

Bryant Le

Ellen Zheng

5 May 2019

O5 - Independent Project

## Data Science in Natural Resource Management



### Overview:

This project idea came about through a merging of two passions - environmental science and data science. By combining data science analytics with environmental science's historical research methods, we believe technology can be used to better natural disaster relief efforts. The main setting we applied our research to was Hurricane Katrina in New Orleans, Louisiana.

## Table of Contents:

Table of Contents.....	2
Introduction.....	3
Project Purpose and Proposed Solution.....	4
Key Stakeholders.....	5
Code Overview.....	6-14
Program Results.....	15-16
Conclusion and Reflection.....	17
Bibliography.....	18



## Introduction:

Hurricane Katrina struck New Orleans, Louisiana on August 29, 2005. It was categorized as a “category 3” hurricane on the Saffir-Simpson Scale and devastated the the US Gulf Coast and its people. In total, an estimated 1,833 people died, with 1,577 of these people being from Louisiana (CNN, 2018). Damage that resulted from storm surge and high winds led to significant devastation. Most of New Orleans was flooded as well. The Congressional Research Service estimated that around 700,000 people or more were impacted by Hurricane Katrina. It is estimated that 77% of New Orleans’ population was affected and 645, 000 people were displaced, with at least half of this number from New Orleans (Gabe, 2005).

In terms of property damage, due to breaches in the levee system and mass flooding, it is estimated that Hurricane Katrina resulted in somewhere between \$81 and \$125 billion in damage (History, 2009).

The social significance of Hurricane Katrina is one that cannot be overlooked. The Congressional Research Service estimated that one in five of those affected by Hurricane Katrina were poor and 44% of victims were African American (Gabe, 2005). The damage and devastation left behind by Hurricane Katrina was magnified by a lack of action by the government. Relief efforts were either late or struggled to aid those who needed it most.

## Project Purpose and Proposed Solution:

After learning about the destruction caused by Hurricane Katrina, we couldn't help but feel disappointed that these disaster victims did not receive the aid that they needed in such critical moments in their lives. They could have received help, but due to inaction, were left on their own.

But we also wondered, why did the government not take action? What stopped them from helping? While of course, the exact reason is unknown, it is important to realize that implementing disaster relief efforts is difficult and we believe this was a challenge being faced by organizations at the time. Organizing at a scale large enough to resolve the aftermath left by Hurricane Katrina is a complex and daunting task.

This resulted in our project idea and a proposed solution: a program that would use computer vision algorithms and image processing to optimize aid deployment in affected areas of natural disasters. This program would then be simplified by implementing a Graphical User Interface, giving it a simplistic and accessible design that would allow it to be easy to use and understand.

## Key Stakeholders:

### Disaster Victims:

This is the group our program is most centered on, and the group we want to make sure benefits the most from its use. Our optimization is based on the most clustered regions: meaning selected areas are ones that will be able to help the greatest number of people.

### Government Organizations:

Government organizations are not directly impacted by the effects of natural disasters but are key decision makers that decide how relief efforts are executed. This program will allow government programs to optimize their limited resources by giving them valuable information that supplements their decision-making.

### Relief Organizations:

Like government organizations, these groups are not directly impacted by natural disasters but play a significant role in helping disaster victims. Our program would allow them to most efficiently distribute their resources among those in need.

# Code Overview:

## Introduction: Optimal locations to deploy aid.

The main issue at hand is deploying aid to victims of natural disasters. As described before, Hurricane Katrina demonstrated the lack of aid throughout most regions. The problem of determining where to deploy aid and the specific amount of resources to give to a certain region can be abstracted to two computer science algorithmic problems: the set cover problem or the K-clustering problem (algorithmic problems explained and solved by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani.)

## The Set Cover Problem

The set cover is used to describe the problem of covering as many sets as possible with a limited amount of covers, which are limited in size. For example, we can specify the set cover problem in the case of building schools in a new city. Given the locations of all the houses in the city, can we determine a way to build schools so that we do not build too many schools, which will be an expensive burden for the city, while not building too few schools that students have to walk a far distance.

We propose that the algorithm that would determine the optimal placement of sets in the most time-efficient manner is through a greedy algorithm:

1. *Consider each point that we are given.*
2. *For each point, check that if we make that point the center of our set, how many other points does it include.*
3. *Find the point where the most points are included, and remove those points from our list of points that we need to build a set for.*
4. *We continue our process of choosing points as centers for our sets, and we continue to remove the point and its included points with the most points until there are no points.*

In the example of the school, we will want to try placing the school in every possible location. We will count the number of students the school is considered in “walking distance”. Once we find a school that covers the most students, we will remove the location of the school, as well as all the homes of the students that are covered by that school, in order to consider building a school that is close for other students as well. We continue iterating through this process until we have built a school that is in walking distance for all students.

This algorithm goes back to our issue to determining where to distribute aid because we need to determine the optimal location where there is the most people. The people caught on footage by the drone will be the “students or homes” and the locations of our aid deployment schools represent the locations of our aid deployments. By solving the set cover problem, we solve the problem of optimizing aid deployments.

### The K-Cluster Problem

The K-Cluster problem describes of the exact same problem except that instead of calling the different combination of points sets, we call them clusters. And the ‘k’ in the name of the algorithm simply means the number of clusters we are allowed to have, analogous to amount of resources that we can deploy to one region.

We will not go into too much detail for this algorithm because it was one of the algorithms that we considered, but it was not the algorithm that we decided on.

The algorithm for K-Cluster is:

- 1. Start by picking any random point and assign that location / point to be the location of the first school.*
- 2. Now we pick a point that is furthest away from that first school, which should in theory cover the students that are too far away to attend the first school. Therefore, we will assign that point to be another school.*

3. *We will add another school that is the point furthest away from both schools, which should in theory also cover the students that are too far away to attend the first two schools.*
4. *We iteratively get the point furthest away from all other points until we have used up all  $K$  schools.*

Although, both algorithms solve the exact same problem in a similar amount of time, we chose the set cover algorithm because the K-cluster algorithm intuitively makes more sense, however is more difficult to implement programmatically.

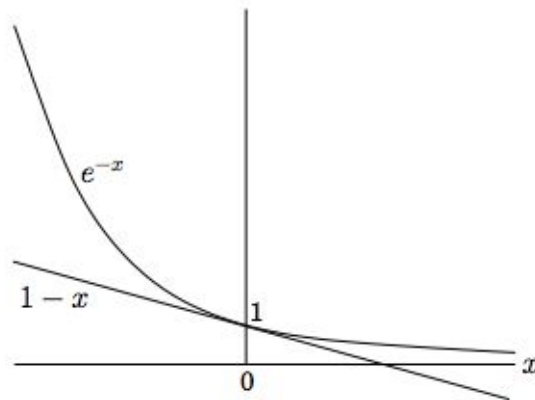
### Greedy Solution vs. Optimal Solution

There is one problem with the solution presented for both the set cover problem and the K-cluster problem: the solutions are not optimal. (There is a better solution.) However, something we have to consider is that lives are on the line when people are in need of aid, and the drone does not have enough time to calculate the most optimal solution in an efficient time. However, by reducing our solution to a greedy solution, a solution that greedily picks possibilities such as picking the school with the most points, we reduce the runtime of our program immensely so that we can effectively save more lives. In this way, while we may not be able to reach everyone right away, we will reach the most amount of people in the least amount of time.

We can compare the greedy algorithm and the optimal solution to estimating multiplied numbers. We could multiply out  $45 * 56$  by carrying numbers and adding exhaustively or we could estimate the value by multiplying  $50 * 60$ . Although the answer is not perfect, the answer is close and can be computed very fast, which is what we aim for in our greedy algorithm.



### Accuracy of the Greedy Solution



Thus

$$n_t \leq n_0 \left(1 - \frac{1}{k}\right)^t < n_0 (e^{-1/k})^t = ne^{-t/k}.$$

At  $t = k \ln n$ , therefore,  $n_t$  is strictly less than  $ne^{-\ln n} = 1$ , which means no elements remain to be covered.

This graph shows that the ratio of the optimal solution to the greedy solution is at most  $\ln(n)$ , which means that the solution will be very close to the optimal solution most of the time, while still being much more computationally efficient than the optimal solution.

### Deconstructing the Program

To determine the optimal location of aid deployment and to run our algorithm for the set cover problem, we need to first deconstruct the program by understanding how we can start from an image of a crowded area and get to the list of locations where we should deploy our resources.

The 'findDarkSpots' function deconstructs images by turning the images into a matrix of values (each spot in the matrix is a color, which as a whole make up the image.) The function iterates through the image for spots that are darker than the threshold. The reason why we decided to use dark pixels as features to classify human activity and clusters is because shadows casted by human-used objects and humans tend to indicate activity. Therefore, the more shadow a spot

has, the more activity there usually is. For example, in a beach setting, umbrella shadows and tent shadows, indicate human activity.

```
def findDarkSpots(img, limit=50):
    """
    Find spots darker than the limit in the grayscale img.

    Inputs: 'img' - the image file
            'limit' - how dark of: str threshold is to qualify a pixel value.
    Outputs: 'posList' - list of positions / pixels such that it is below the 'limit' value.
    """

    posList = []
    for row in range(len(img)):
        for col in range(len(img[0])):
            if (img[row][col] <= limit):
                pos = (row, col, img[row][col])
                posList.append(pos)
    print(posList)
    return posList
```

An important function that we should have when comparing and determining distance is the 'euclideanDistance' function, which finds the distance between two points. We want to use this function to determine whether a person is close enough to an aid or not. Analogously, we want to use this function to determine is a student is within walking distance of the school or not.

```
def euclideanDistance(pointA, pointB):
    """
    Finds euclidean distance from point A to point B.

    Inputs: 'pointA' - a position.
            'pointB' - a position.
    Output: Return - distance between the two points.
    """

    x = pointA[0] - pointB[0]
    y = pointA[1] - pointB[1]
    return (x**2 + y**2)**.5
```

The most important function must be the set cover algorithm itself, “greedySetCoverAlgorithm.” As described before, we will want to iterate through each point to determine the point that can cover the most points, and we want to iteratively do this until we find enough set covers to cover every point.

```
def greedySetCoverAlgorithm(points, aidDropCount = 5, trashDist = 200):
    """
    Greedy algorithm for set cover. Takes key points and we want to get the most objects we
    Inputs: 'points' - list of positions / pixels such that it is below the 'limit' value.
           'aidDropCount' - number of aid packages we can drop.
           'trashDist' - how much distance each aid package can cover.
    Outputs: 'sets' - the set of different circles and the points associated to each center.
    """

    sets = []
    while points and aidDropCount > 0:
        maxPoints = []
        for point in points:
            tempSet = [point]
            for otherPoint in points:
                if otherPoint != point and euclideanDistance(point, otherPoint) < trashDist:
                    tempSet.append(otherPoint)
            if len(tempSet) > len(maxPoints):
                maxPoints = tempSet[:]
        sets.append(maxPoints)
        print(sets)

        for point in maxPoints:
            for p in range(len(points)):
                if euclideanDistance(point, points[p]) == 0:
                    points.pop(p)
                    break
        aidDropCount = aidDropCount - 1
    print(sets)
    return sets
```

‘SetCenters’ is an important function because after sorting the points into different sets, we still need to determine where to deploy aid. It would make the most sense to drop our aid to the center of the set as opposed to the edge of the set. To do this, our ‘setCenters’ function iterates through each set of points and calculates the average location of all the points, which should be closer to where there are more humans.

```

def setCenters(sets):
    """
    Find center of each set.

    Input: 'sets' - a set of sets each containing the different values assoc.
    Output: 'centers' - a list of centers of the circles.
    """
    centers = []
    for setter in sets:
        x = []
        y = []
        for s in setter:
            x.append(s[0])
            y.append(s[1])
        avgX = np.average(x)
        avgY = np.average(y)
        centers.append((int(avgX), int(avgY)))
    return centers

```

The 'visualizeRadius' takes in the locations of the centers of the sets as we had calculated and draws the results onto the images so that we can directly see where the aid will be deployed.

```

def visualizeRadius(imgFile, centers, radius):
    """
    Draw centers and corresponding circles on img.

    Inputs: 'imgFile' - the image file we want to edit.
           'centers' - the list of centers of the circles.
           'radius' - radius for the circles.
    Output: 'tempImg' - the edited image.
    """
    tempImg = cv.imread(imgFile)
    for center in centers:
        circle = cv.circle(tempImg, (center[1], center[0]), radius, (0, 255, 0), 5)
        cv.imwrite(imgFile, circle)
        tempImg = cv.imread(imgFile)
    return tempImg

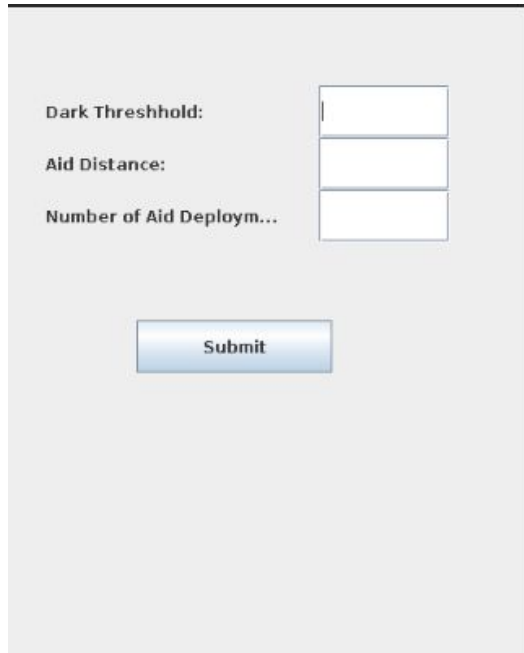
```

We decided to build a graphical user interface (GUI) in Java, and to get the inputs that a user can put into the GUI, we used an Args Parse solution in Python. This simplifies the

process of running the program as opposed to booting up the Terminal and manually executing the program. By doing this, the GUI and the algorithmic program can communicate with what is needed and what the optimal solution is based on the needs provided.

```
// add event listener
submit.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Clicked!");
        String a = darkThreshInput.getText();
        String b = aidDistInput.getText();
        String c = numberAidInput.getText();
        Process p = Runtime.getRuntime().exec("python beachTrashSetCover.py
    }
});
```

```
parser = argparse.ArgumentParser(description=" Determine Optimal Positions for
parser.add_argument('--file', dest='file', required=True)
parser.add_argument('--aidDropCount', dest='aidDropCount', required=False)
parser.add_argument('--darkThresh', dest='darkThresh', required=False)
parser.add_argument('--aidDist',dest='aidDist', required=False)
args = parser.parse_args()
imgFile = args.file
```



A simple GUI form with a light gray background. It contains three input fields stacked vertically on the right side. To the left of each input field is a label: 'Dark Threshold:', 'Aid Distance:', and 'Number of Aid Deploym...'. Below the input fields is a blue 'Submit' button.

Dark Threshold:	<input type="text"/>
Aid Distance:	<input type="text"/>
Number of Aid Deploym...	<input type="text"/>

The long term goal of this project is to replace the GUI with an automated system so that whenever the drone detects human activity, it can automatically send the information and inputs over to the algorithmic program to calculate for the optimal deployment location.

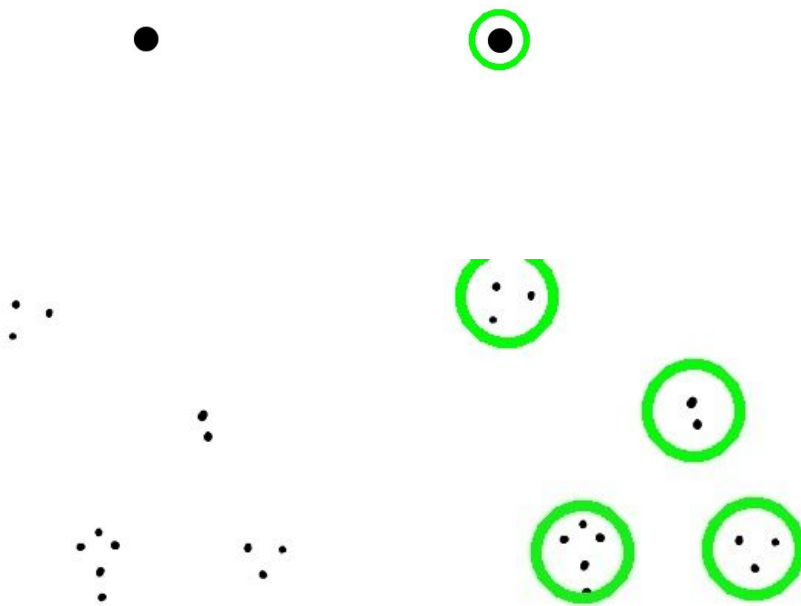
Note: there are many other functions that had been implemented in the making of this program but have not been included.

Please view the following Github link for the entire program:

<https://github.com/bryantlta/Optimizing-Aid-Deployment->

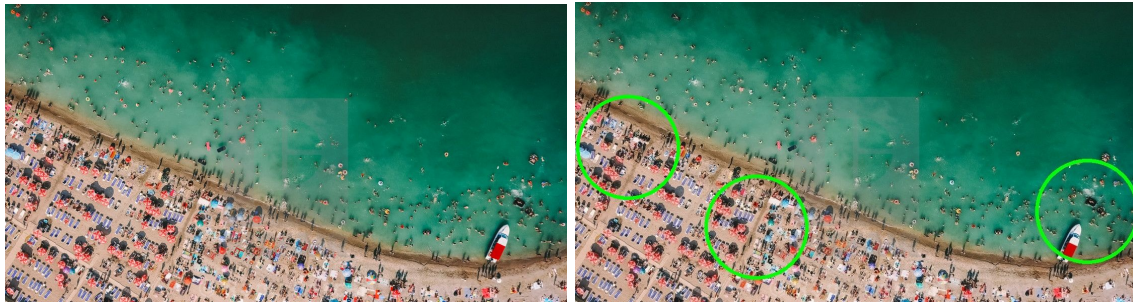
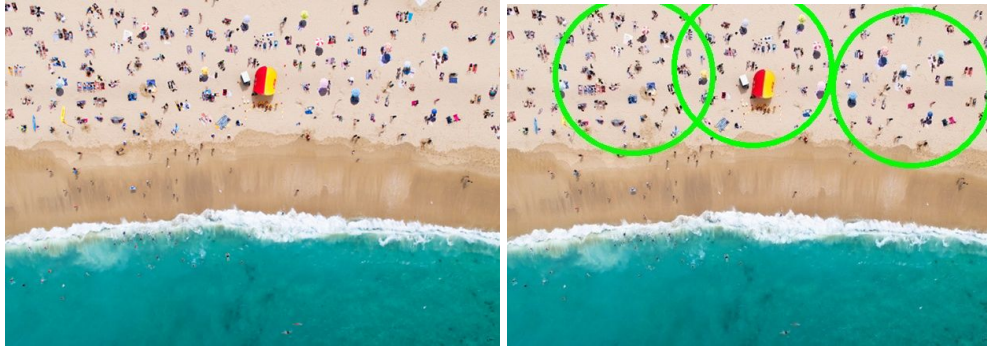
## Program Results:

To test the accuracy of our algorithm, we wanted to test on simple binary images before exhausting our program on more complicated cases. The following are the results of running our set cover algorithm on binary images:



The experimental results show that our program worked perfectly well, finding very good solutions by using a greedy algorithm. The following are the results of running our set cover algorithm on complicated images with human activity:





Our program performed nearly optimal on the set of images. Beach images were chosen with the purpose of testing whether our program would be able to detect human activity in the ocean. This is important because during natural disasters, such as Hurricane Katrina, most grounds were flooded and it would have been important to have an algorithm that could have detected human activity in water in order to deploy aid.



## Conclusion and Reflection:

Overall, we believe our program addresses our main objective and would greatly help decision makers in resource distribution. Our main goal is to help the greatest number of people who are in need, and this program effectively resolves this concern.

One future improvement we would like to work on is optimizing our program not just to help the most populated areas but also areas most at risk. For example, how could our program best help young children or the elderly, who are less likely to help themselves during a time of crisis?

Although we were able to find spots of aid deployments with nearly optimal results, the algorithm still requires manual inputs into the GUI and the algorithm's performance is still uncertain at night when it is hard for even the human eye to find human silhouettes. We want to eliminate the need for the GUI and make everything automatic so that when the drone flies by, it is able to instantly classify, detect, and find optimal aid deployment spots. We want to optimize this so that we would be able to send aid to people right away whether it is during the day or night. We can optimize this through machine learning and artificial intelligence, which could be used to automatically tune the thresholding settings for classifying human activity.

We can extend this project for natural disasters beyond the circumstances of Hurricane Katrina. For example, it can be used to determine optimal locations to place trash cans to reduce beach pollution. It can also be used to find optimal locations to drop water that would reduce forest fires at their source.

## Bibliography:

Gabe, Thomas, et al. "Hurricane Katrina: Social-Demographic Characteristics of Impacted Areas." *Hurricane Katrina: Social-Demographic Characteristics of Impacted Areas*, Congressional Research Service, 4 Nov. 2005, congressionalresearch.com/RL33141/document.php.

"Hurricane Katrina." *History.com*, A&E Television Networks, 9 Nov. 2009, www.history.com/topics/natural-disasters-and-environment/hurricane-katrina.

"Hurricane Katrina Statistics Fast Facts." *CNN*, Cable News Network, 30 Aug. 2018, www.cnn.com/2013/08/23/us/hurricane-katrina-statistics-fast-facts/index.html.

S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani. "Algorithms." 2006, http://algorithmics.lsi.upc.edu/docs/Dasgupta-Papadimitriou-Vazirani.pdf