Image Segmentation Using Linear Programming

Team members:

- Manasa Maganti
- Arantza Garcia Delfin
- Navya Singhal
- Cole Brown

Objective

Evaluate the feasibility of implementing image segmentation using the max flow/min cut theorem and linear programming, providing insights and recommendations for its integration into our product development.

Introduction

This project aims to explore image segmentation using **linear programming** and the **max flow/min cut theorem** to efficiently separate the foreground and background in images. Image segmentation is a key enabler for automating processes such as **object detection** and **visual data analysis** across various industries. By framing the segmentation problem as a linear program, we aim to enhance both **accuracy** and **scalability**, making it highly adaptable for practical applications, including **medical imaging** and **autonomous systems**.

The ultimate goal is to develop a solution that seamlessly integrates into our existing workflows, significantly improving the **speed** and **precision** of image processing tasks. The insights from this project will directly inform how we incorporate advanced segmentation techniques into our current and future product offerings, driving both innovation and operational efficiency.

Problem Description

The project involves converting a 2D image into a network of nodes and edges, where each pixel is treated as a node, and the similarity between neighboring pixels is represented as the edge weights connecting them. This conversion allows us to use mathematical optimization to identify the optimal “cut” that separates the foreground from the background. By framing the problem in this way, we can leverage the Max Flow / Min Cut Theorem to find the solution efficiently.

The key challenge lies in the following:

1. Selecting the Source and Sink Pixels:
- The source corresponds to a pixel (or pixels) in the background of the image.
- The sink corresponds to a pixel (or pixels) in the foreground.
- These nodes serve as the start and end points for calculating the flow in the network.
1. Constructing the Network:
- Each pixel in the image is treated as a node in the network.
- Edges between neighboring pixels are weighted by the similarity of their intensity values, calculated using the formula:

$$\text{Similarity}(I_i, I_j) = 100 \cdot \exp\left(-\frac{(I_i - I_j)^2}{2\sigma^2}\right)$$

Where:

- I_i and I_j represent the intensity values of the two pixels.
- σ is a parameter that controls how sharply the similarity decreases as the difference between the pixel intensities increases.

Once the network is constructed, we frame the image segmentation problem as a max flow problem, where the objective is to maximize the flow from the source node (representing the background pixel) to the sink node (representing the foreground pixel). The edges that become fully saturated with flow represent the cut that separates the background from the foreground.

Methodology

In this project, the image segmentation problem is modeled by treating each pixel as a node in a network, with edges representing the similarity between neighboring pixels. The segmentation is solved using the max flow/min cut theorem and linear programming, and the following key components are implemented:

1. **Sigma Value for Similarity Calculation**: The sigma (σ) value is critical for controlling sensitivity to differences in pixel intensities. In the code, σ is dynamically calculated based on the range of pixel intensities in the image:

$$\sigma = 0.01 \times (\text{max\_intensity} - \text{min\_intensity})$$

Alternatively, the user can manually set the σ value, allowing fine-tuning for sharper or smoother segmentation based on pixel intensity differences.

1. **Adjacency and Neighborhood Definition**: Each pixel is connected to its adjacent pixels (up, down, left, right), forming the edges of the network. The similarity between two neighboring pixels is calculated using a Gaussian function:

$$\text{Similarity}(I_i, I_j) = 100 \cdot \exp\left(-\frac{(I_i - I_j)^2}{2\sigma^2}\right)$$

Higher similarity values correspond to stronger connections between pixels, ensuring that similar pixels remain grouped together during segmentation.

1. **Flow Conservation Constraints**: The flow conservation constraint ensures that the flow into each pixel equals the flow out, except at the source and sink nodes. This balance is crucial for solving the max flow problem:

$$\sum_j f_{ji} = \sum_j f_{ij} \text{ for each pixel } i$$

This guarantees a balanced flow at each node, ensuring mathematical correctness during optimization.

1. **Cut Identification via Depth-First Search (DFS)**: After solving the linear program to compute the maximum flow, a Depth-First Search (DFS) is performed on the residual network to identify the minimum cut. This cut separates the reachable pixels (background) from the non-reachable pixels (foreground), ensuring the segmentation is based on flow saturation.

2. **Visualization**: Once the minimum cut is identified, the cut edges are visualized on the original image, highlighting the boundary between the foreground and background. This provides a clear and actionable visual representation of the segmentation.

Recommendation

The Image Segmentation Project, which implemented the max-flow min-cut algorithm using linear programming, has demonstrated exceptional results in accurately segmenting grayscale images by leveraging the Gurobi library and representing pixels as nodes in a graph. This approach effectively identified and highlighted distinct regions, showcasing adaptability in handling various image formats and sizes. However, the current limitation is that the model can efficiently process images up to 250x250 pixels, requiring significant computational power for larger resolutions. Additionally, the sigma value, which influences pixel similarity calculations, needs to be adjusted for different cases to ensure accurate segmentation results.

In a business context, this segmentation model can be particularly useful in the retail industry for analyzing customer traffic patterns within stores. By applying this technique to surveillance footage, businesses can identify high-traffic areas and optimize product placement or marketing displays, thereby improving sales strategies. For future enhancements, expanding the model to handle color images and incorporating machine learning techniques can further improve segmentation accuracy. Implementing performance optimizations and developing a user-friendly interface would make this tool more versatile and accessible, enabling it to evolve into a comprehensive solution for diverse industries requiring image analysis and processing.

Scope for further improvement

1. **Processing Time for Large Images**: While the algorithm works efficiently for smaller image sets, processing large-scale images or high-resolution data can cause

performance bottlenecks. Optimizing the code for parallel processing and incorporating GPU acceleration can significantly reduce the time needed for high-resolution images.

2. **Manual Tuning of Sigma (σ)**: Currently, adjusting the sigma value requires some manual intervention or estimation to fine-tune the segmentation for different images. Future versions could include an automated sigma adjustment based on image characteristics, making the tool more user-friendly.

3. **Handling of Complex Edges**: The current model may struggle with highly complex or overlapping objects, where the similarity function does not perfectly capture fine-grained differences in edge detail.

4. **Adding More Image Formats**: The current model is equiped to handle jpg or jpeg images only. It reduces the flexibility of the system, particularly in industries that rely on specialized image formats such as PNG, TIFF, BMP, and DICOM (commonly used in medical imaging).

We need to expand the algorithm to support a wider range of image formats will significantly increase its usability across various sectors:

- ***PNG***: Often used for high-quality images in web and design applications, supporting transparency.
- ***RAW***: Commonly used in autonomous driving, as it preserves unprocessed data directly from vehicle sensors, providing maximum detail for real-time decision-making.
- ***DICOM***: The industry standard for handling, storing, and transmitting medical images (e.g., CT scans, MRIs). Adding support for DICOM would greatly enhance the system's relevance in medical imaging, allowing for more seamless integration with hospital systems.

By addressing these constraints, we can further improve the algorithm's scalability, usability, and accuracy, making it even more competitive in the image processing market.

```python
import numpy as np
import csv
import math
import gurobipy as gp
from gurobipy import GRB
import matplotlib.pyplot as plt
import os
from PIL import Image
from collections import defaultdict as dd

import sys

# Need this to avoid RecursionError for dfs
sys.setrecursionlimit(100000)
```

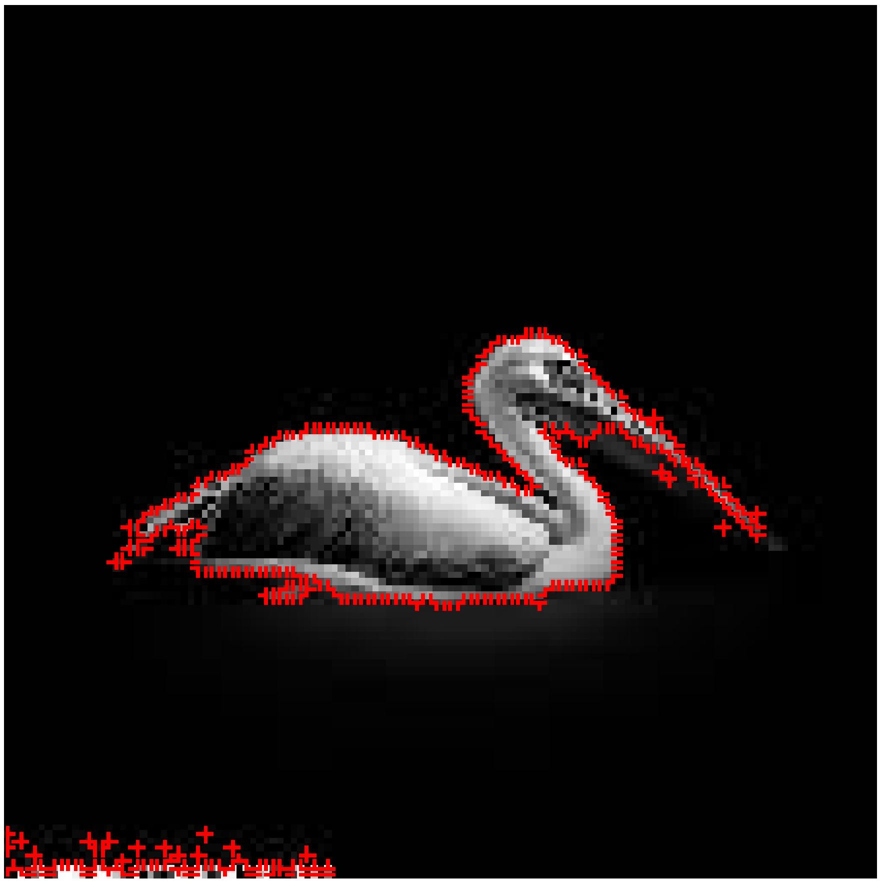
```
class ImageSegmentation:
    """
    This class is used to segment an image into two regions using the
max-flow min-cut algorithm.
    The image is represented as a graph where each pixel is a node and
the edges are weighted by the similarity between the pixels.
    """

    model = None

    source = None
    sink = None
    image_array = None
    sigma = None
    fg_pixel = ()
    bg_pixel = ()
    image_arr_flat = None

    def __init__(self, filepath, sigma=None):
        """
        This function initializes the ImageSegmentation class with the
image file path and the sigma value.
        It also loads the image and sets the background and foreground
pixels.
        """
        self.load_image(filepath)
        if sigma is None:
            self.sigma = 0.01 * (np.max(self.image_array) -
np.min(self.image_array))
        else:
            self.sigma = sigma

    def load_image(self, file_path):
        """
        This function loads the image from the file path and converts
it to a greyscale 2D array.
        """

        if file_path.endswith(".jpg") or file_path.endswith(".jpeg"):
            image_array = np.array(Image.open(file_path).convert('L'))
            self.image_array = image_array
        else:
            # Load the image and convert it to a greyscale 2D array
            self.image_array = np.loadtxt(file_path, delimiter=',')

        self.bg_pixel = np.argmin(self.image_array.flatten())
        self.fg_pixel = np.argmax(self.image_array.flatten())
        self.image_arr_flat = self.image_array.flatten()

    def calc_similarity(self, pixel_i, pixel_j):
        """
        This function calculates the similarity between two pixels
using the exponential function.
        """
        # return np.ceil(100 * np.exp(-
((float(self.image_arr_flat[pixel_i]) -
float(self.image_arr_flat[pixel_j])) ** 2) / (2 * self.sigma ** 2)))
        return 100 * np.exp(-((float(self.image_arr_flat[pixel_i]) -
float(self.image_arr_flat[pixel_j])) ** 2) / (2 * self.sigma ** 2))

    def create_network(self):
        """
        This function creates the network for the image graph.
        Here we use a defaultdict to store the network as a dictionary
of dictionaries so that we can handle files which can process large
number of pixels.
        """
        rows, cols = self.image_array.shape
        size = rows * cols

        network = dd(dict)
        for x in range(size):
            r_num = x // cols
            c_num = x % cols
            near_pixels = []
            if r_num > 0:
                near_pixels.append(x - cols)
            if r_num < rows - 1:
                near_pixels.append(x + cols)
            if c_num > 0:
                near_pixels.append(x - 1)
            if c_num < cols - 1:
                near_pixels.append(x + 1)

            for pixel in near_pixels:
               network[x][pixel] = self.calc_similarity(x, pixel)

        # adding the source and sink (source --> background &&
foreground --> sink)
        network = self._add_terminal_nodes(network, size)
        return network

    def _add_terminal_nodes(self, network, size):
        """
        This function adds the source and sink nodes to the network.
        """
        network[size][self.bg_pixel] = np.inf # source
        network[self.fg_pixel][size+1] =  np.inf # sink
        self.source = size
        self.sink = size+1
        return network

    def solve_lp(self, network):
        """
        This function solves the linear program using the Gurobi
library.
        """
        if None in [self.source, self.sink]:
            raise ValueError("Source and Sink not set")

        self.model = gp.Model()

        # Create variables for the flow on each edge
        flow = {(i, j): self.model.addVar(lb=0, ub=network[i][j],
name=f'{i}-{j}')
                for i in network for j in network[i]}

        # Update the model to include these variables
        self.model.setObjective(gp.quicksum(flow[self.source, j] for j
in network[self.source]), GRB.MAXIMIZE)

        # Add flow conservation constraints
        for i in network:
            if i != self.source and i != self.sink:
                inflow = gp.quicksum(flow[j, i] for j in network if i
in network[j])
                outflow = gp.quicksum(flow[i, j] for j in network[i])
                self.model.addConstr(inflow == outflow)

        self.model.optimize()

        return flow

    def locate_cuts(self, network, flow):
        """
        This function locates the cuts in the network using a depth-
first search recursive function.
        Note: This is why we set the recursion limit at the beginning
of the script.
        """
        visited = set()

        def dfs(node):
            """
            This function performs a depth-first search to identify
the cuts.
            """
            visited.add(node)
            for neighbor in network[node]:
                if neighbor not in visited and
self.model.getVarByName(f'{node}-{neighbor}').X < network[node]
[neighbor] - 1e-6:
                    dfs(neighbor)

        dfs(self.source)

        # Identify cuts where one node is in visited and the other is
not
        cuts = [(i, j) for i in visited for j in network[i] if j not
in visited]

        return cuts


    def draw(self, cuts = []):
        """
        This function draws the image with the cuts highlighted in
red.
        """
        plt.figure(figsize=(5, 5))
        plt.imshow(self.image_array, cmap='gray')
        rows, cols = self.image_array.shape
        if cuts:
            for i, j in cuts:
                ri, ci = divmod(i, cols)
                rj, cj = divmod(j, cols)
                plt.plot([ci, cj], [ri, rj], color='red',
linewidth=1.5)
        plt.title("Image with Cuts Highlighted in Red")
        plt.axis('off')
        plt.tight_layout()
        plt.show()


    def run_segmentation(self):
        """
        This function runs the image segmentation algorithm.
        """
        print("Creating Network...")
        network = self.create_network()
        print("Solving LP...")
        flow = self.solve_lp(network)
        print("Locating Cuts...")
        cuts = self.locate_cuts(network, flow)
        print("Drawing Image...")
        self.draw(cuts)

#Create an object and run the image segmentation algorithm
```

```
segmentation = ImageSegmentation('P.jpg')
segmentation.run_segmentation()

"""
NOTE:
oval-1.csv - Remove sigma argument from above, let it take the default
from the class
Pic3.jpg (Pelican Image) - Remove sigma argument from above, let it
take the default from the class
box.csv - Remove sigma argument from above, let it take the default
from the class
circle_and_square.jpg - Sigma 5
bottle - sigma 3.5
flower.jpg - sigma 5
"""

Creating Network...
Solving LP...
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (mac64[arm] - Darwin
23.3.0 23D56)

CPU model: Apple M3 Pro
Thread count: 11 physical cores, 11 logical processors, using up to 11
threads

Optimize a model with 16384 rows, 65026 columns and 130050 nonzeros
Model fingerprint: 0x9c3718d3
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [2e-319, 1e+02]
  RHS range        [0e+00, 0e+00]
Presolve removed 1497 rows and 6427 columns
Presolve time: 0.03s
Presolved: 14887 rows, 58599 columns, 117049 nonzeros

Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only...

Ordering time: 0.01s

Barrier statistics:
 AA' NZ     : 2.920e+04
 Factor NZ  : 3.418e+05 (roughly 30 MB of memory)
 Factor Ops : 1.338e+07 (less than 1 second per iteration)
 Threads    : 9

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl
Time
```

```
   0   2.30515620e+01  5.84758692e+06  2.31e+01 3.08e+01  4.79e+03
0s
   1   3.47446989e+01  1.51949796e+06  3.40e+00 1.12e+01  8.67e+02
0s
   2   8.91545598e+00  6.97216849e+04  4.47e-01 2.33e-01  3.89e+01
0s
   3   2.45050805e-01  2.72748704e+02  1.31e-02 1.56e-13  1.63e-01
0s
   4   1.41892300e-02  7.46673859e+00  9.51e-04 5.44e-12  4.80e-03
0s
   5  -7.93668704e-04  1.17425289e+00  1.66e-05 3.32e-11  6.52e-04
0s
   6  -1.50909348e-04  1.77949755e-01  1.56e-06 4.77e-12  9.79e-05
0s
   7  -5.67557374e-05  3.01769333e-02  9.46e-08 1.59e-12  1.65e-05
0s
   8  -2.26734895e-05  4.39196292e-03  2.92e-09 1.56e-13  2.41e-06
0s
   9  -5.38879725e-06  1.92713423e-04  6.67e-08 2.98e-14  1.07e-07
0s
  10  -2.98819357e-06  1.01024589e-05  7.82e-08 2.84e-14  5.96e-09
0s
  11  -1.94619669e-06  2.83986765e-06  1.68e-07 2.84e-14  1.40e-09
0s
  12  -6.26196406e-08  2.47372884e-06  1.24e-07 1.95e-14  1.12e-10
0s
  13  -6.15777150e-09  2.34251789e-06  2.79e-07 1.53e-14  7.57e-12
0s
  14  -1.53863308e-10  2.33560843e-06  1.32e-07 2.84e-14  3.41e-13
0s
  15  -1.86771505e-11  2.33518859e-06  1.56e-07 1.98e-14  3.56e-14
0s

Barrier solved model in 15 iterations and 0.23 seconds (0.33 work
units)
Optimal objective -1.86771505e-11

Crossover log...

   14862 DPushes remaining with DInf 0.0000000e+00                 0s
       0 DPushes remaining with DInf 0.0000000e+00                 1s

       1 PPushes remaining with PInf 1.4229042e-04                 1s
       0 PPushes remaining with PInf 1.4229042e-04                 1s

  Push phase complete: Pinf 1.4229042e-04, Dinf 0.0000000e+00      1s

Iteration    Objective       Primal Inf.    Dual Inf.      Time
   14862    2.3322722e-06   1.422904e-04   0.000000e+00      1s
```

```
Solved with barrier
Extra simplex iterations after uncrush: 818
   15809    1.1585978e-06   0.000000e+00   0.000000e+00      1s

Solved in 15809 iterations and 0.57 seconds (0.88 work units)
Optimal objective  1.158597798e-06
Locating Cuts...
Drawing Image...
```

Image with Cuts Highlighted in Red

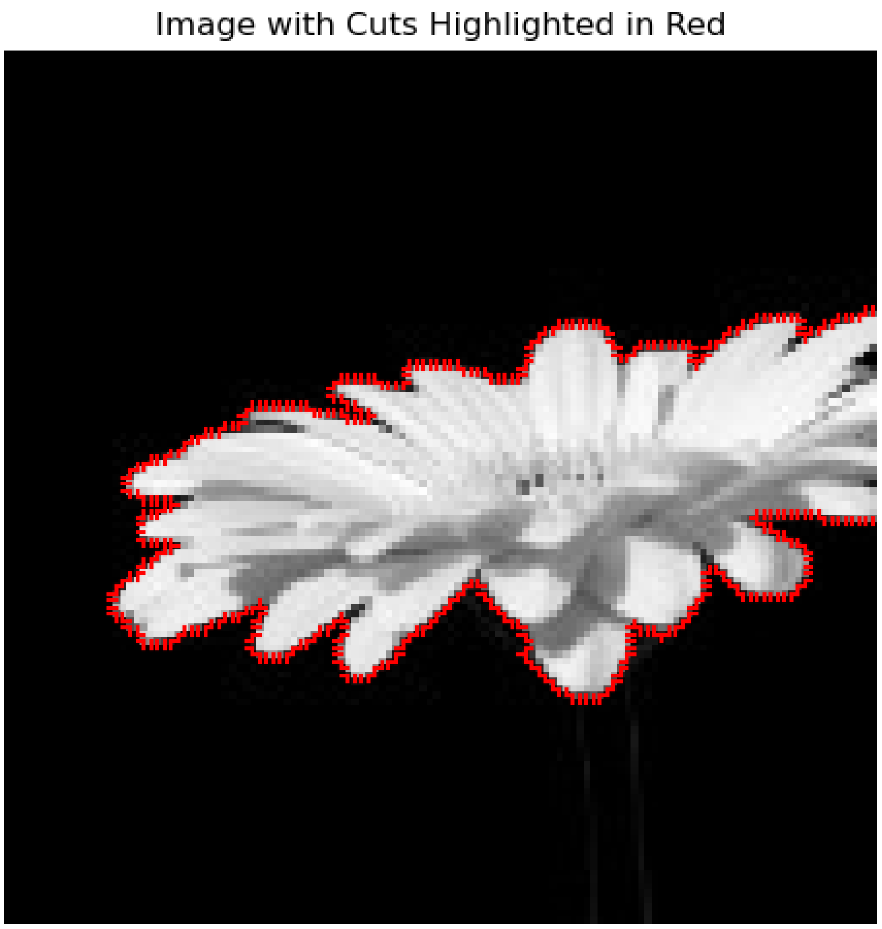


```
'\nNOTE:\noval-1.csv - Remove sigma argument from above, let it take
the default from the class\nPic3.jpg (Pelican Image) - Remove sigma
argument from above, let it take the default from the class\nbox.csv -
Remove sigma argument from above, let it take the default from the
class\ncircle_and_square.jpg - Sigma 5\nbottle - sigma 3.5\nflower.jpg
- sigma 5\n'

segmentation = ImageSegmentation('sf.jpg')

segmentation.run_segmentation()
```

```
Creating Network...
Solving LP...
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (mac64[arm] - Darwin
23.3.0 23D56)

CPU model: Apple M3 Pro
Thread count: 11 physical cores, 11 logical processors, using up to 11
threads

Optimize a model with 16384 rows, 65026 columns and 130050 nonzeros
Model fingerprint: 0x72db0c2f
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [2e-319, 1e+02]
  RHS range        [0e+00, 0e+00]
Presolve removed 1574 rows and 7395 columns
Presolve time: 0.03s
Presolved: 14810 rows, 57631 columns, 115129 nonzeros

Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only...

Ordering time: 0.01s

Barrier statistics:
 AA' NZ     : 2.871e+04
 Factor NZ  : 2.932e+05 (roughly 30 MB of memory)
 Factor Ops : 8.187e+06 (less than 1 second per iteration)
 Threads    : 9

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl
Time
   0   2.12433845e+01  3.52496655e+06  2.13e+01 1.47e+01  2.94e+03
0s
   1   2.42066330e+01  9.17527451e+05  3.39e+00 7.27e+00  5.37e+02
0s
   2   3.58192831e+00  3.14381762e+04  4.42e-01 9.95e-14  1.83e+01
0s
   3   1.10361984e-01  1.57496117e+02  1.49e-02 3.23e-12  1.04e-01
0s
   4  -3.40742093e-03  1.62232469e+01  7.57e-04 3.27e-12  9.62e-03
0s
   5  -2.15305814e-03  5.67147723e+00  5.07e-05 1.01e-12  3.18e-03
0s
   6  -8.40585691e-04  7.21956663e-01  3.88e-06 7.82e-14  4.03e-04
0s
   7  -3.32922280e-04  5.88000070e-02  9.63e-08 6.61e-13  3.29e-05
0s
```

```
   8  -3.32311129e-04  5.76146323e-02  2.95e-07 6.54e-13  3.22e-05
0s
   9  -2.81827093e-04  4.29183124e-03  3.49e-07 8.34e-14  2.56e-06
0s
  10  -2.62139190e-04  7.34993165e-04  4.31e-07 2.84e-14  5.66e-07
0s
  11  -2.36054273e-04  5.67564865e-05  3.78e-07 8.34e-14  1.71e-07
0s
  12  -2.41392513e-06  8.13251829e-06  4.36e-07 4.79e-14  4.35e-09
0s
  13  -3.16890025e-07  3.48699225e-06  4.84e-07 2.84e-14  5.16e-10
0s
  14  -1.02989853e-08  3.04883934e-06  3.98e-06 1.68e-14  8.93e-11
0s
  15  -5.03391441e-10  2.91858935e-06  2.39e-07 1.95e-14  1.08e-11
0s
  16  -3.95999051e-10  2.91600100e-06  6.23e-07 2.11e-14  9.32e-12
0s
  17  -3.78897254e-11  2.90231666e-06  2.18e-07 2.84e-14  1.49e-12
0s
  18  -2.71409758e-11  2.90116797e-06  1.93e-07 2.84e-14  8.46e-13
0s
  19  -9.48822750e-12  2.90082225e-06  2.11e-07 1.83e-14  6.41e-13
0s
  20  -9.38516844e-12  2.90078597e-06  2.08e-07 2.59e-14  6.21e-13
0s
  21  -7.55171797e-12  2.90059292e-06  1.71e-07 2.16e-14  5.12e-13
0s
  22  -2.50441208e-12  2.89974650e-06  1.10e-07 2.95e-14  3.78e-14
0s

Barrier solved model in 22 iterations and 0.33 seconds (0.46 work
units)
Optimal objective -2.50441208e-12

Crossover log...

   14802 DPushes remaining with DInf 0.0000000e+00                 1s
       0 DPushes remaining with DInf 0.0000000e+00                 1s

       0 PPushes remaining with PInf 2.9646429e-04                 1s

  Push phase complete: Pinf 2.9646429e-04, Dinf 7.1054545e-15      1s

Iteration    Objective       Primal Inf.    Dual Inf.      Time
   14731    1.4708415e-06   2.964643e-04   0.000000e+00      1s

Solved with barrier
Extra simplex iterations after uncrush: 656
   15419    4.8069244e-09   0.000000e+00   0.000000e+00      1s
```

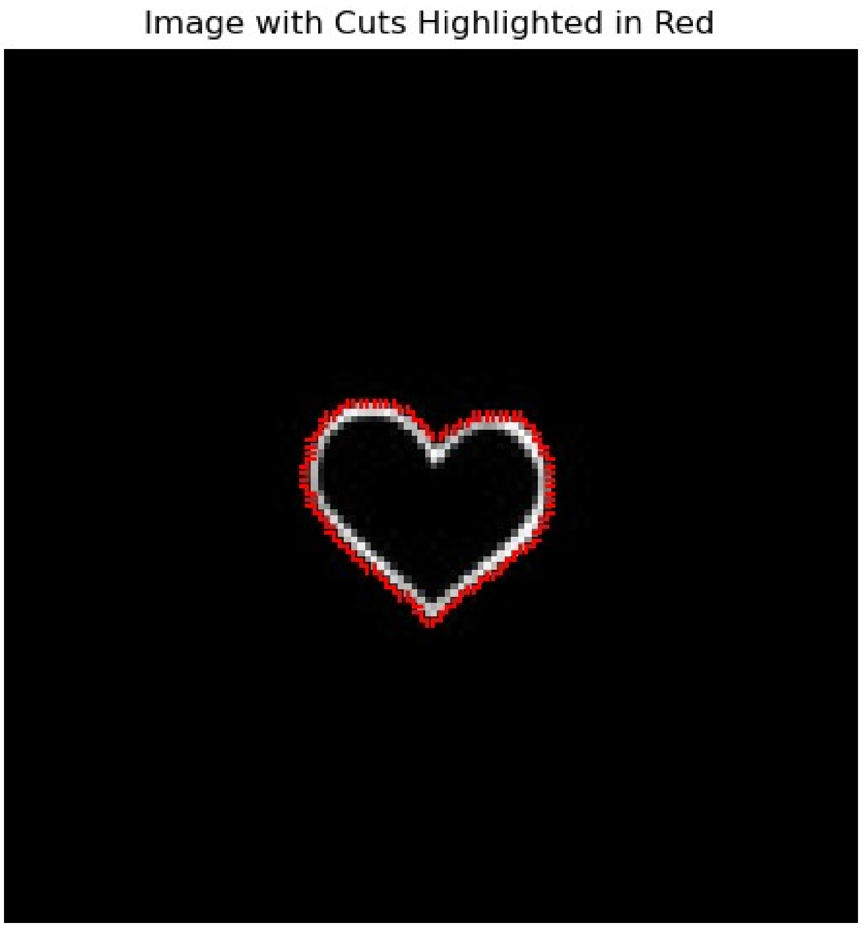
```
Solved in 15419 iterations and 0.68 seconds (1.00 work units)
Optimal objective  4.806924390e-09
Locating Cuts...
Drawing Image...
```



```
segmentation = ImageSegmentation('h.jpg')

segmentation.run_segmentation()

Creating Network...
Solving LP...
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (mac64[arm] - Darwin
23.3.0 23D56)

CPU model: Apple M3 Pro
Thread count: 11 physical cores, 11 logical processors, using up to 11
threads

Optimize a model with 16768 rows, 66556 columns and 133110 nonzeros
Model fingerprint: 0x659c1c19
```

```
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [6e-319, 1e+02]
  RHS range        [0e+00, 0e+00]
Presolve removed 383 rows and 1597 columns
Presolve time: 0.03s
Presolved: 16385 rows, 64959 columns, 129891 nonzeros

Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only...

Ordering time: 0.01s

Barrier statistics:
 AA' NZ     : 3.246e+04
 Factor NZ  : 4.098e+05 (roughly 36 MB of memory)
 Factor Ops : 2.014e+07 (less than 1 second per iteration)
 Threads    : 9

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl
Time
   0   2.45536973e+01  4.56273174e+06  2.46e+01 1.45e+01  3.37e+03
0s
   1   2.64139484e+01  1.13260225e+06  3.03e+00 6.20e+00  5.77e+02
0s
   2   3.61053151e-01  2.91629488e+04  2.02e-01 1.85e-13  1.44e+01
0s
   3  -1.49794129e-03  6.03387153e+01  1.00e-03 7.11e-14  2.98e-02
0s
   4  -1.96038436e-03  1.54291777e+00  1.21e-05 5.68e-14  7.61e-04
0s
   5  -4.55887969e-04  2.33513651e-01  8.36e-07 4.83e-13  1.15e-04
0s
   6  -8.23333166e-05  3.21358797e-02  6.21e-09 2.42e-13  1.59e-05
0s
   7  -2.00573533e-05  3.53736502e-03  1.51e-08 4.26e-14  1.75e-06
0s
   8  -2.72992177e-06  1.53017235e-04  6.26e-08 1.50e-14  7.65e-08
0s
   9  -7.55239533e-08  5.30108055e-06  3.23e-08 2.11e-14  2.44e-09
0s
  10  -6.98077100e-10  6.50532370e-07  2.42e-08 1.42e-14  1.12e-10
0s
  11  -4.67063836e-12  4.55731226e-07  4.74e-09 1.42e-14  1.56e-11
0s
  12  -2.63740430e-13  4.24433653e-07  4.99e-09 2.84e-14  1.43e-13
0s
  13  -5.65044210e-16  4.24145186e-07  5.24e-09 1.42e-14  5.39e-16
```

```
0s

Barrier solved model in 13 iterations and 0.23 seconds (0.32 work
units)
Optimal objective -5.65044210e-16


Solved with dual simplex
Extra simplex iterations after uncrush: 167
Iteration    Objective       Primal Inf.    Dual Inf.      Time
    2299   -0.0000000e+00   0.000000e+00   0.000000e+00      1s

Solved in 2299 iterations and 0.54 seconds (1.27 work units)
Optimal objective -0.000000000e+00
Locating Cuts...
Drawing Image...
```

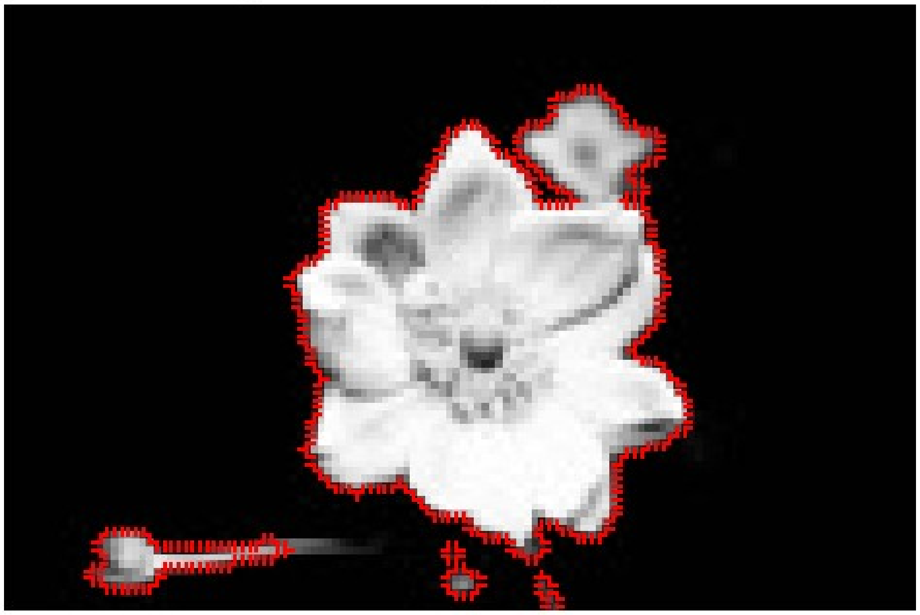


```python
segmentation = ImageSegmentation('f1.jpg')

segmentation.run_segmentation()
```

```
Creating Network...
Solving LP...
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (mac64[arm] - Darwin
23.3.0 23D56)

CPU model: Apple M3 Pro
Thread count: 11 physical cores, 11 logical processors, using up to 11
threads

Optimize a model with 10880 rows, 43096 columns and 86190 nonzeros
Model fingerprint: 0x7fb6ad37
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [2e-319, 1e+02]
  RHS range        [0e+00, 0e+00]
Presolve removed 1311 rows and 5853 columns
Presolve time: 0.02s
Presolved: 9569 rows, 37243 columns, 74363 nonzeros

Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only...

Ordering time: 0.00s

Barrier statistics:
 AA' NZ     : 1.853e+04
 Factor NZ  : 1.852e+05 (roughly 20 MB of memory)
 Factor Ops : 4.908e+06 (less than 1 second per iteration)
 Threads    : 9

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl
Time
   0   2.15715500e+01  3.71803830e+06  2.16e+01 3.29e+01  4.79e+03
0s
   1   3.47980484e+01  9.99934946e+05  3.38e+00 1.30e+01  9.02e+02
0s
   2   3.70324438e+00  2.60075038e+04  2.59e-01 8.53e-14  2.30e+01
0s
   3   8.44668757e-02  2.13147654e+02  6.54e-03 2.13e-13  1.93e-01
0s
   4  -1.18034388e-03  7.08547190e+00  1.77e-04 2.13e-13  6.27e-03
0s
   5  -1.64708586e-03  1.71619717e+00  2.28e-05 6.25e-13  1.49e-03
0s
   6  -4.71230888e-04  2.10077274e-01  1.05e-06 2.23e-12  1.82e-04
0s
   7  -1.30862420e-04  3.11310517e-02  3.53e-08 2.98e-13  2.69e-05
0s
```

```
   8  -1.21636092e-04  5.47240675e-03  2.30e-07 1.42e-13  4.82e-06
0s
   9  -1.12199183e-04  3.21856955e-04  2.85e-07 1.71e-13  3.83e-07
0s
  10  -1.09674082e-04  1.82271556e-04  3.02e-07 1.14e-13  2.60e-07
0s
  11  -7.03046535e-05  2.65045945e-05  9.95e-07 1.14e-13  8.69e-08
0s
  12  -5.57911203e-05  6.23105098e-06  2.34e-06 2.56e-13  5.49e-08
0s
  13  -3.79479601e-06  4.37837180e-06  1.44e-06 7.11e-14  3.87e-09
0s
  14  -2.86750649e-07  4.11550448e-06  1.28e-06 3.55e-14  3.38e-10
0s
  15  -2.67218691e-07  4.06572568e-06  1.13e-06 2.84e-14  2.75e-10
0s
  16  -3.47002262e-08  4.04355919e-06  2.50e-06 1.83e-14  3.86e-11
0s
  17  -3.25675760e-09  4.04020399e-06  1.47e-06 1.95e-14  6.37e-12
0s
  18  -1.66922729e-09  4.03770408e-06  1.74e-06 2.84e-14  2.52e-12
0s
  19  -1.55924238e-09  4.03764773e-06  1.68e-06 2.84e-14  2.37e-12
0s
  20  -5.83558453e-10  4.03761635e-06  1.99e-06 2.84e-14  1.43e-12
0s

Barrier performed 20 iterations in 0.20 seconds (0.29 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with primal simplex
Warning: Markowitz tolerance tightened to 0.5
Extra simplex iterations after uncrush: 613
Iteration    Objective       Primal Inf.    Dual Inf.      Time
    8806    1.6070708e-09   0.000000e+00   0.000000e+00      0s

Solved in 8806 iterations and 0.23 seconds (0.32 work units)
Optimal objective  1.607070815e-09
Locating Cuts...
Drawing Image...
```

Image with Cuts Highlighted in Red

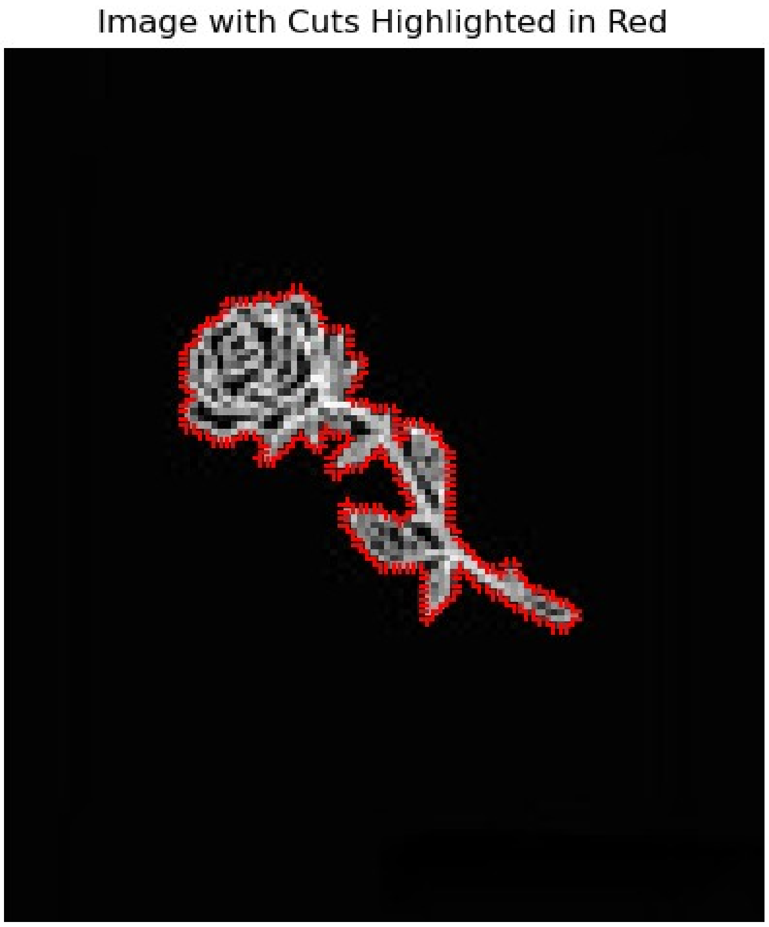


```
segmentation = ImageSegmentation('f2.jpg')

segmentation.run_segmentation()
```

```
Creating Network...
Solving LP...
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (mac64[arm] - Darwin
23.3.0 23D56)

CPU model: Apple M3 Pro
Thread count: 11 physical cores, 11 logical processors, using up to 11
threads

Optimize a model with 14934 rows, 59248 columns and 118494 nonzeros
Model fingerprint: 0xda4f9047
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [5e-322, 1e+02]
  RHS range        [0e+00, 0e+00]
Presolve removed 976 rows and 4027 columns
Presolve time: 0.03s
Presolved: 13958 rows, 55221 columns, 110413 nonzeros

Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only...

Ordering time: 0.02s
```

```
Barrier statistics:
 AA' NZ     : 2.759e+04
 Factor NZ  : 3.311e+05 (roughly 30 MB of memory)
 Factor Ops : 1.390e+07 (less than 1 second per iteration)
 Threads    : 9

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl
Time
   0   2.40930022e+01  5.33539395e+06  2.41e+01 2.85e+01  4.64e+03
0s
   1   2.69506816e+01  7.60147827e+05  3.44e+00 1.79e+00  4.58e+02
0s
   2   3.85286778e-01  2.06127575e+04  1.06e-01 3.27e-13  1.20e+01
0s
   3   1.23177144e-03  2.58422740e+01  1.18e-03 4.97e-14  1.51e-02
0s
   4  -1.54213383e-03  1.13410236e+00  3.41e-06 6.73e-13  6.58e-04
0s
   5  -2.43414857e-04  1.81502247e-01  5.73e-08 2.27e-13  1.05e-04
0s
   6  -4.55416598e-05  2.99409997e-02  9.89e-09 5.68e-14  1.74e-05
0s
   7  -6.92659374e-06  1.32428708e-03  1.29e-08 4.26e-14  7.71e-07
0s
   8  -1.10764221e-06  2.93236998e-05  1.99e-08 4.44e-14  1.74e-08
0s
   9  -4.37979513e-09  2.43989806e-06  2.72e-08 1.74e-14  1.13e-09
0s
  10  -1.10976817e-10  7.29608303e-07  1.30e-08 2.23e-14  1.41e-10
0s
  11  -3.43720893e-12  5.03232208e-07  8.67e-09 1.50e-14  9.47e-12
0s
  12  -2.63209868e-13  4.87519451e-07  7.39e-09 1.42e-14  3.66e-13
0s
  13  -2.71012667e-14  4.86889590e-07  6.23e-09 1.42e-14  3.75e-16
0s

Barrier solved model in 13 iterations and 0.21 seconds (0.26 work
units)
Optimal objective -2.71012667e-14

Crossover log...

   13958 DPushes remaining with DInf 0.0000000e+00                 0s
       0 DPushes remaining with DInf 0.0000000e+00                 0s

       0 PPushes remaining with PInf 0.0000000e+00                 0s

  Push phase complete: Pinf 0.0000000e+00, Dinf 0.0000000e+00      0s
```

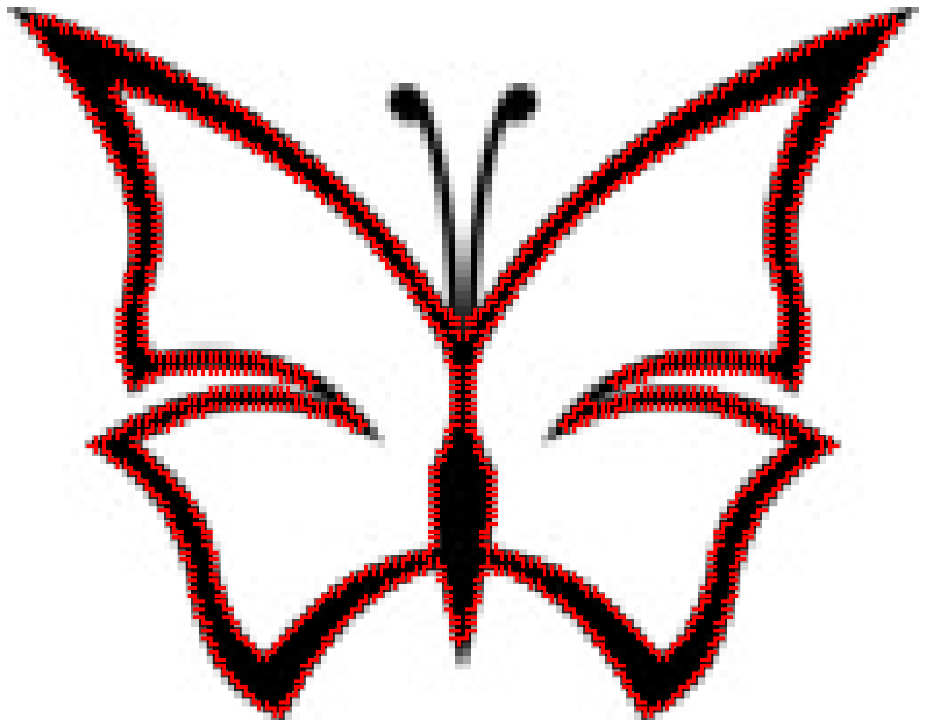
```
Solved with barrier
Warning: Markowitz tolerance tightened to 0.5
Extra simplex iterations after uncrush: 595
Iteration    Objective       Primal Inf.    Dual Inf.      Time
   14553    4.4262450e-07   0.000000e+00   0.000000e+00      1s

Solved in 14553 iterations and 0.54 seconds (0.78 work units)
Optimal objective  4.426245011e-07
Locating Cuts...
Drawing Image...
```



```
segmentation = ImageSegmentation('m.jpg')

segmentation.run_segmentation()

Creating Network...
Solving LP...
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (mac64[arm] - Darwin
23.3.0 23D56)
```

```
CPU model: Apple M3 Pro
Thread count: 11 physical cores, 11 logical processors, using up to 11
threads

Optimize a model with 12800 rows, 50746 columns and 101490 nonzeros
Model fingerprint: 0xef279918
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [2e-319, 1e+02]
  RHS range        [0e+00, 0e+00]
Presolve removed 2415 rows and 10653 columns
Presolve time: 0.03s
Presolved: 10385 rows, 40093 columns, 80051 nonzeros

Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only...

Ordering time: 0.01s

Barrier statistics:
 AA' NZ     : 1.994e+04
 Factor NZ  : 1.689e+05 (roughly 20 MB of memory)
 Factor Ops : 3.090e+06 (less than 1 second per iteration)
 Threads    : 9

Barrier performed 0 iterations in 0.09 seconds (0.08 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex
Extra simplex iterations after uncrush: 1018
Iteration    Objective       Primal Inf.    Dual Inf.      Time
    1810    2.3850973e-08   0.000000e+00   0.000000e+00      0s

Solved in 1810 iterations and 0.13 seconds (0.18 work units)
Optimal objective  2.385097297e-08
Locating Cuts...
Drawing Image...
```

Image with Cuts Highlighted in Red



```
segmentation = ImageSegmentation('cat.jpg')

segmentation.run_segmentation()
```

```
Creating Network...
Solving LP...
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (mac64[arm] - Darwin
23.3.0 23D56)

CPU model: Apple M3 Pro
Thread count: 11 physical cores, 11 logical processors, using up to 11
threads

Optimize a model with 17280 rows, 68596 columns and 137190 nonzeros
Model fingerprint: 0xb331e7dd
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [2e-319, 1e+02]
  RHS range        [0e+00, 0e+00]
Presolve removed 1877 rows and 8419 columns
Presolve time: 0.04s
Presolved: 15403 rows, 60177 columns, 120227 nonzeros

Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only...
```

```

Ordering time: 0.01s

Barrier performed 0 iterations in 0.09 seconds (0.09 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex
Extra simplex iterations after uncrush: 495
Iteration    Objective       Primal Inf.    Dual Inf.      Time
     714    2.5457297e-08   0.000000e+00   0.000000e+00      0s

Solved in 714 iterations and 0.14 seconds (0.15 work units)
Optimal objective  2.545729672e-08
Locating Cuts...
Drawing Image...
```

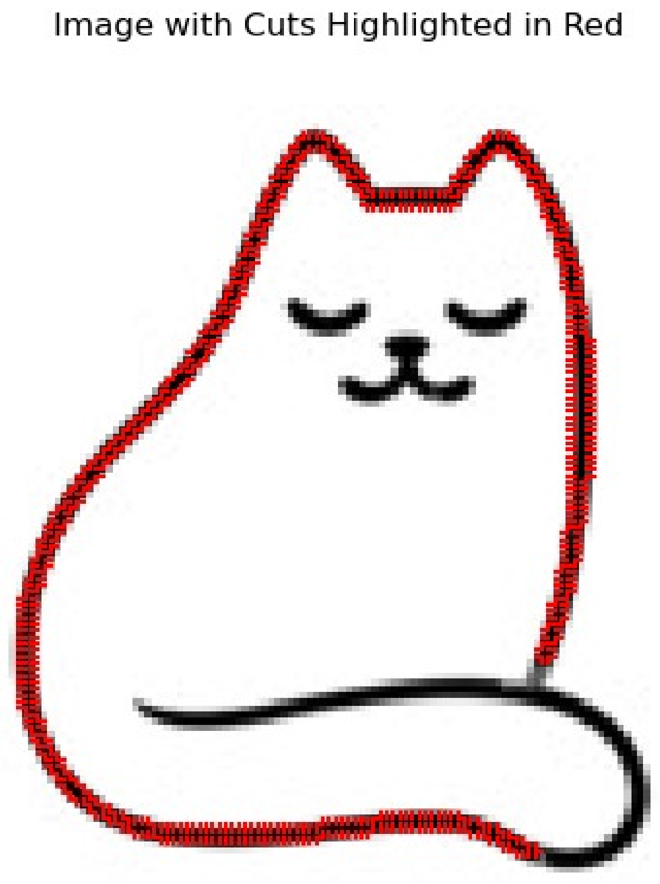


Image with Cuts Highlighted in Red