Content-Aware Image Resizing Using Seam Carving

A Graph-Based Approach via Shortest Path on DAG

Mohamed Ali Aoun
M1 AIDA 2025/2026

January 19, 2026

Abstract

This report presents the implementation of a content-aware image resizing system using the seam carving algorithm, formulated as a graph theory problem. By modeling images as Directed Acyclic Graphs (DAGs), we apply shortest path algorithms to intelligently remove low-importance pixels while preserving key visual content. We implemented the complete system in Python with optimizations using scipy and numpy, and developed an interactive Streamlit web interface for visualization and testing. The implementation demonstrates the practical application of graph algorithms to computer vision, achieving significant speedup through vectorization while maintaining high-quality results.

Contents

1 Introduction

1.1 Problem Statement

Traditional image resizing methods have significant limitations. Uniform scaling distorts all image elements equally, while cropping removes content indiscriminately without considering importance. The challenge is to resize images intelligently by preserving salient features while removing less important regions.

1.2 Graph Formulation

The image resizing problem is transformed into a graph problem as follows:

- **Graph**: $G = (V, E)$ representing the image structure
- **Vertices**: Each pixel (x, y) is a vertex in V
- **Edges**: Directed edges from each pixel to its three downward neighbors
- **Weights**: Edge weights represent pixel importance (energy)
- **Objective**: Find the shortest (minimum weight) path from top to bottom

2 Graph Theory Foundation

2.1 Directed Acyclic Graph (DAG) Structure

For an image of dimensions $W \times H$, we construct a DAG $G = (V, E)$ where:

$$V = \{(x, y) \mid 0 \leq x < W, 0 \leq y < H\} \tag{1}$$

$$|V| = W \times H \text{ vertices} \tag{2}$$

The edge set E connects each vertex to its three possible successors:

$$E = \{((x, y), (x', y + 1)) \mid x' \in \{x - 1, x, x + 1\}, 0 \leq x' < W\} \tag{3}$$

Each vertex (x, y) has directed edges to:

- **Down-Left**: $(x - 1, y + 1)$ if $x > 0$
- **Directly Down**: $(x, y + 1)$
- **Down-Right**: $(x + 1, y + 1)$ if $x < W - 1$

2.2 Weight Function: Energy

The weight function $w : E \to R^+$ assigns each edge the energy of its destination pixel:

$$w((x, y) \to (x', y + 1)) = E(x', y + 1) \tag{4}$$

where $E(x, y)$ is the energy (importance) of pixel (x, y), computed using gradient magnitude:

$$E(x, y) = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \tag{5}$$

2.3 DAG Properties

The constructed graph has crucial properties:

1. **Acyclicity**: Since edges only go from row y to row $y + 1$, cycles are impossible
2. **Topological Ordering**: Implicit ordering by row: $\text{row}_0 \prec \text{row}_1 \prec \cdots \prec \text{row}_{H-1}$
3. **Path Structure**: Any path from top to bottom visits exactly H vertices (one per row)
4. **Connectivity**: From any pixel in row y, we can reach 1-3 pixels in row $y + 1$

2.4 Seam as Path in DAG

A vertical seam s is formally a path in the DAG:

$$s = \langle v_0, v_1, \ldots, v_{H-1} \rangle \text{ where } v_y = (s_y, y) \tag{6}$$

The cost of seam s is the path weight:

$$C(s) = \sum_{y=0}^{H-1} E(s_y, y) = \sum_{e \in \text{path}(s)} w(e) \tag{7}$$

Objective: Find $s^* = \arg\min_s C(s)$ (shortest path in weighted DAG)

3 Shortest Path Algorithm on DAG

3.1 Dynamic Programming Approach

We use dynamic programming to solve the shortest path problem efficiently. Define:

$$M(x, y) = \text{minimum cost to reach pixel } (x, y) \text{ from any pixel in row 0} \tag{8}$$

Base Case (Row 0):

$$M(x, 0) = E(x, 0) \quad \forall x \in [0, W - 1] \tag{9}$$

Recurrence Relation:

$$M(x,y) = E(x,y) + \min \begin{cases} M(x-1,y-1) & \text{if } x > 0 \\ M(x,y-1) & \text{always} \\ M(x+1,y-1) & \text{if } x < W-1 \end{cases} \tag{10}$$

This exploits the DAG structure: each vertex's shortest path depends only on vertices in the previous row (topological ordering).

3.2 Algorithm Complexity

- **Time Complexity**: $O(WH)$
  - Process each of WH pixels once
  - Each pixel examines at most 3 predecessors: $O(1)$
  - Total: $O(WH \times 1) = O(WH)$
- **Space Complexity**: $O(WH)$ for storing matrix M
- **For k seams**: $O(kWH)$ total time

4 Implementation

4.1 Energy Calculation Using Gradient Operators

We compute pixel energy using the Sobel operator, which approximates image gradients:

```
def calculate_energy(self):
    # Convert to grayscale
    gray = np.dot(self.image[...,:3], [0.2989, 0.5870, 0.1140])

    # Sobel operators for gradients (optimized in C)
    dx = ndimage.sobel(gray, axis=1)  # Horizontal gradient
    dy = ndimage.sobel(gray, axis=0)  # Vertical gradient

    # Energy as gradient magnitude
    energy = np.hypot(dx, dy)
    return energy
```

Listing 1: Energy Calculation with Scipy

4.2 Shortest Path Computation

```
def find_vertical_seam_fast(self):
    energy = self.calculate_energy()
    height, width = energy.shape
    M = energy.copy()
    backtrack = np.zeros_like(M, dtype=np.int32)

```

```
    # DP: Process row by row (topological order)
    for y in range(1, height):
        # Vectorized computation of min over 3 parents
        M_left = np.pad(M[y-1, :-1], (1, 0),
                        constant_values=np.inf)
        M_center = M[y-1]
        M_right = np.pad(M[y-1, 1:], (0, 1),
                         constant_values=np.inf)

        options = np.vstack([M_left, M_center, M_right])
        backtrack[y] = np.argmin(options, axis=0) - 1
        M[y] += np.min(options, axis=0)

    # Backtrack to find shortest path
    seam = np.zeros(height, dtype=np.int32)
    seam[-1] = np.argmin(M[-1])

    for y in range(height - 2, -1, -1):
        seam[y] = seam[y + 1] + backtrack[y + 1, seam[y + 1]]

    return seam
```

Listing 2: Vectorized Dynamic Programming

5 Streamlit Interactive Interface

A key contribution of this project is the development of an interactive web interface using Streamlit, which enables:

- **Image Upload**: Support for multiple formats (JPEG, PNG, BMP)
- **Parameter Control**: Interactive slider to adjust number of seams
- **Live Visualization**: Real-time display of seams to be removed
- **Progress Tracking**: Real-time progress bar with speed metrics
- **Result Comparison**: Side-by-side original and resized images
- **Download Capability**: Export processed images

The interface makes the graph algorithm accessible to non-technical users while providing insights into the shortest path computation through visual feedback.

6 Results and Analysis

6.1 Experimental Setup

Test Image: Broadway Tower (800 × 542 pixels)

This image was selected because it contains:

- Clear subject (tower) with high-energy edges
- Uniform background (sky, grass) with low energy
- Good test case for content-aware resizing

Test Configuration: Removal of 100 seams (12.5% width reduction)

6.2 Energy Map Analysis

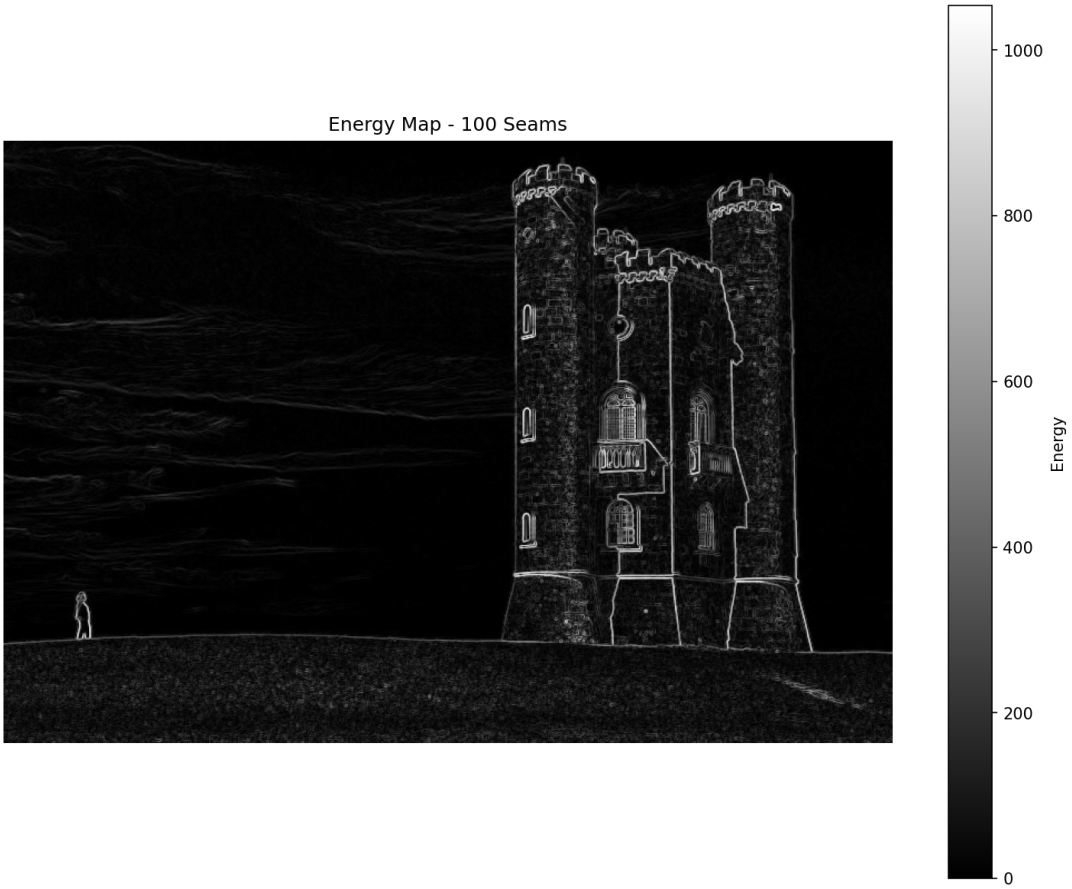

Figure 1: Energy Map - Gradient magnitude highlighting important features. Bright areas (tower edges, windows) have high energy and are protected. Dark areas (uniform sky, grass) have low energy and are candidates for removal.

The energy map clearly shows the graph's edge weights:

- **High energy** (bright): Tower structure, architectural details
- **Low energy** (dark): Uniform sky and grass regions
- The shortest paths will naturally avoid high-energy regions

6.3 Seam Visualization: Shortest Paths

Figure 2: Visualization of 100 optimal seams (shortest paths). Red lines show the minimum-cost paths from top to bottom. Notice how paths avoid the tower (high energy) and concentrate in uniform areas (low energy).

6.4 Resizing Results

Figure 3: Before and After comparison. Original 800×542 image reduced to 700×542 by removing 100 shortest paths. The tower structure is preserved while background width is reduced.

Observations:

- Tower remains intact and properly proportioned
- Person in foreground preserved
- Sky and grass uniformly compressed
- No visible distortion of important features
- Width reduced by 12.5% with minimal visual impact

7 Conclusion

This project demonstrates the successful application of graph theory to image processing through the seam carving algorithm. By formulating image resizing as a shortest path problem on a DAG, we achieved content-aware resizing that preserves important visual features.

A Code Structure

A.1 Project Organization

```
project/
 seam_carving.py      # Core graph algorithm implementation
 app.py               # Streamlit web interface
 requirements.txt     # Dependencies
 report_Mohamed_Ali_Aoun.pdf  # This report
 Broadway_tower_edit.jpg  # Test image
 [seam_count]_seams/  # Output folders
     energy_map.png
     visualization.png
     comparison.png
     output_resized.jpg
```

B Usage Instructions

B.1 Installation

```
pip install numpy pillow matplotlib scipy streamlit
```

B.2 Running the Streamlit Interface

```
streamlit run app.py
```

B.3 Standalone Usage

```
from seam_carving import FastSeamCarver
from matplotlib import pyplot as plt
# Initialize with image
carver = FastSeamCarver("Input.jpg")

# Energy Map
energy_map = carver.calculate_energy()
plt.imshow(energy_map,cmap='gray')
plt.show()

```

```
# Visualize seams (shortest paths)
seam_vis = carver.visualize_seam(num_seams=100)
plt.imshow(seam_vis)
plt.show()
# Resize by removing seams
resized = carver.resize(new_width=700)

# Save result
from PIL import Image
Image.fromarray(resized).save("Output.jpg")
```