UBC Mathematics Summer Reading Program 2024
Convex Optimization

Hilary Lau

August 23, 2024

1 Introduction

Resource allocation problems, or more generally, placement problems, are common problems found both in Operations Research and Optimization. In this paper, we will investigate a kind of placement problem - finding the optimal location placement of one, or multiple, computer servers, so as to minimize user latency. Since our objective function would primarily be based on the Euclidean norm, we want to attempt to limit the objective function and feasible region to be convex, so that we can utilize convex optimization techniques to solve this problem.

2 Theoretical Analysis

Consider a square city of arbitrary size as our convex domain D. Let $a_i \in D$ be some user.

We assume that the latency of the user is perfectly and linearly proportional to the distance between the server and the user. Then, the latency of each user can be estimated using the Euclidean norm, or the square of the Euclidean norm, since both would give equivalent placements. In this problem, we are more interested in the values of the variables that optimize the objective function, rather than the value of the objective function itself.

2.1 One Server

For simplicity, we first investigate the problem when we only have one single server to place. Assume $x \in D$ is the location of the server.

To find the optimal location of the one server for n users, we could use the objective function

$$\min_{x \in D} \sum_{i=1}^{n} ||x - a_i||$$

Since norms are convex, the sum of norms is still convex, and this is a convex optimization problem that can be solved (see 3.1). In fact, this optimization problem solves for the geometric median of the set of points (this is different from the centroid or the center of mass), and there does not exist a closed form formula to solve for this point for number of points greater than 5 (shown using Galois Theory) [1].

2.2 Many Servers

If we have m servers $x_1, \cdots, x_m \in D$, this could be considered a min-min problem, where our objective could be

$$\min_{x_1,\cdots,x_m \in D} \sum_{i=1}^{n} \min\{||x_1 - a_i||, \cdots, ||x_m - a_i||\}$$

However, this objective function is no longer convex, since the minimum function is not a convex function. In fact, this is a variation of the k-median problem, where k is the number of servers in our case. The k-median problem is as follows: given points are divided into k partitions, where the sum of the distances between each point and the center of its partition is minimzed (Voronoi cells). This problem is known to be NP-hard, but research has shown it written out as an integer program, relaxed into a linear program, then solved and rounded to for a good approximation [2].

3 Algorithmic Solution

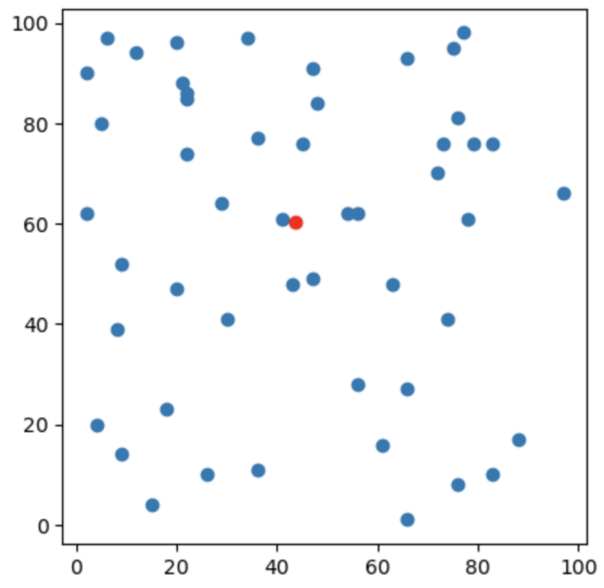
3.1 One Server Solution with cvxpy

To solve the one server problem, we generated a randomly distributed set of users in Python using `numpy`[3] that lie within D, taking D to be a square of side length 100 for this situation. Then, using the `cvxpy` Python solver package[4, 5], we were able to find the optimal solution using the code below:

```
x = cp.Variable((1, 2))
obj = cp.Minimize(cp.sum(cp.norm(coords - x, 2, axis=1)))
problem = cp.Problem(obj)
problem.solve()
```

Where `obj` is the objective function that was described to be convex in 2.1.

This solved for the following optimal solution (shown in red) on the map:



3.2 One Server Solution with Gradient Descent

We can also implement the solution naively with the gradient descent algorithm [6]. Here, we slightly modify the objective function to use the squared Euclidean norm to get nicer partial derivatives. First find the gradient:

$$\begin{aligned}\nabla \sum_{i=1}^{n} ||x - a_i||^2 &= \nabla \sum_{i=1}^{n} ((x_1 - a_{i_1})^2 + (x_2 - a_{i_2})^2) \\ &= \begin{bmatrix} \sum_{i=1}^{n} 2(x_1 - a_{i_1}) \\ \sum_{i=1}^{n} 2(x_2 - a_{i_2}) \end{bmatrix}\end{aligned}$$

We implemented this calculation as the Python function `grad`, and the calculation of the objective value as the Python function `obj`.

Then recall this pseudocode for gradient descent:

```
$x \leftarrow$ some coordinate in D
**while** $\nabla f(x) > \varepsilon$ **do**
    $\Delta x \leftarrow -\nabla f(x)$
    Find optimal $t$ step size
    $x \leftarrow x + t\Delta x$
**end while**
```

Then, we implemented gradient descent, using $\varepsilon = 10^{-6}$ as our stopping condition for the norm of the gradient. To find the optimal t step size, we also implemented backtracking line search (See [6]) where we start with step size being 1, and shrink it until a stopping condition is met for the optimal convergence of the gradient descent algorithm according to this pseudocode:

```
$t \leftarrow 1$
**while** $f(x + t\Delta x) > f(x) - t\alpha||\Delta x||^2$ **do**
    $t \leftarrow \beta t$
**end while**
```

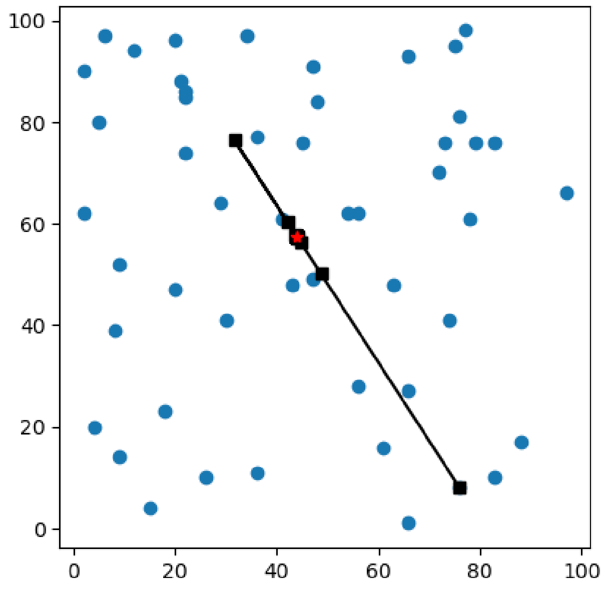
With $\alpha = 0.3$ and $\beta = 0.7$. Note that we manipulated the right hand side of the stopping condition for backtracking line search in [6] as follows:

$$\begin{aligned}&f(x) + t\alpha\nabla f(x)^T \Delta x \\ =&f(x) - t\alpha\nabla f(x)^T(\nabla f(x)) \\ =&f(x) - t\alpha||\Delta x||^2\end{aligned}$$

This resulted in the following code for gradient descent:

```python
x = coords[0]
steps = []
ep = 1e-6
beta = 0.7
alpha = 0.5
while np.linalg.norm(grad(x)) > ep:
  steps.append(x)
  del_x = -grad(x)
  t = 1
  while obj(x + t*del_x) > (obj(x) - (t * alpha) * np.linalg.norm(del_x)**2):
    t = beta * t
  x = x + t * del_x
```

Running this code took 892 iterations and resulted in the same optimal server placement as calcualted by `cvxpy` in 3.1. By plotting `steps`, we are able to see the progress of gradient descent:



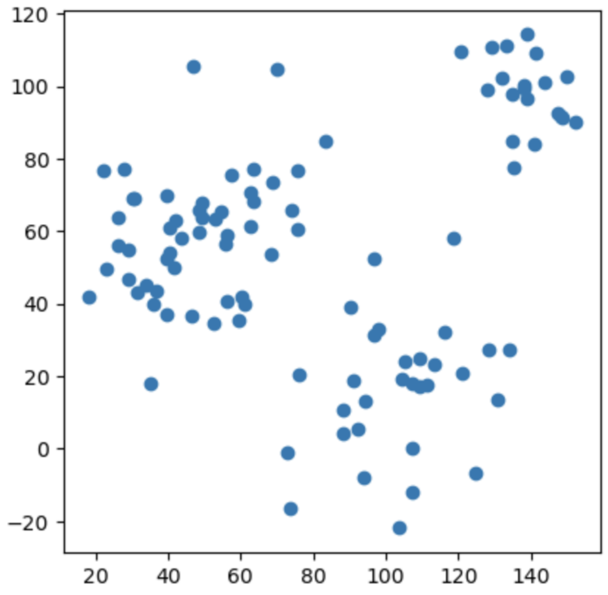
Since the gradient is linear, gradient descent approached the optimal solution in a line as expected.

3.3 Multiple Server Solution with k-medians

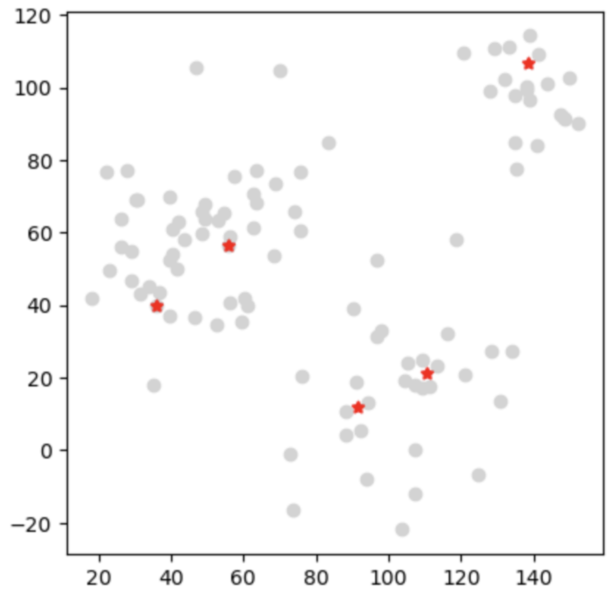
To approximate a solution for multiple servers, the `pyclustering` Python package[7] was used, which included a k-medians approximation algorithm based on a greedy approach.

For this test, we chose $k = 5$, to place 5 servers into the city.

Firstly, the users dataset was generated using `numpy` as a combination of normally distributed points around three different centers. This could represent different population concentrations around a city.



Then, 5 initial cluster centers were randomly chosen out of all the points in order to start the iterative greedy algorithm, which yielded appropriate clustering and placement of the servers.



This is a simpler method than rounding LP method described in 2.2. However, this algorithm for placement of the servers has a large margin of error, and is especially sensitive based on the randomized inital cluster centers. The algorithm based on the rounded linear program could provide more accurate results with tradeoffs to developer time and running time.

4 Summary

Here we investigated the optimal placement of one server in a city using convex optimization methods, as well as the potential optimal placement of multiple servers using a greedy approximation. As further analysis, we could investigate the accuracy of using the rounded linear program as a good approximation for the multiple server problem.

References

[1] C. Bajaj, "Proving geometric algorithm non-solvability: An application of factoring polynomials," *Journal of Symbolic Computation*, vol. 2, no. 1, pp. 99–102, 1986.

[2] M. Charikar, S. Guha, Éva Tardos, and D. B. Shmoys, "A constant-factor approximation algorithm for the k-median problem," *Journal of Computer and System Sciences*, vol. 65, no. 1, pp. 129–149, 2002.

[3] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[4] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *Journal of Control and Decision*, vol. 5, no. 1, pp. 42–60, 2018.

[5] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[6] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2004.

[7] "Pyclustering: Data mining library," *Journal of Open Source Software*.