# INTRO

[Overleaf document](https://www.overleaf.com/project/66c7587508c6c67137a00bd9)

[Variable Combinations](https://docs.google.com/spreadsheets/d/17JAFaahG1gZ_C45X5_xIexmCkbB2RfyYEpFWfhxLqCA/edit)

Journal Vs Conference

**Results**

Philadelphia additional tract ratios

Columbus additional loads

Chicago 1:5

Chicago mini and the comparison to deliver AI

New York

**Pareto Solutions**

**Ways and Nodes used for locational data instead of only ways**

**Comparison of Qcenses correctness**

**Scalabity analysis (size of city, vs populations, vs census tract number**

**Journal Checklist (**Decided on 21th March 2025**):**

1. **Related Works Section** – Add more papers (*Everyone*)
2. **Graphs** – Use **TikZ** and include Standard Deviation (SD) of all repetitions (*Nahid*)
3. **Results** – Show **Chebyshev** Inequality analysis - (*Need to decide*)
4. **Pareto Frontie**r Solutions – *(Ashman, Yusuf)*
5. **Problem Definition** – Provide detailed explanation with a possible image (*Robert*)
6. **Solution Approach** – Explain in detail (*Robert, Yusuf*)

# 

# JOURNAL PAPER TODO

* ~~Set up docker colab~~
* ~~Start running q table for new york~~
* ~~show Nahid how to generate philadelphia~~
* **~~Write email to Graphhopper - Oct 25, 2024!!~~**
* ~~Get graphhopper data for NYC - 1:10, 1:15 Oct 27, 2024~~
* ~~Get “node” working on Overpass API query~~
* Delivery sets
  + Generate 9 additional delivery sets for Philadelphia ratios and get results ([Robert Kilgore](mailto:kilgorrj@miamioh.edu))
  + ~~Collect Different Loads for Columbus (~~[~~Yusuf Ozdemir~~](mailto:ozdemiye@miamioh.edu)~~)~~
    - ~~350~~
    - ~~400~~
    - ~~450~~
    - ~~500~~
  + ~~Simulate Different Loads for Columbus (~~[~~Yusuf Ozdemir~~](mailto:ozdemiye@miamioh.edu)~~)~~
    - ~~350~~
    - ~~400~~
    - ~~450~~
    - ~~500~~
  + Different order sets
* Data Collection (graphhopper)
  + Ratios
    - Chicago 1:5 (verify it is correct)
    - ~~Test cluster sizes of 1:20, 1:30, 1:40 Philadelphia~~
    - ~~Identify best ratio~~
  + Other Cities
    - San francisco / Seattle / Portland
    - Salt Lake City
    - ~~Santa Monica~~
    - ~~DC~~
    - ~~New York~~ *~~(larger)~~*
* Q-Table Generation
  + ~~New York~~
    - ~~1:10~~
    - ~~1:15~~
  + Chicago
    - 1:5
* Potential improvements to the success rate in our implementation?
  + Increase success rate by modifying simulation?
  + A lower success rate skews total distance/average time as failed deliveries are not counted.
* Include percentage of shares (instead of hops?)
  + Percentage of distance traveled using shares
* Track CPU and Memory Resource Usage (Imp.)
  + <https://pypi.org/project/memory-profiler/>
* Plotting Pareto Solutions and comparing DeliverAI and QuikDel with Pareto frontiers
* Messages exchanged between the agents and handlers
* Establishing Markov and Chebyshev inequalities for the simulation metrics
* Running multiple times at different times of the day, considering traffic considerations, and taking average (Imp.)
* Compare performance with other papers (?) (DeliverAI?)
* Ratio of population vs census tracts (scalability analysis)
* Test with a large cluster (for one particular city) - check all metrics and computation costs and find the pareto front. - Philadelphia + Ashman
* Ashman finds a matching venue and works on the Pareto front. + find population of cities (<https://worldpopulationreview.com/us-cities>)
* Genetic algo, Ant colony or other M.O.O techniques ([f20212508@goa.bits-pilani.ac.in](mailto:f20212508@goa.bits-pilani.ac.in))

# 

PROCESS OF SUPER-SPOT SELECTION

1. For each census tract, get the number of producers, consumers and bordering census tracts.
2. Min-max normalize these three values and get their mean. This creates a score for each tract.
3. Pick the super-spots based on score, highest to lowest.
4. To avoid bordering super-spots (for even distribution throughout the graph), we pick super-spots in descending order by score. If a selected hotspot already borders an existing super-spot, it is skipped and not added as a super-spot.
   1. The number of super-spots, *n*, selected can be changed as needed. Currently, we use , where *x* is the number of census tracts in the city.
   2. We decided on this ratio due to our preliminary tests. It provides enough hotspots to cover a city, but not too many that it adds an excessive amount.
   3. The grouping process will remove extra hotspots if necessary.

PROCESS OF HOTSPOT GROUPING UNDER SUPER-SPOTS

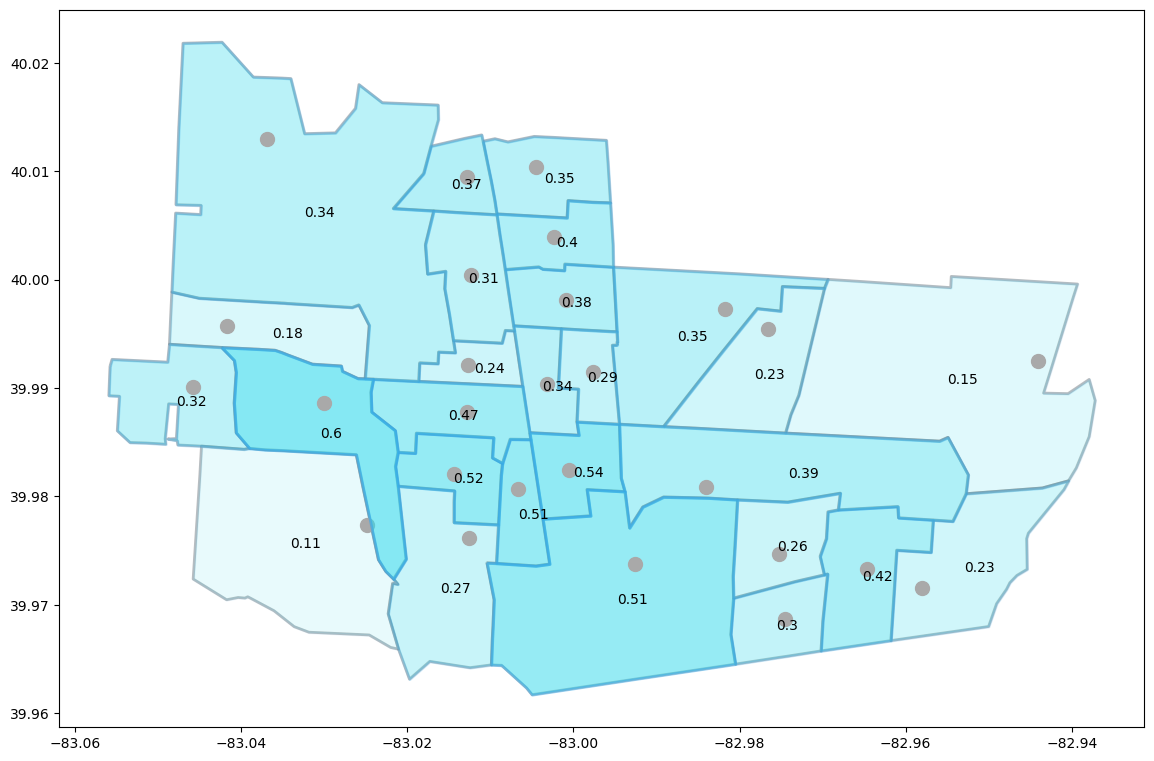
1. All hotspots except for the super-spots are added to a queue
2. Moving through the queue, each hotspot is assigned to its nearest super-spot based on the distance between the two from their centroids.
3. Now that all super-spots have hotspots as children, any super-spot with less than a minimum number of children, *k*, is removed as a super-spot. It and its children are reallocated to their new nearest super-spots.
   1. This reduces the number of unnecessary super-spots in the graph.

Our goal is to select the best second-layer hotspots (which will be referred to as super-spots from now on) from the given hotspots in a city. The end goal is to have a layer of selected super-spots that is evenly distributed throughout the city, with the ultimate goal being to group all hotspots in a city effectively under a smaller number of super-spots. This way, we can abstract delivery routes between super-spots to reduce computational costs, moving to the regular hotspot layer when necessary.

To select the best possible super-spots, we need some sort of scoring system to determine which hotspots are the most eligible to become super-spots. To create this scoring method, we will have to utilize the data we have about each hotspot. In this situation, we had data about the number of consumers and producers in each hotspot at our disposal. Although we only had this data to work with, additional data about each hotspot could easily be utilized to further improve hotspot scoring as necessary.

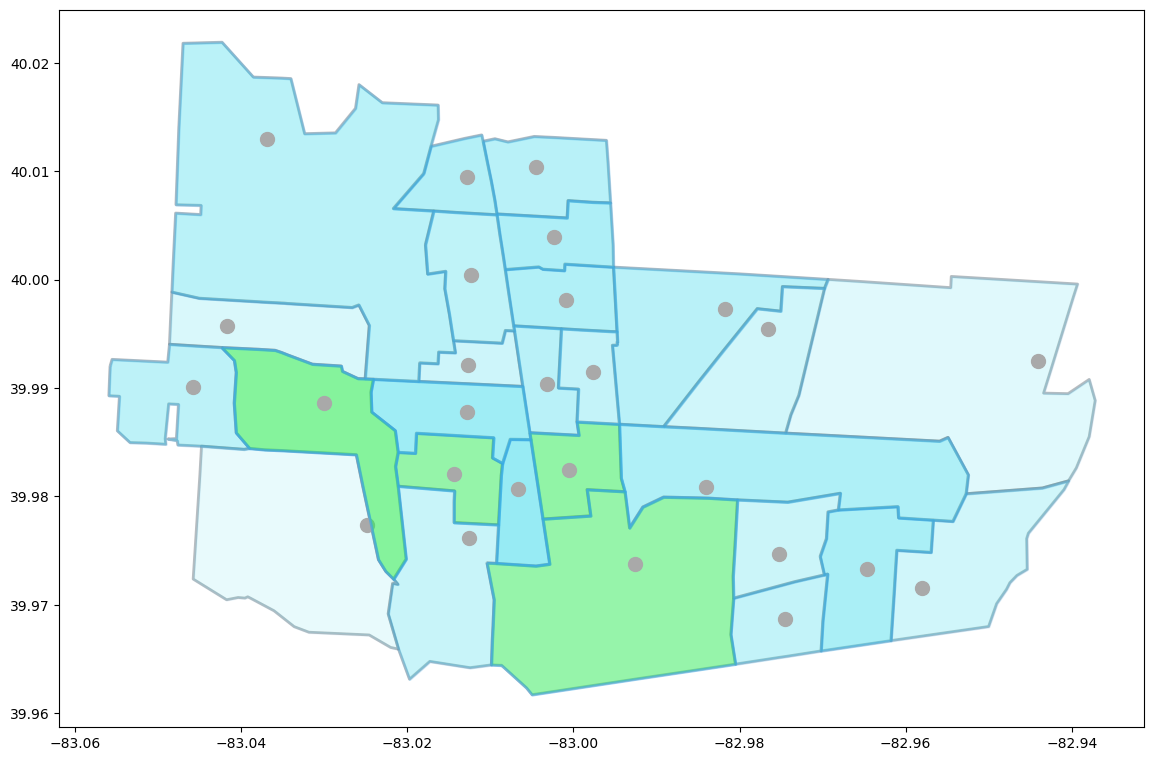
To compute the score of each hotspot, we looked at three different data values: the number of consumers, the number of producers, and the number of bordering census tracts for each hotspot. All of these values were min-max normalized to put them all on a scale of 0-1. Then, the mean of these three values was calculated. This gave us a score for each hotspot between 0 and 1 that could easily be compared to each other.

In Figure 1, you can see a subset of the census tracts in Columbus, Ohio with scores assigned to each hotspot.



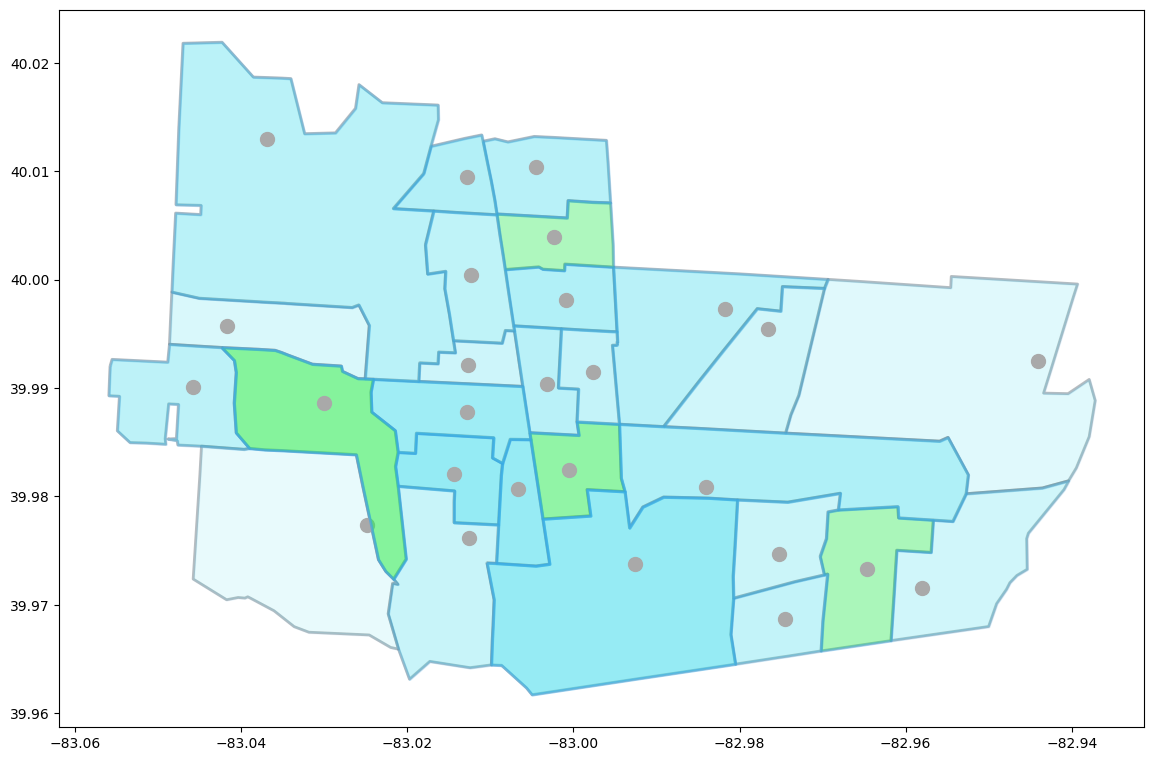
*Figure 1*

Now that each tract has a score, the best ones need to be chosen as the super-spots. The easy thing to do would be to just pick the best scoring hotspots as the super-spots directly. This method was tried first. We selected four super-spots in our test run (we call the number of super-spots to choose *n*). You can see which super-spots were selected in Figure 2 using this method.



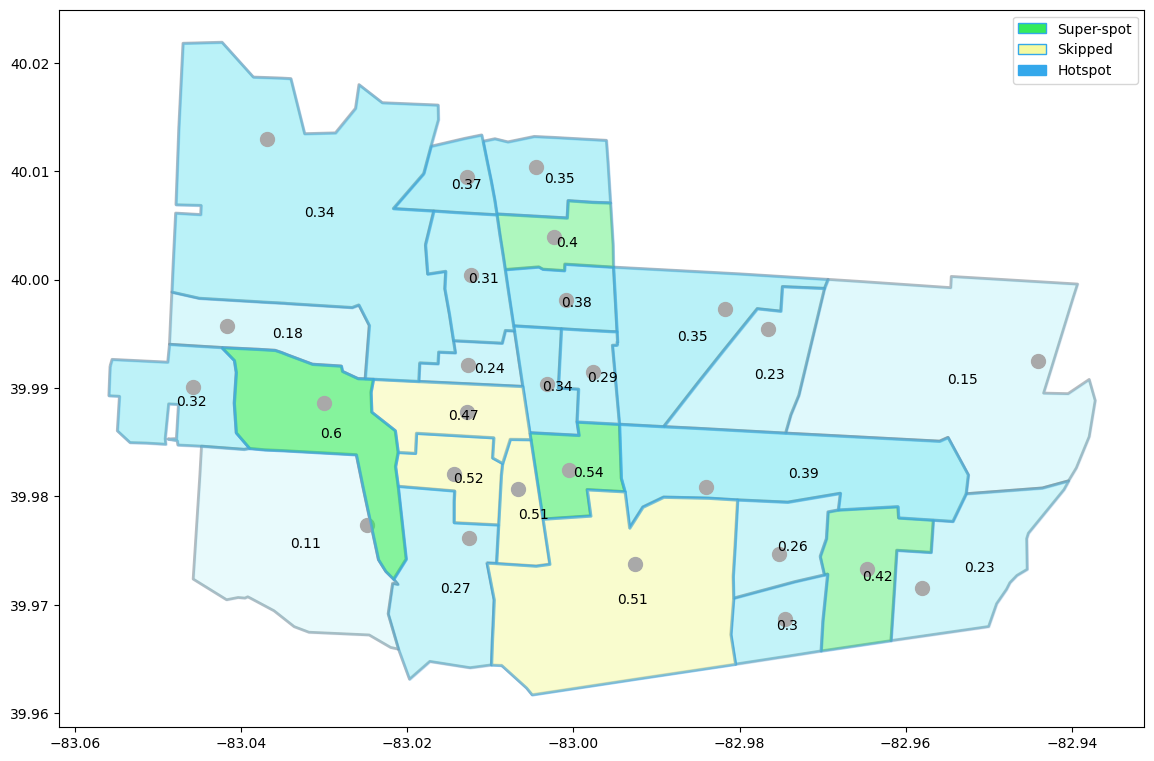
*Figure 2*

The results using this method of super-spot selection aren’t great. The issue is that these super-spots are clustered very close together -- however, we want them to be evenly distributed across all the hotspots. That way, when the hotspots are grouped together under the super-spots, they are grouped in consistent and even groups. To solve this major issue, we made sure super-spots did not share any borders. First, we ordered all hotspots by their scores, in descending order. We then checked each of those hotspots. If the hotspot did not border any existing super-spot, it would be made into a super-spot. This would be done until we had *n* super-spots or no hotspots remaining. We used this new method and selected the super-spots again. The resulting super-spot selection is visible in Figure 3.



*Figure 3*

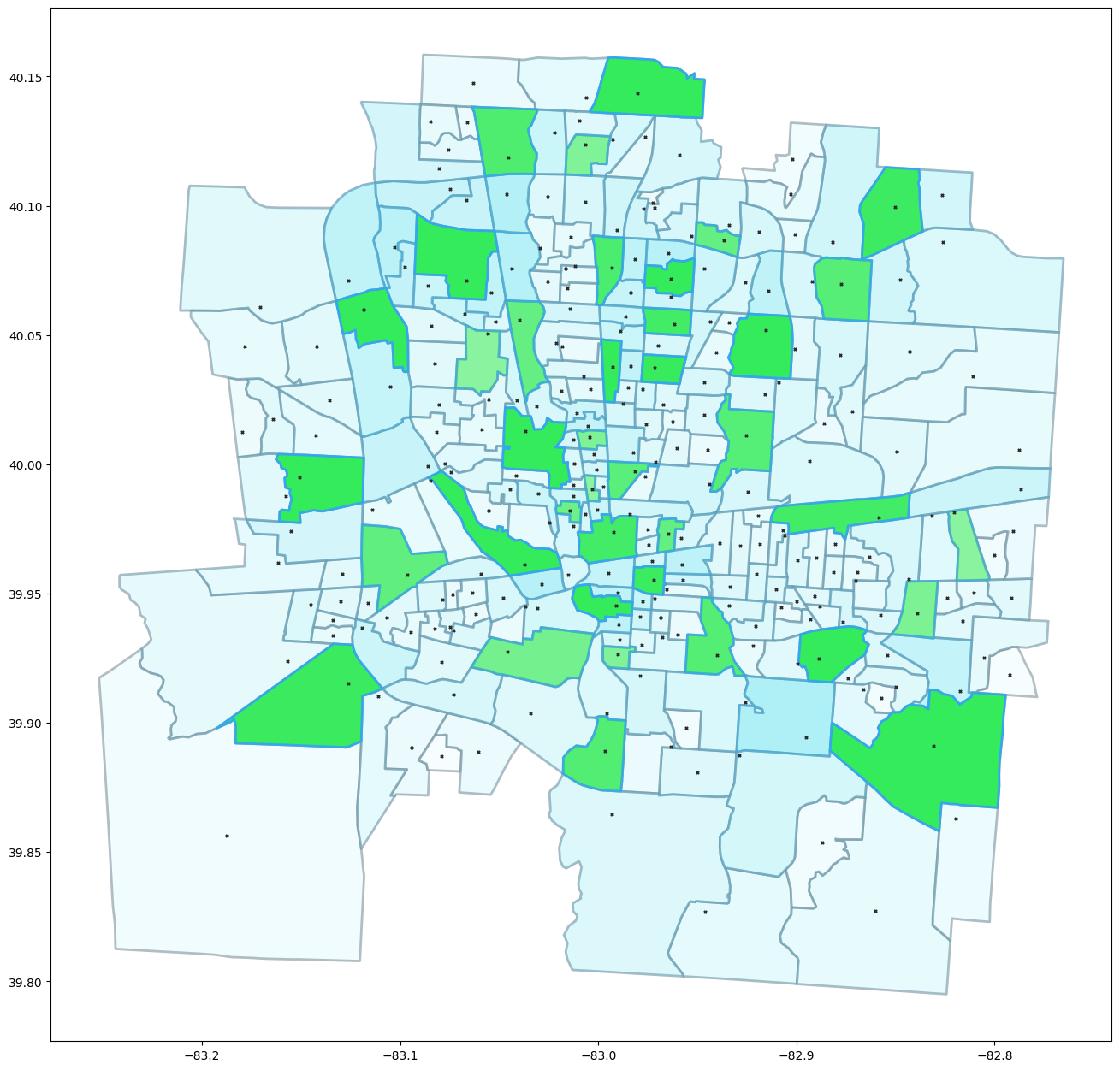
This result is a lot better. It evenly distributes the super-spots throughout the entirety of the city, so that when the rest of the hotspots are grouped under the super-spots, each one will have a super-spot in close proximity. Figure 4 shows which hotspots were skipped in the super-spot selection.



*Figure 4*

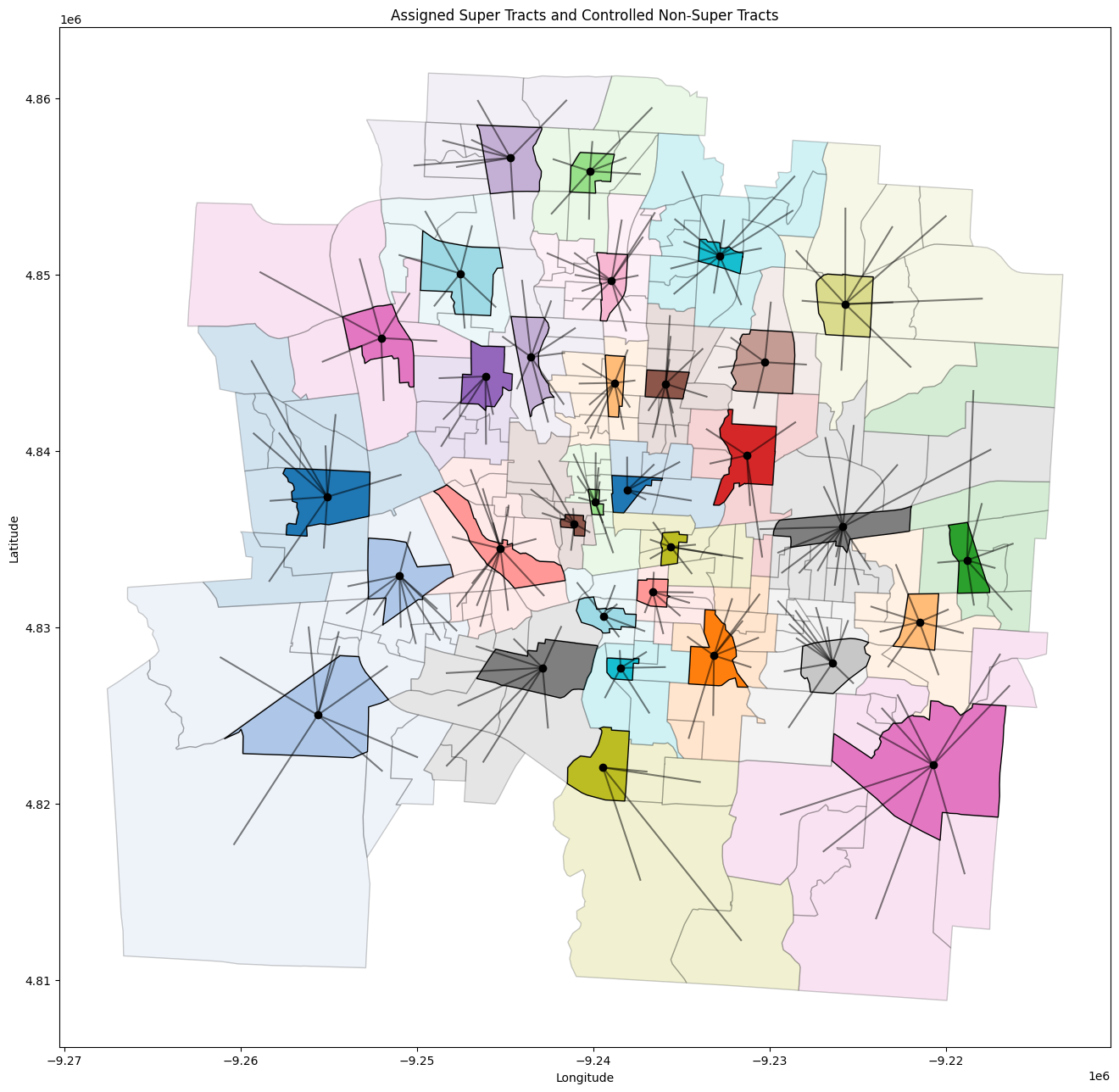
The number of super-spots selected can be changed depending on necessity. This method for super-spot selection can easily be used to select any number of super-spots, making it extremely versatile.

Now that we have selected super-spots from our list of hotspots, we need to group the remaining hotspots under these super-spots. Hotspots should be grouped under super-spots by distance, so that when a delivery entity travels to a super-spot, its next move to a subsequent hotspot should be as short as possible. This is intuitive. To assign hotspots to their super-spots, we use Depth First Search. Each hotspot is assigned to its nearest super-spot by distance. Distance is measured from the center of a super-spot’s census tract to the center of the census tract of a hotspot. Let’s look at the full map of Columbus in figure 5 with all super-spots selected.



*Figure 5*

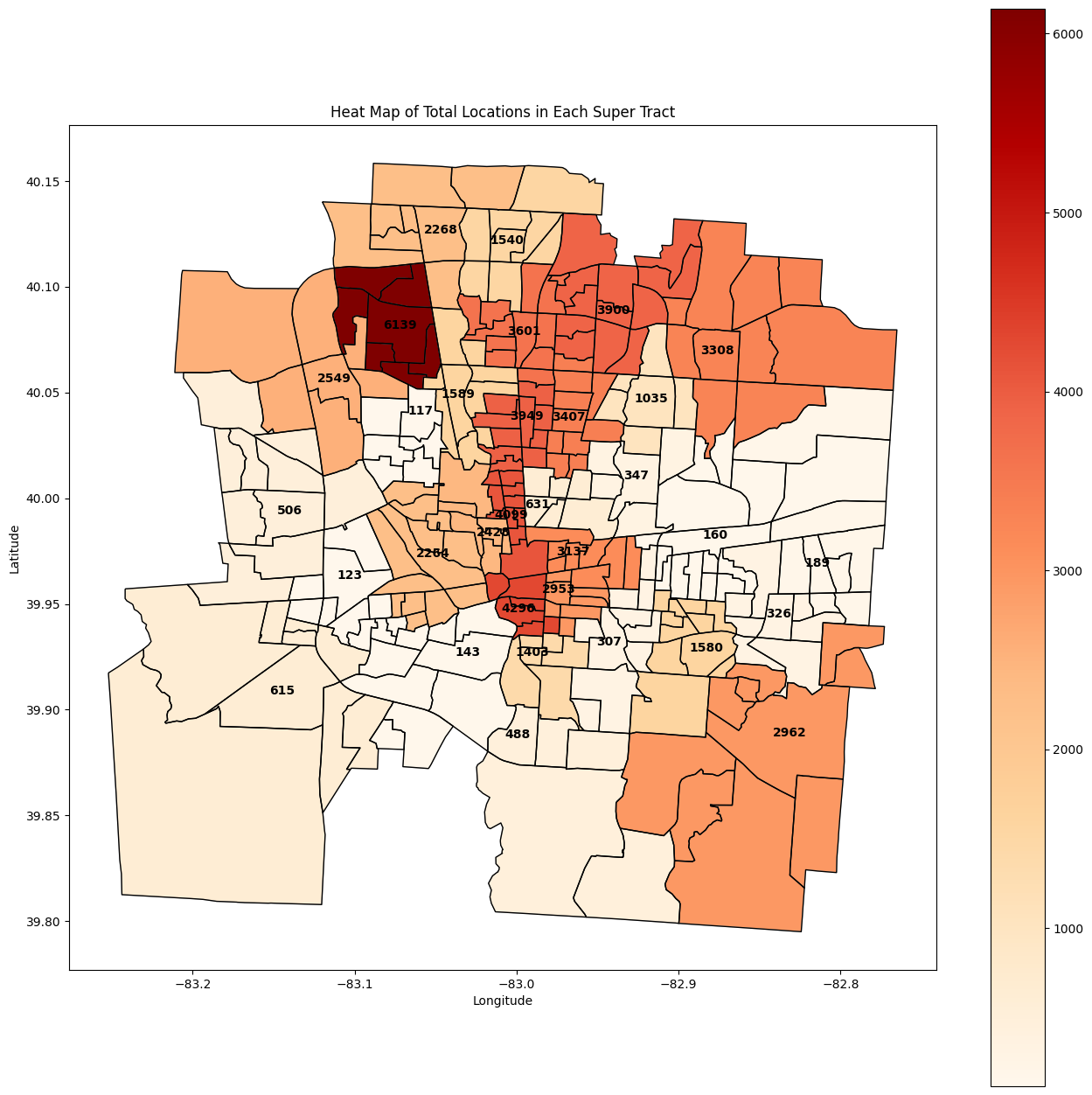
Looking at figure 5, we can see that in certain areas there are several super-spots close together. This occurs mainly in the clusters of smaller census tracts. This is because these regions are more densely populated, and as such have more businesses and homes. They are scored higher, and more super-spots appear there. However, many of these super-spots will end up being largely redundant. Their close proximity to other super-spots makes them unnecessary. To address this issue, we set a threshold of the minimum number of children a super-spot must have. Once the hotspots are allocated to the super-spots, if a super-spot does not meet this threshold, it is removed and the children are allocated to surrounding super-spots. This reduces the number of unnecessary super-spots from the map. In figure 6, you can see how hotspots are grouped under super-spots using our method.



*Figure 6*

In figure 6, we specified that the minimum number of children a super-spot must have was 4. The resulting groups of hotspots are evenly distributed throughout the city. If you compare the super-spots in figure 5 to figure 6, you can see that there are fewer super-spots in the central, denser areas in the final grouping.

In figure 7 we have a heatmap that displays the total number of producers and consumers in each cluster of hotspots.



*Figure 7*

The number of locations is not perfectly distributed between hotspot clusters. However, this is natural. Cities have central downtown areas that contain many more locations (consumers and producers) than other less populated areas. This map gives us an idea of which hotspot clusters are more densely populated in the city.

# Q TABLE WRITEUP

Now that we have grouped the census tracts under their respective super-spots, we need to construct Q tables for pathing between hotspots. One Q table will be constructed for each hotspot cluster, and an additional Q table will be created for pathing between super-spots only. These Q tables are stored in a dictionary, where the key is the index of the super-spot of the group, and the value contains the Q table. In the case of the super-spot Q table, the key is always *-1*.

Something important to note is the concept of indexing census tracts, both globally and locally. Census tracts are identified by their unique GeoID, which is a sequence of numbers. However when storing data about census tracts in data structures like matrices, from which you access data by specifying an index from 0 to its dimensional size, you cannot use GeoIDs for this. Before the Q table is generated, all GeoIDs are paired with a **global** index that can be used to identify them if necessary. This allows for consistency to make sure data is never mixed up between census tracts. However in the case of the Q tables, these global indices do not work. Since Q tables are local to their groups, the first row/column in a group’s Q table could be any GeoID globally. In addition to this, since we have multiple Q tables the indexes 0, 1, 2, … are repeated multiple times across different Q tables. To address this issue, we mapped each **global** index to the **local** index it has inside its Q table. For example, a census tract could have **global** index 6, but within its **local** Q table it is the third row/column. Using this method, we are able to efficiently traverse Q tables in the **local** state while being able to tell the delivery algorithm where it is in the city **globally**.

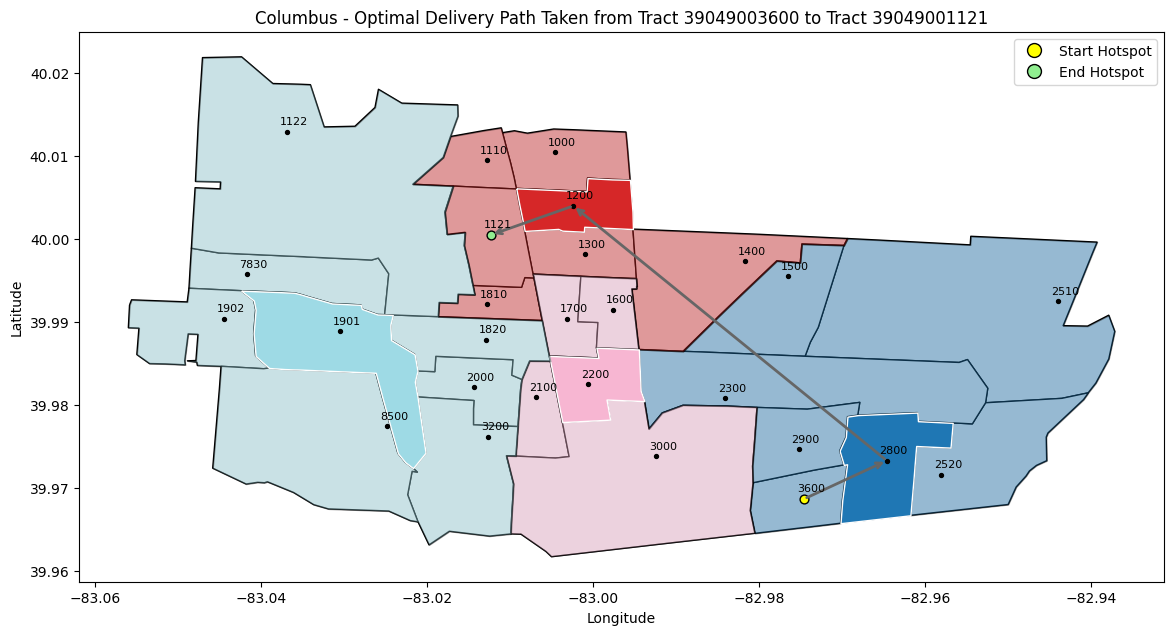
The Q tables are created as discussed in the previous paper, however here they are of course only being created for the hotspots in their own groups.

Now that we have multiple Q tables, we need a simple process to link them all together in order to find paths between hotspots. For hotspots within the same group, path-finding is always a single hop, from the starting point to the destination point. For hotspots in different groups, it is a little more involved.

There are three steps to finding a path between hotspots in different groups.

* **Step 1:** The agent begins at its starting point. Since its final destination is in a different group, it updates its temporary destination to the super-spot of its local group. Using the Q table for its current group, it then finds the path to the super-spot.
* **Step 2:** The agent is now at the super-spot of the starting group. It updates its temporary destination to the super-spot of the group its final destination resides in. It now uses the Q table for the super-spot group to find the path from the starting group’s super-spot to the destination group’s super-spot.
* **Step 3:** The agent is now at the super-spot of the destination group. It updates its destination to the final destination. It now uses the Q table for the destination group in order to find the path to the destination hotspot.

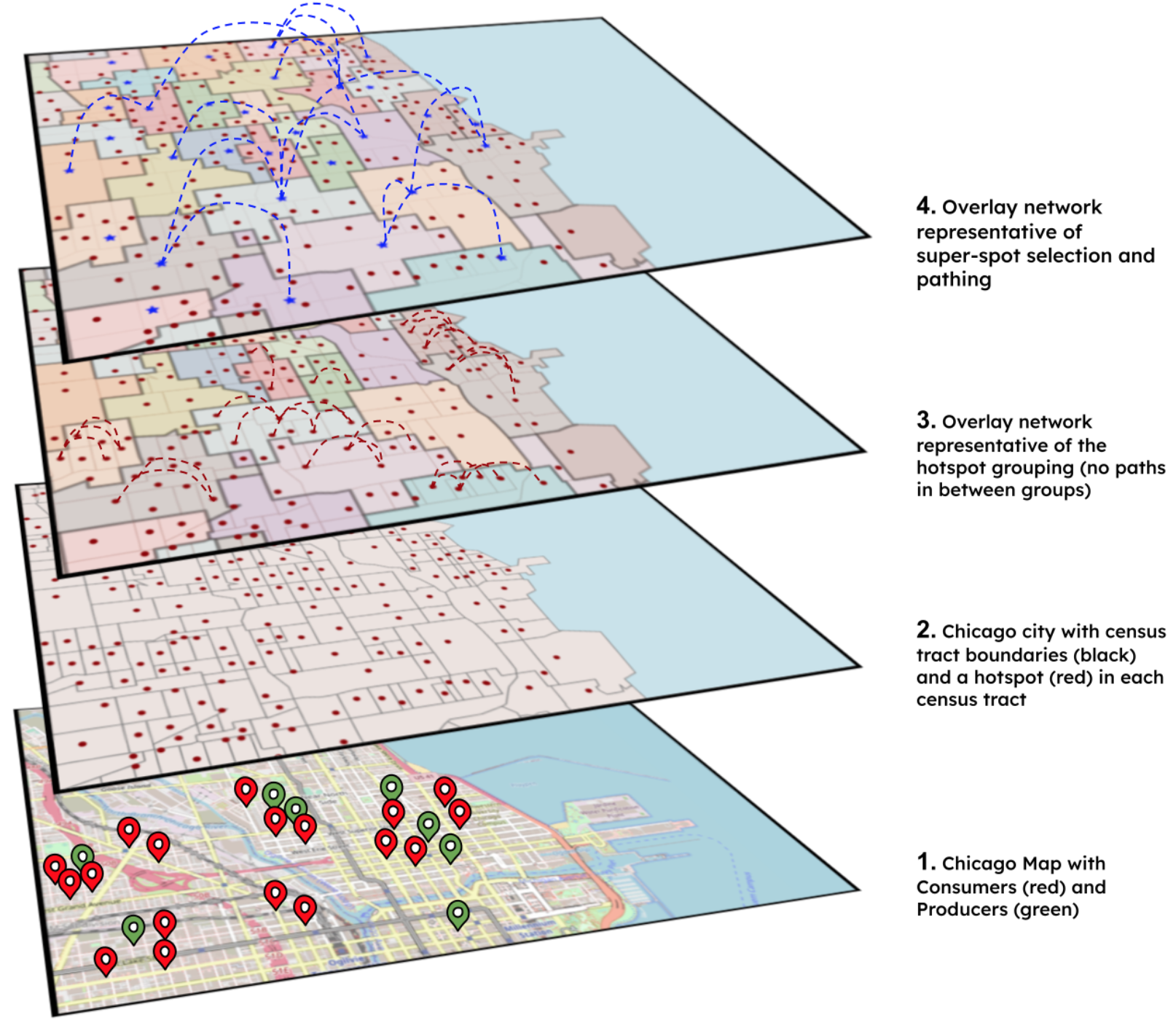
In figure 8, you can see a subset of Columbus. Here, you can see the path that a delivery from tract 39049003600 to tract 39049001121 takes.



*Figure 8*

NOTE ABOUT JUSTIFICATION FOR NOT TRAVELING DIRECTLY FROM PRODUCER TO CONSUMER AND VISITING THE RESPECTIVE HOTSPOTS FIRST

* Most census tracts are small in size, so the overhead of traveling to the hotspot first is nearly insignificant compared to the complete journey.
* The census tracts are placed strategically and close to the consumers to minimize the overhead. The hotspots are placed at the centroid of all consumer locations. This guarantees that the hotspot is placed in a consumer-dominated environment, which allows quick delivery in the last mile of the delivery route.
* The overhead is deemed necessary because otherwise, the vehicles will have complete mobility, which makes fleet management challenging and complex (the simplicity of the PDV and CDV system is lost). This also means that the system will have to account for vehicles that might leave their census tract to deliver orders directly and keep additional vehicles to compensate for that. This can lead to additional costs (more than what is borne with the overhead of going to the hotspot)

**

*Figure 9*

Figure shows the multi-level Overlay Network visualization on the map of Chicago. It shows the step-by-step construction of the proposed architecture.

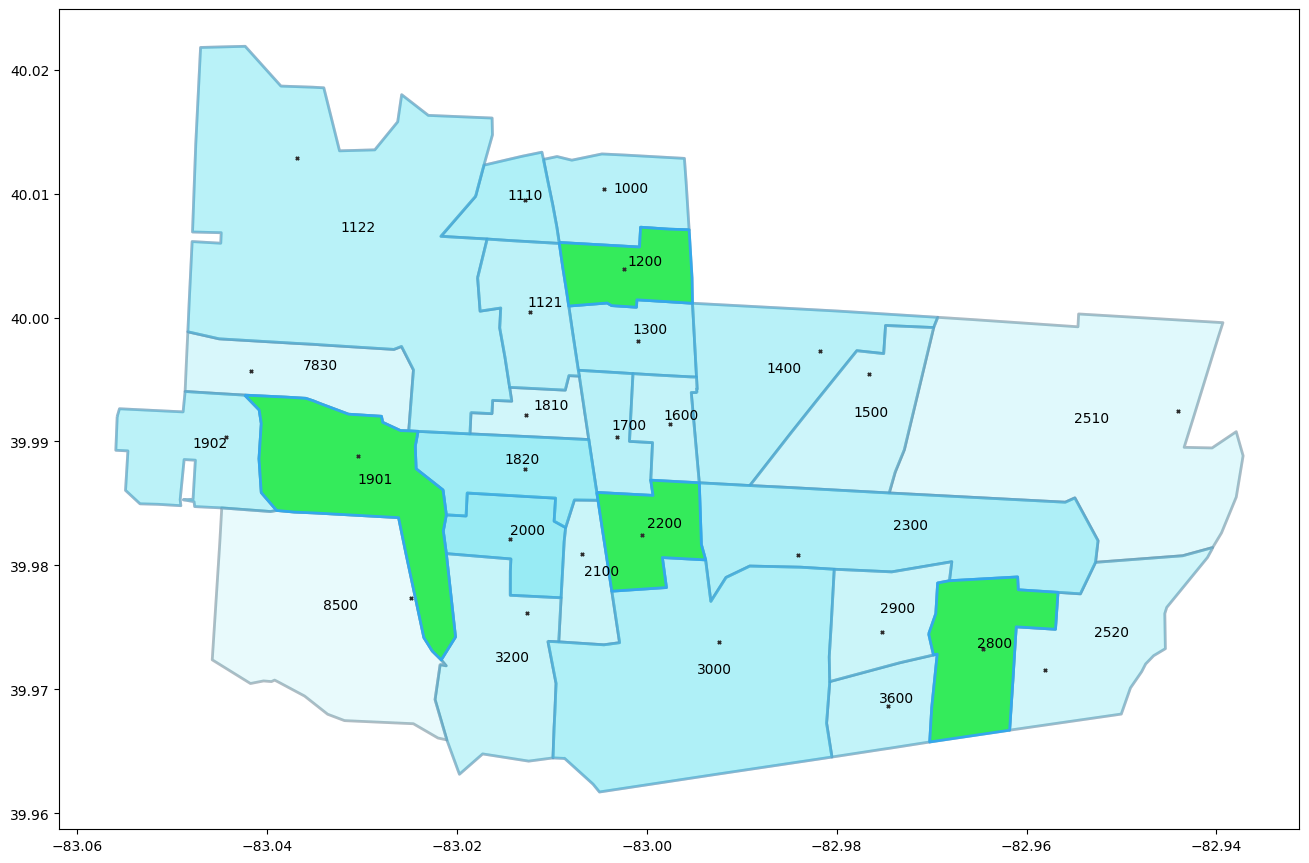
**Layer 1:** Base layer with the raw city map along with the consumer (red pins) and producer (green pins) locations

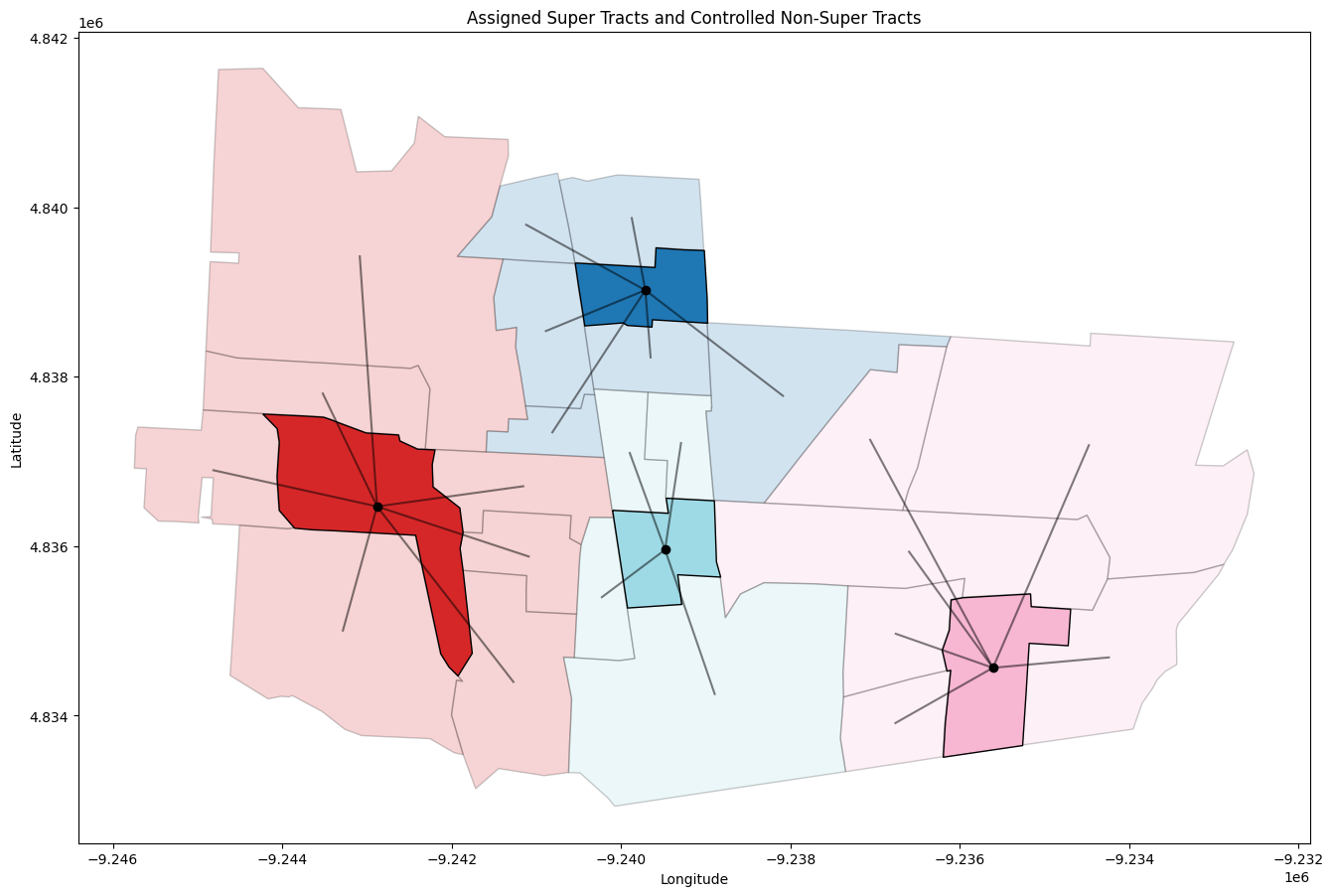
**Layer 2:** This layer shows the division of the city into smaller, non-overlapping and mutually exhaustive units (census tracts). Along with this each census tract has a single hotspot that has been placed optimally at the center of all consumer and producer locations.

**Layer 3:** This layer shows the grouping of census tracts into clusters which we call super-spot clusters. Each census tract (and its hotspot) belong to a single cluster (refer to Section XXX). Each of these super-spot clusters forms a connected clique i.e. each census tract is connected to every other within its super-spot. This is the first level of the overlay network.

**Layer 4:** Each super-spot cluster is assigned a super-spot that is chosen from the hotspots that lie within it (refer to Section XXX). These super-spots form a clique and the 2nd level of the overlay network.

# REFERENCE MAPS





Separate Q-Tables for super-spots and each hotspot group

Routing table (? look into this) to direct the pathing

* If start/end not in same group, use local QT to direct to super-spot
* Super-spot QT to get to destination super-spot (using routing table?)
* Local QT again to get to destination

[Distance Vector Routing (DVR) Protocol - GeeksforGeeks](https://www.geeksforgeeks.org/distance-vector-routing-dvr-protocol/)

<https://docs.ray.io/en/latest/rllib/rllib-replay-buffers.html>

<https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf>

# TIME AND SPACE COMPLEXITY ANALYSIS OF DECISION-MAKING COMPONENTS

| **Component** | **Single Level Hotspot** | **Multi Level Hotspot** |
| --- | --- | --- |
| **Space** for distance/time edges | O(H2 + H\*L) | O(Hs\*hm+ H\*L) |
| **Time** for Training agents | O(H\*Esingle\*tsingle) | O((Hs+hm)\*Emulti\*tmulti) |
| **Space** for storing Q tables | Ө(H3) | O() |
| **Time** for each agent interaction | O(H)  {Note: intersection of 2 PAS can be found using hashing in linear time} | O(hm)1, O(Hs)2  {1. Time within a group, 2. Time in super-spot layer} |
| **Time** for all agent interactions | O(D\*H)  {Note: line 5 occurs parallelly} | O(D\*hm) |
| **Time** taken by Request Handler | O(R+Dt) = O(R)  {Note1: line 18 occurs during agent interaction, so excluded from this}  {Note2: Dt ≤ R, so lines 3-15 and 28,29 are already included in the upper bound} | O( ) |

[Variable Combinations](https://docs.google.com/spreadsheets/d/17JAFaahG1gZ_C45X5_xIexmCkbB2RfyYEpFWfhxLqCA/edit)

NOTE: Here, it is assumed that the Request Handler begins only when all requests are received for the time step under consideration. In the case

KEY AND NOTATIONS

* O = big ‘O’ notation; Ө = theta notation
* H = number of hotspots in the overlay network
* Hs= number of super-spots in the overlay network
* hm= maximum number of hotspots inside a super-spot group
* L = upper bound for the number of locations (consumer/producer) in a hotspot
* Ex = number of epochs for training in architecture x
* tx = average length of training episode in architecture x
* D = upper bound for deliveries in a circle of radius Ragent during the simulation
* R = upper bound on the number of requests received by the request handler at any step
* Ax = upper bound on the number of deliveries waiting at a hotspot for decision