```
Import libraries such as:

from skimage.segmentation import active_contour
import networkx as nx
import opencv as cv2

(And any other libraries required for this task)


def GCBAC(Image_filename):
      ## Read the image
      ## Let Img be the image matrix

      Img_current_cont = active_contour(gaussian(Img, 3),init,
      alpha=0.015, beta=10, gamma=0.001)

      ## This while loop keeps repeating until flag = 0
      ## Hence when the new contour == previous contour, the loop will break.
      while(flag == 1):
            Img_dilate = cv2.dilate(Img_current_cont, (15,15), 5)
            Img_edge = cv2.Canny(Img_dilate,100,200)

       ## Using the two-edges detected for the contour, and using the dilated image matrix,
       ## create a graph where the edges represent the source and the sink node and
       ## the remaining pixels are inner nodes of the graph.

       ## create_Graph() fnx is used to create a graph using dilated image and its edges.
       ## It first removes the edge pixels from the dilated image pixels to avoid
       ## overlapping. As in the paper, the edge weight between pixels is computed as:
            cost_edge = exp(- (Img[i] - Img[j])^2/2*sig^2)
            where sig: scale parameter
       ## It returns the source node, sink node, and the Graph
            source, sink, G_cont_initial = create_Graph(Img_dilate,
                                                        Img_edge)

       ## Using the networkx.minimum_cut() or any other method for finding minimum
       ## cut, find the partitions
            cut_value, partition = nx.minimum_cut(G_cont_initial,
                                                  source, sink)

      ## Calculate the capacity for each partition obtained using the min cut.
            for each partition p(i):
                  flow_value, flow_dict = nx.maximum_flow(p(i),
                                                p_source, p_sink)
```

The new contour would be the partition with the minimum max flow.

```
if partition1 > partition2:
     G_cont_new = partition2
else:
     G_cont_new = partition1
```

Compare the G_cont_initial and G_cont_new.

```
if (G_cont_initial similar to G_cont_new):
     flag = 0
```

The while loop keeps repeating until fla = 0
Return the segmented image file

```
return Img_segmented
```