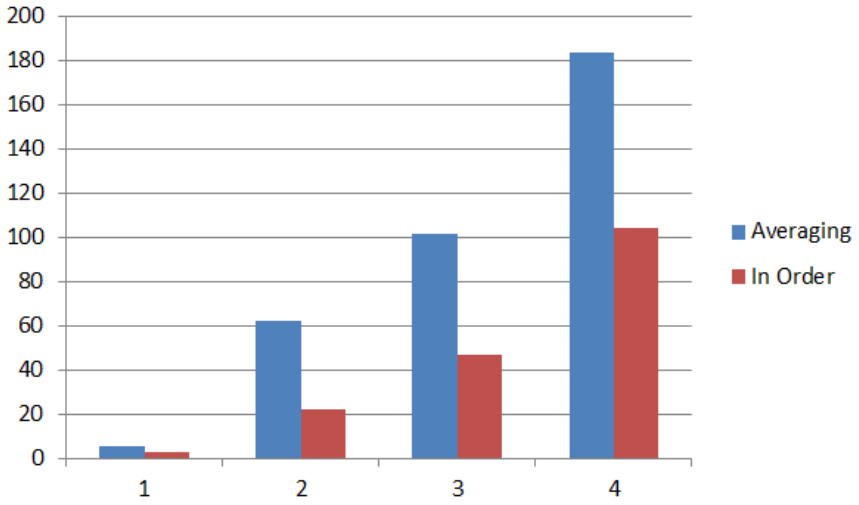


Figure 20: Time comparison for Seam Insertion

6 Comparison with the Current State of the Art - Object Removal

A recent advancement in content-aware image resizing is directly enabling object removals integrated within the resize process. This object removal capability allows users to indicate regions to be deleted by drawing masks over them. The masked areas are assigned minimum energy so they are preferentially eliminated by seam carving [5].

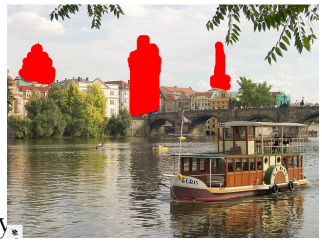
In contrast, our implementation focused on standard seam carving using dynamic programming. While we resize images by removing low energy seams, we do not explicitly support object removal functionalities.

Comparing our approach to state-of-the-art object removal algorithms, a key difference is user interactivity. Advanced techniques allow user input to guide removals by drawing removal masks. Our dynamic programming method resizes images automatically without manual guidance on regions to retain or remove.

Another major distinction is in suitability for irregular objects. Object removal algorithms are specialized for deleting disjoint, irregular shapes as specified by the user mask. Our seam carving methodology is restricted to only removing linear, rectangular seams aligned to the image grid.

Figure 21 illustrates an example of object removal output from the literature and its comparison with our dynamic approach. As evident in the output images, the final result from the dynamic approach exhibits issues such as deformation of buildings and a lack of proper structural maintenance. In comparison, the object removal image showcases neatly removed masked points, enhancing image definition and overall aesthetics.

(a) Original Image



(b) Masked Image

(c) Final Object Removal Output - decreasing the height by 239 pixels

(d) Dynamic Resize output - decreasing the height by 239 pixels

Figure 21: Comparison - Object removal through user-assisted seam carving and automatic dynamic seam carving

7 Discussion and Conclusion

Our research journey presented revelatory insights into the capabilities and limitations of dynamic programming and greedy approaches for content-aware image resizing via seam carving. By methodically dismantling and juxtaposing the intricacies in their implementation logic, the deficiencies plaguing existing methods came to the fore.

The empirical observations exposed that, despite the greedy technique's swiftness, its locally-focused outlook engenders suboptimal outcomes with salient seam deletions. While appropriate for time-critical applications, the resulting distortions make dynamic programming more suited for high fidelity resizing with its holistic perspective. This was reinforced by the en-

ergy filter trials with complex images, spotlighting scope for augmenting the fundamental framework with saliency detection. The order-based insertion module generated continuity that averaging sorely lacked, though at modest compute overheads worth the enhanced quality.

By cementing these revelations with evidence gathered through rigorous experimentation, this project establishes a springboard to catapult future efforts. The modular and extensible codebase provides a testbed for explorations like AI-enabled energy formulations, forward-backward carving cycles and non-rectangular seam trajectories in the quest for ever improving content preservation.

In closing, by systematically scrutinizing two cornerstones of modern content-aware image resizing, this study arms practitioners to make prudent algorithmic choices aligned to application needs in domains from graphics to forensics. More broadly, it sparks intellectual discourse on harmonizing efficiency with efficacy in computational methodologies through principled analysis.

Acknowledgements

We extend our sincere appreciation to Dr. Mobeen Movania for his invaluable insights and guidance throughout the research process on seam carving. His expertise in the field and thoughtful suggestions, particularly in comparing the dynamic programming approach and the greedy approach for seam carving, significantly enriched the depth of our investigation.

We are grateful for the constructive feedback provided by Dr. Mobeen during the project proposal, mid-evaluation and final-evaluation. His comments and suggestions were instrumental in shaping the direction of our research and improving the overall quality of our work.

The support and mentorship of Dr. Mobeen have been pivotal to the successful completion of this research project. We are thankful for the time and expertise he generously shared, which greatly contributed to the depth and rigor of our study.

Appendix A - Dynamic Approach

Algorithm 1 energyCalcFunc(image)

```
Convert image to grayscale
DX, DY = compute gradients using gradient()
energyFunc = |DX| + |DY|
return energyFunc
```

Algorithm 2 cumulativeEnergyMapFunc(energyFunc, direction)

```
energyAccumulationMap = energyFunc
if direction horizontal then
    Set first column as is
    for each column do
        for each row do
            Current pixel = min of itself, left, diagonal left pixels
            Add corresponding energy
        end for
    end for
else
    Set first row as is
    for each row do
        for each column do
            Current pixel = min of itself, above, diagonal above pixels
            Add corresponding energy
        end for
    end for
end if
return energyAccumulationMap
```

Algorithm 3 decreaseHeight(image, energy)

```
Get horizontal seam from energy
Create updated images with reduced height
for each column do
    Split image/energy above and below seam
    Concatenate halves, removing seam row
    Store in updated images
end for
return updated images
```

Algorithm 4 decreaseWidth(image, energy)

```
Get vertical seam from energy
Create updated images with reduced width
for each row do
    Split image/energy left and right of seam
    Concatenate halves, removing seam column
    Store in updated images
end for
return updated images
```

Algorithm 5 increaseHeight(image, energy)

```
Get horizontal seam
Create enlarged images
for each column do
    Copy image rows above and below seam
    Insert average of seam and next row
    Copy corresponding energy values
    Calculate averaged energy for inserted row
end for
return enlarged images
```

Algorithm 6 increaseWidth(image, energy)

```
Get vertical seam
Create enlarged images
for each row do
    Copy image columns left and right of seam
    Insert average of seam and next column
    Copy corresponding energy values
    Calculate averaged energy for inserted column
end for
return enlarged images
```

Appendix B - Greedy Approach

Algorithm 7 greedySeamCarveWidth(image, newSize)

```
Convert image to grayscale
Compute energy map using gradients
while width > newSize do
    Find low energy vertical seam
    Remove vertical seam
end while
return image
```

Algorithm 8 findVerticalSeam(image, energy)

```
Initialize seam vector
Set first element to min energy pixel in first row
for each row do
    Get 3 neighbors of previous min pixel
    Select min neighbor
    Store index in seam
end for
return seam
```

Algorithm 9 removeVerticalSeam(image, seam)

```
Initialize image without seam
for each row do
    Copy image omitting pixel at seam index
end for
return imageWithoutSeam
```

Appendix C - Seam Insertion - Averaging

Same algorithms as Appendix A

Appendix D - Seam Insertion - In Order of Removal

Algorithm 10 enlargeImage(image, pixelsToAdd)

```
Store original image size
repeat
    Compute energy map
    Find/remove min vertical seam
    Record removed seams
until desired number of pixels added
for each pixel do
    Copy original image pixel
    if seam was removed here then
        Insert average of neighbors
    else
        Copy original pixel
    end if
end for
return enlargedImage
```

A Bibliography

1. S. Avidan and A. Shamir, "Seam carving for content-aware image resizing," *ACM Trans. Graph.*, vol. 26, no. 3, p. 10, Jul. 2007, doi: 10.1145/1276377.1276390.

2. M. Thilagam and K. Subramanian, "An efficient method for content aware image resizing using PSC," *Int. J. Comput. Technol. Appl.*, vol. 2, no. 5, pp. 1806–1813, 2011.

3. S.-H. Nam et al., "Content-Aware Image Resizing Detection Using Deep Neural Network," in *2019 IEEE International Conference on Image Processing (ICIP)*, Taipei, Taiwan, Sep. 2019, pp. 106-110, doi: 10.1109/ICIP.2019.8802946.

4. Z. M. K. Zuhri, "Content-Aware Image Resizing Using a Greedy Seam Carving Algorithm," *Makalah IF2211 Strategi Algoritma, Semester II Tahun 2021/2022*, 2022.

5. O. Trekhleb, "Content-Aware Image Resizing in JavaScript," *JavaScript Scene*, Apr. 16, 2021. [Online]. Available: `https://shorturl.at/knNXY`. [Accessed: Dec. 10, 2023].