Optimization Project 1: A New Tool for Image Segmentation Using Linear Programming

Oliver Gault, Sushanth Ravichandran, Gayathree Gopi, Brooks Li

Section 1: Intro

In this project, we developed a new image segmentation tool that leverages linear programming to optimally process small grayscale images by applying the max flow/min cut theorem to reformat them based on pixel intensities. This tool represents an exciting application of linear programming, efficiently handling both simple and complex images across a range of sizes. By modeling each image as a flow network, we treat the pixels as nodes and the edges as representations of the similarity between neighboring pixels. The segmentation process separates the image into foreground and background by identifying optimal cuts that minimize the "water flow" between different regions. By solving the max flow problem between the source (representing the background) and the sink (representing the foreground), the tool finds the most optimal cut for the image.

After testing various images, our team confidently recommends this tool as a powerful addition to our company's repertoire. Its performance surpasses Photoshop's Magic Wand tool, offering users a more precise and efficient solution. Furthermore, there is exciting potential to expand the tool's capabilities to handle larger images, incorporate more colors, and even work with complex digital artwork and video editing, making it a strong asset in attracting new users and improving overall performance.

Section 2: An Overview of Max Flow/Min Cut Theory as it applies to Image Processing

The Max Flow/Min Cut theory is often applied to image processing and segmentation. It states that the maximum amount of flow passing from the source to the sink is equal to the total weight of the edges in a minimum cut. The theory is often used in image segmentation due to the ability to treat each pixel in an image as a node in a graph, where the edges of the nodes represent the similarity between the adjacent pixels. High similarity means the capacity of the edge is low.

The minimum cut of an image is defined as the minimum sum of weights of edges that can be divided into two different sets when the image is removed from the graph. The maximum flow of an image is defined as the maximum amount of flow that the image is able to allow to flow from the source node to the sink. To make things simple, max flow is about finding how much water can flow through a network of pipes, while min cut is about finding the fewest number of pipes needed to stop the water from flowing.

In this project, we view every picture as a network of pixel similarities, calculated by their intensities

```
similarity = 100 * np.exp(-intensity_diff_squared / (2 * sigma ** 2))
```

, where

intensity_diff_squared is the squared difference between the two pixel intensities, and the overall similarity function is scaled by a factor of 100 to ensure a similarity range between 0 and 100. The larger the difference between the intensities of the pixels, the smaller the similarity score. We also add a source node and sink node to represent flow from the source node to one background pixel as well and one foreground pixel to the sink node, where the maximum flow rate for both is analogous to the largest similarity between any two pixels in the network.

```
source_node = num_pixels
sink_node = num_pixels + 1
```

We then formulate a linear programming problem with decision variables corresponding to each non-zero link in the matrix that represent the amount of flow between each node and constraints representing the sum of the flow into each node. The next step is to calculate the residual network, where each connection is the difference between the maximum possible flow and the actual flow between the nodes, cut off connections between nodes accessible from the source node and inaccessible nodes, and add the maximum flow rates to find the optimal cuts to make on the image. Finally, this model is applied to separate grayscale images from their backgrounds. Here are some screenshots below of how we defined the residual calculation, min cut, and max flow segmentation:

```
def dfs(residual_network, source_node):
    visited = set()
    stack = [source_node]
    while stack:
        node = stack.pop()
        if node not in visited:
            visited.add(node)
            # Get neighbors with positive residual capacity
            neighbors = [j for (i, j), capacity in residual_network.items() if i == node and capacity > 1e-5]
            stack.extend(neighbors)
    return visited
```

```python
def find_min_cut(network, flow, source_node):
    # Build the residual network
    residual_network = {}
    for (i, j), capacity in network.items():
        flow_value = flow.get((i, j), 0)
        residual_capacity = capacity - flow_value
        if residual_capacity > 1e-5:
            residual_network[(i, j)] = residual_capacity
    # Perform DFS to find all accessible nodes from the source in the residual network
    accessible_nodes = dfs(residual_network, source_node)
    # Identify cut edges in the original network
    num_nodes = max(max(edge) for edge in network.keys()) + 1
    cut_edges = []
    # Any edge from an accessible node to an inaccessible node is part of the min cut
    for node in accessible_nodes:
        for neighbor in range(num_nodes):
            if (node, neighbor) in network and neighbor not in accessible_nodes:
                cut_edges.append((node, neighbor))
    # Sum the capacities of all cut edges
    cut_value = sum(network[i, j] for i, j in cut_edges)
    return cut_edges, cut_value
```

```python
def maxflow_segmentation(file, source_pixel, sink_pixel, sigma):
    # Load image data
    image_data = load_image(file)
    num_rows, num_cols = image_data.shape
    num_pixels = num_rows * num_cols
    # Build the network
    network, source_node, sink_node = build_network(image_data, sigma, source_pixel, sink_pixel)
    # Initialize the optimization model
    mod = gp.Model()
    # Edges are the keys in the network dictionary
    edges = list(network.keys())
    # Create decision variables for every edge
    flow = mod.addVars(
        edges,
        lb=0,
        ub=network,
        name="flow"
    )
```

Section 3: Methodology of our New Image Processing Model

This tool processes .jpg and .csv files, normalizes, creates a network of cuts, and returns the segmented image. Below is a detailed summary of the process:

- File Processing
    - If the image is a .jpg file, the tool initially reads the image in grayscale, normalizes the pixel intensities to the range [0, 1], and returns the normalized image. If it is in .csv format, the tool returns it as a NumPy array.
    - Normalization is also used to ensure the pixel intensity differences are constrained within a range to improve the stability of the model.
- Similarity Calculation

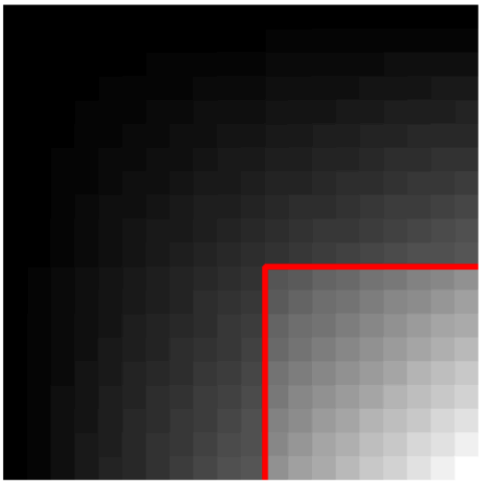
    - The tool then calculates similarity between pixel intensities, which decreases as the difference in intensity increases. This information is used to construct a sparse matrix representing a pixel-based flow network, where each pixel is connected to its neighbors (right, left, above, below) based on similarity. Source and sink nodes are added to this network to represent flow to the background and from the foreground.
    - Accessible nodes are identified using a depth-first search on the residual network, starting from the source node. The source node represents the foreground, while the sink node represents the background. The connections between the source and sink nodes will allow the model to decide which pixels belongs to the foreground and which belongs to the background during the segmentation process
- Max Flow Setup
    - The crux of the new tool is the **maxflow_segmentation** function, which uses the Gurobi optimizer to solve for the maximum flow algorithm in the context of the given image with the objective of maximizing flow into the sink node.
- Min Cut Identification
    - After solving the max flow problem, the model identifies accessible nodes starting from the source using a depth-first search, where it will start at the root nodes and work its way up to along each branch of nodes to find all accessible nodes.
    - The minimum cut is then extracted from the residual network that is corresponding to the edges in the original image network that was separated from the background
- Visualization of the image
    - The model then reconstructs the segmented image by grouping all the pixels into the foreground and background regions. The final result should show clear cuts around the object.

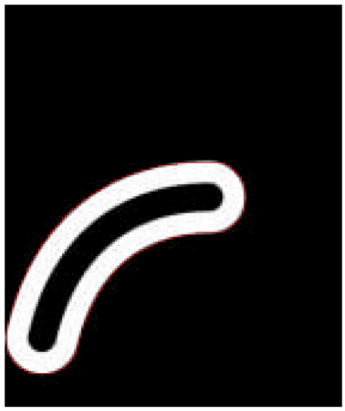
Section 4: Results

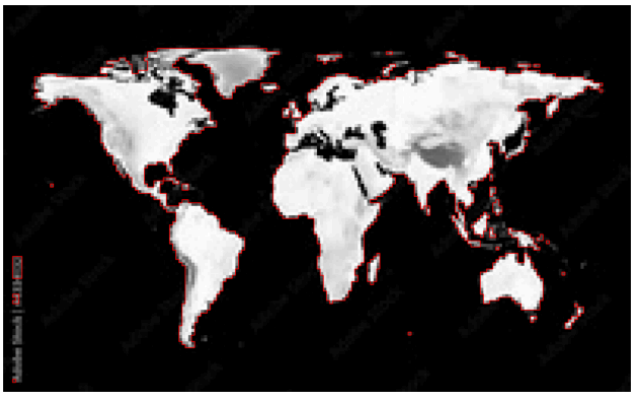
```
Optimize a model with 400 rows, 1522 columns and 3042 nonzeros
Model fingerprint: 0x1beb54c0
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [2e-13, 1e+02]
  RHS range        [0e+00, 0e+00]
Presolve removed 7 rows and 22 columns
Presolve time: 0.01s
Presolved: 393 rows, 1500 columns, 2998 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    3.2968136e+01   4.946917e+01   0.000000e+00      0s
Extra simplex iterations after uncrush: 22
      93    2.6461137e-03   0.000000e+00   0.000000e+00      0s

Solved in 93 iterations and 0.03 seconds (0.00 work units)
Optimal objective  2.646113669e-03
Min-Cut value (Max-Flow): 0.002646113669140901
```







The images are clearly segmented into foreground and background across differing pixel intensities, with the cuts represented by red lines. As illustrated above, the model is able to work efficiently for both simple and complex images.

Section 5: Conclusion and Recommendations

Overall, this image segmentation tool is effective when it comes to segmenting images, especially when clear boundaries exist between the foreground and background. By framing the problem as a network flow optimization problem, we were able to extract the optimal cuts. In addition, due to the use of sparse matrix and Gurobi, this model scaled exceptionally well for a variety of images, which makes it suitable for a diverse range of image segmentation tasks.

Comparing our model to Photoshop, we are confident that our max flow/min cut approach can offer a more straightforward, highly efficient solution for tasks such as object isolation or background removal. The automated nature of this algorithm also provides an advantage for users who may not be familiar with photo editing tools/skills, offering them a more accessible tool to achieve fantastic image editing results. In addition, the sigma value does not need to be changed for different images since we normalized the pixel intensity values, making generalizing this model to other images even more accessible and convenient.

While the current model performs well, we also believe that further improvements, such as more efficient memory/RAM management parallel processing, could make the tool even more efficient and process larger images faster. Solutions like exploring GPU-based acceleration could allow us to better segment high-resolution images. Furthermore, additional testing on more complex images, especially those with occlusions and overlapping objects, will help us fine-tune the model algorithm to handle edge cases and improve segmentation accuracy for a more diverse range of images.