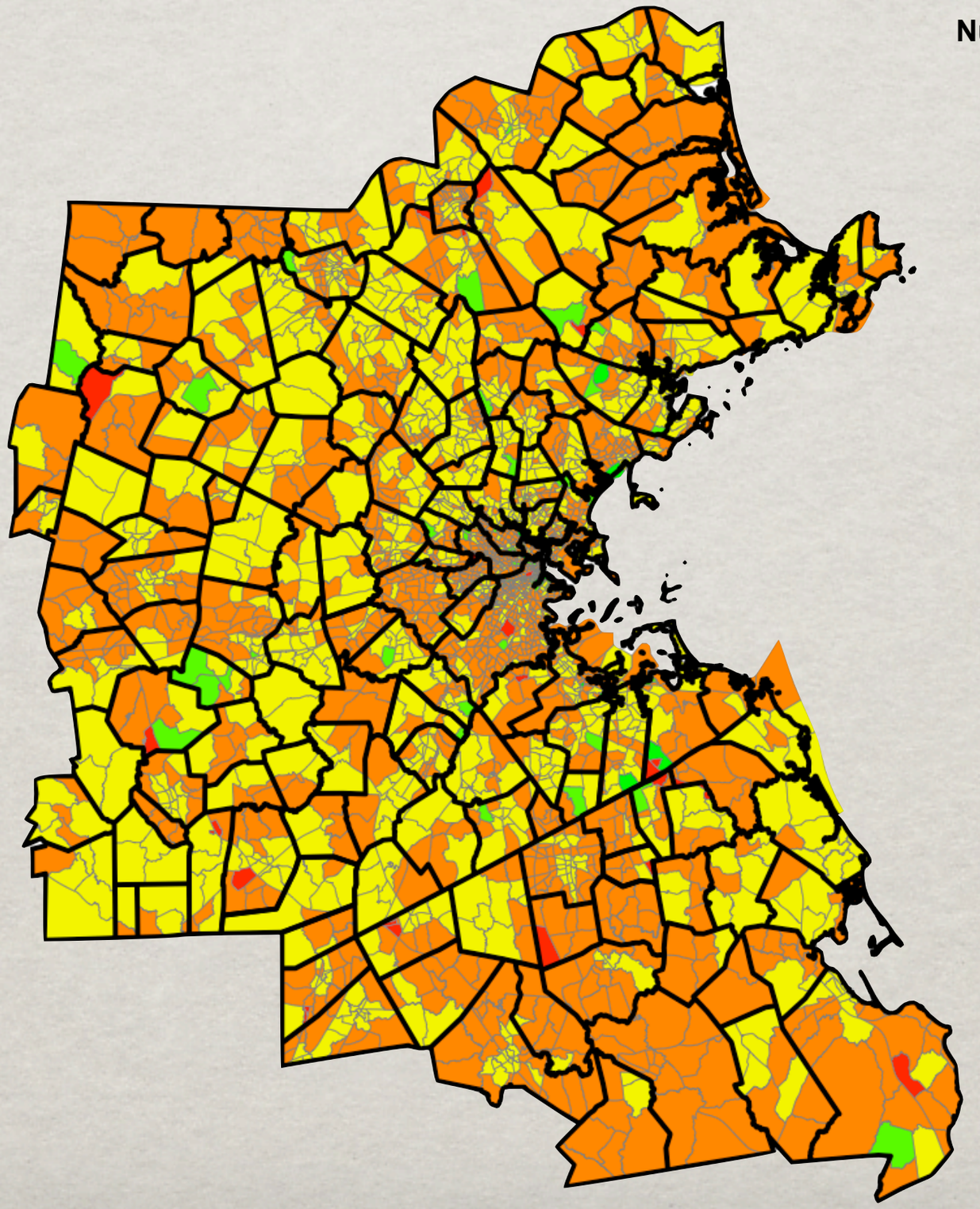
From TAZs to Grids

Paul Green

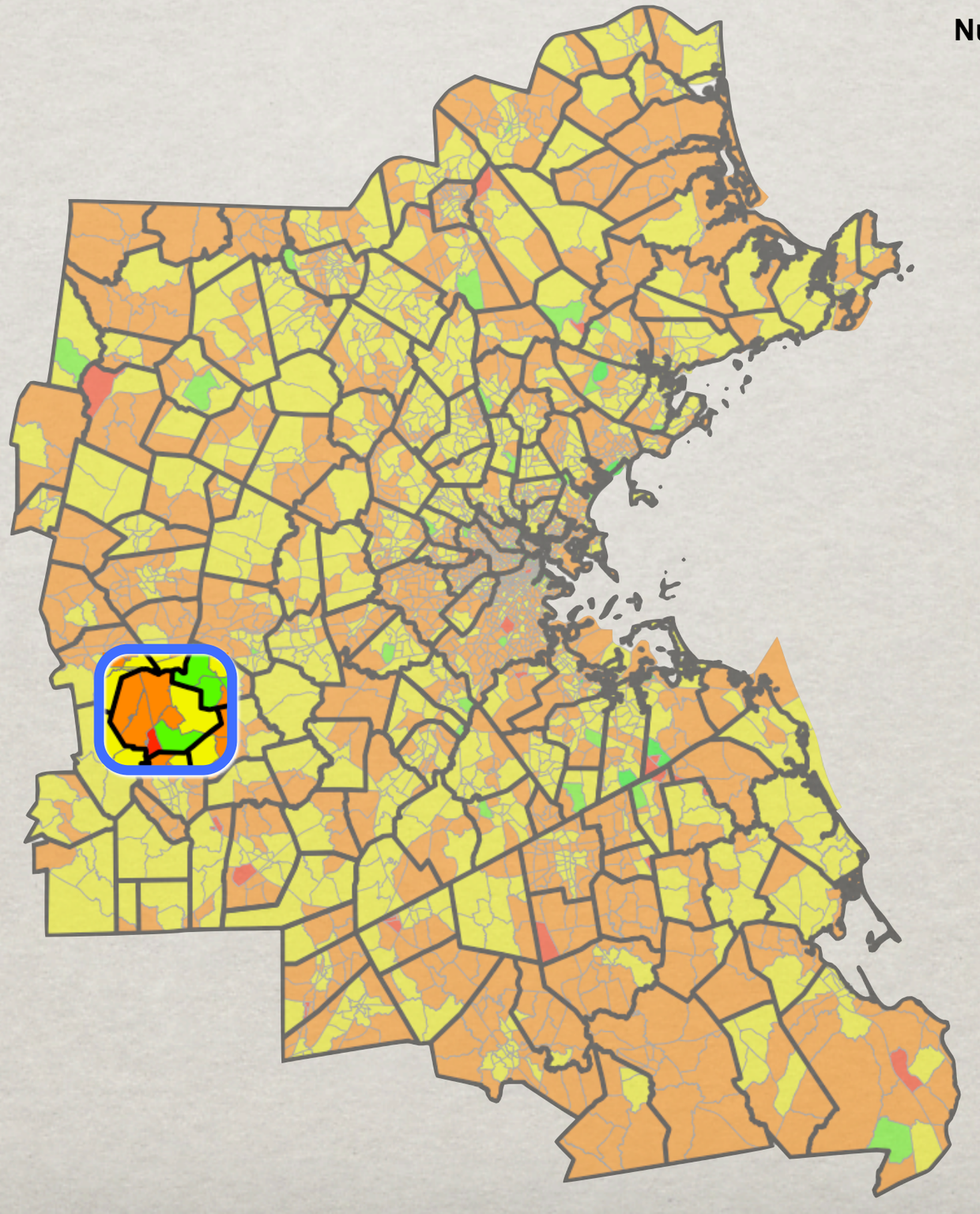
11.521 Final Report

Regional Demand



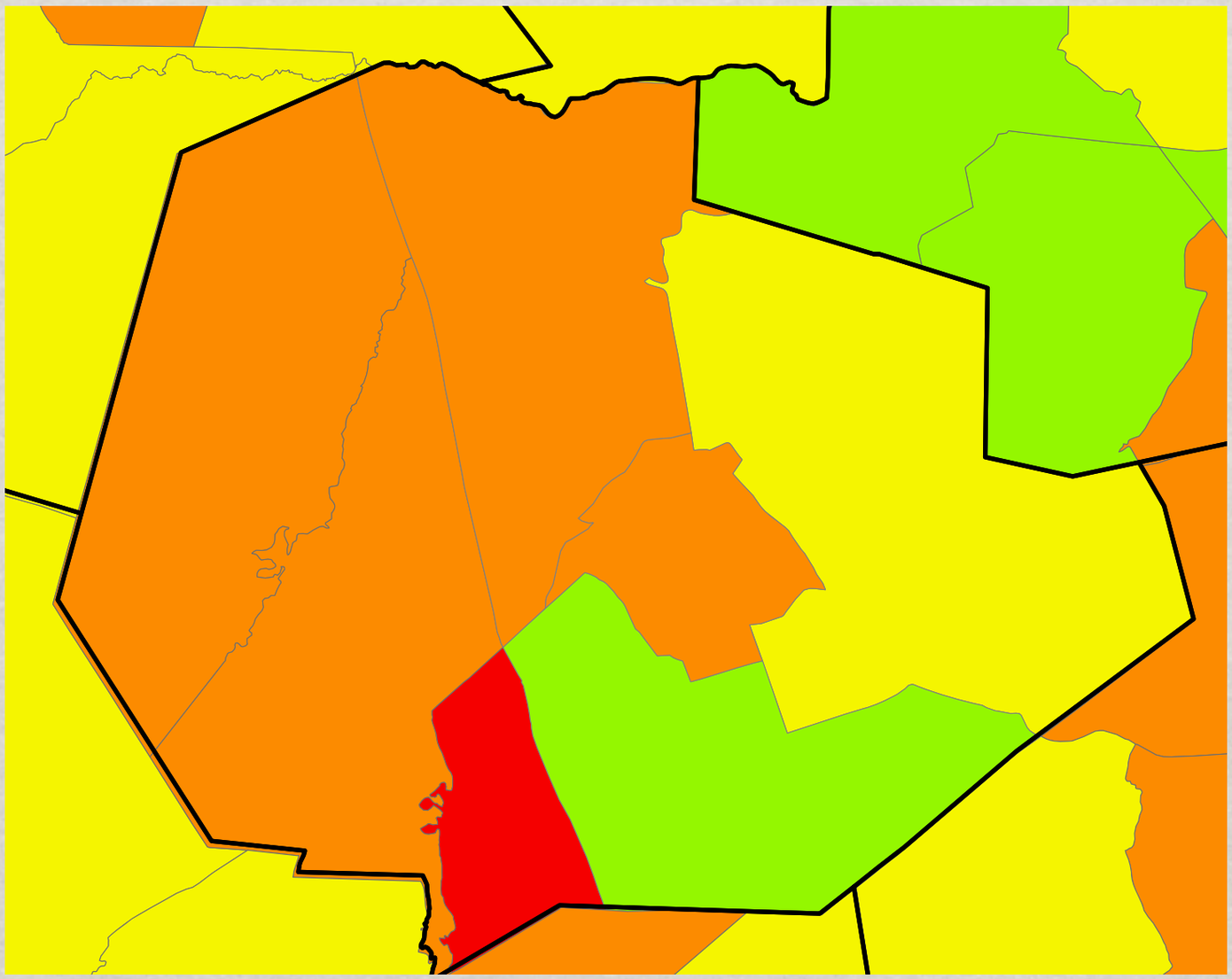
The goal of this presentation is to describe the methods developed by the 11.521 class for taking projections of the metro Boston area new housing unit demand in year 2030 from Traffic Analysis Zones (TAZ's) to a 250 square meter uniform grid. This map shows regional demand for total new units desired (under the Winds of Change scenario) in each TAZ. TAZ boundaries are shaded in light grey, Town boundaries are shown in black. These projections were produced as a part of the MetroFuture project by the Metro Area Planning Council (MAPC).

Regional Demand



Lets take a closer look at one town.

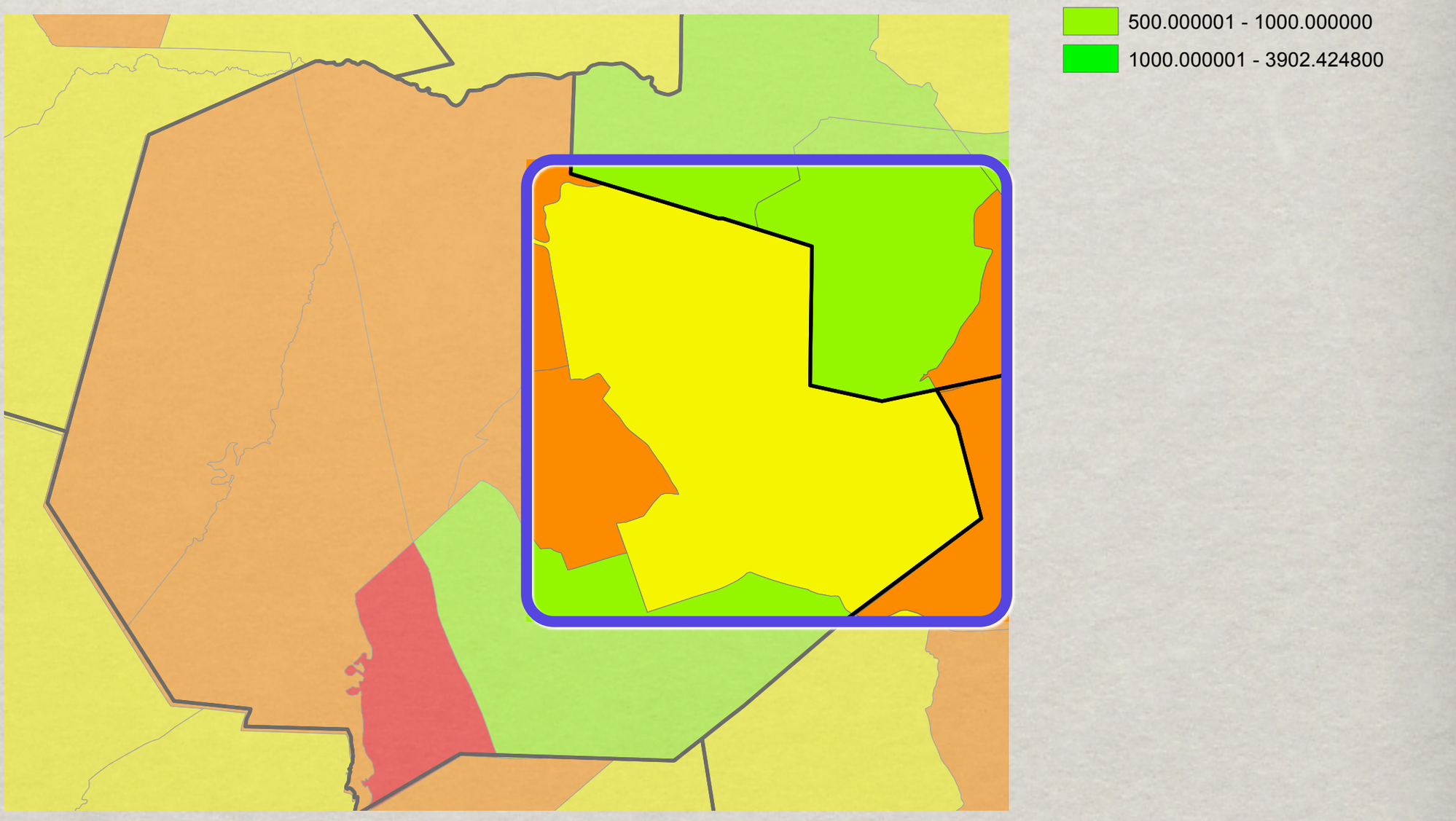
Hopkington



This is a close up of the town of Hopkington, which contains 7 TAZs. I have focused on this town to give some context of the current allocations. Notice, there is one TAZ that is allocated 0 new units (the red TAZ), and directly adjacent is a TAZ with > 500 units (in green). Ideally, an allocation strategy would notice this imbalance, and perhaps shift some of the new units from the green to red TAZ.

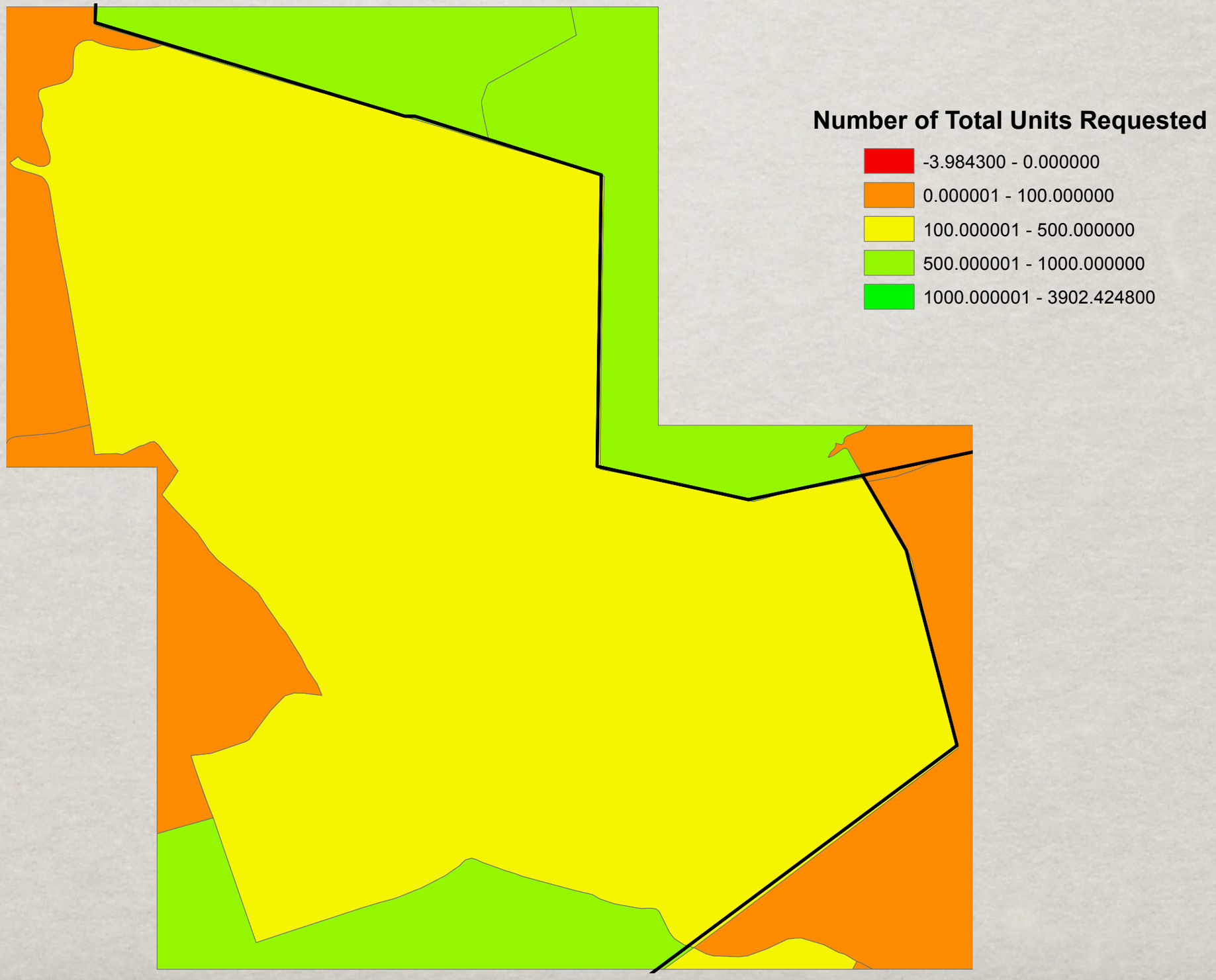
However, in order to minimally affect the MAPC allocation at the TAZ level, and to simplify our allocation algorithm, we consider each TAZ individually.

Hopkington



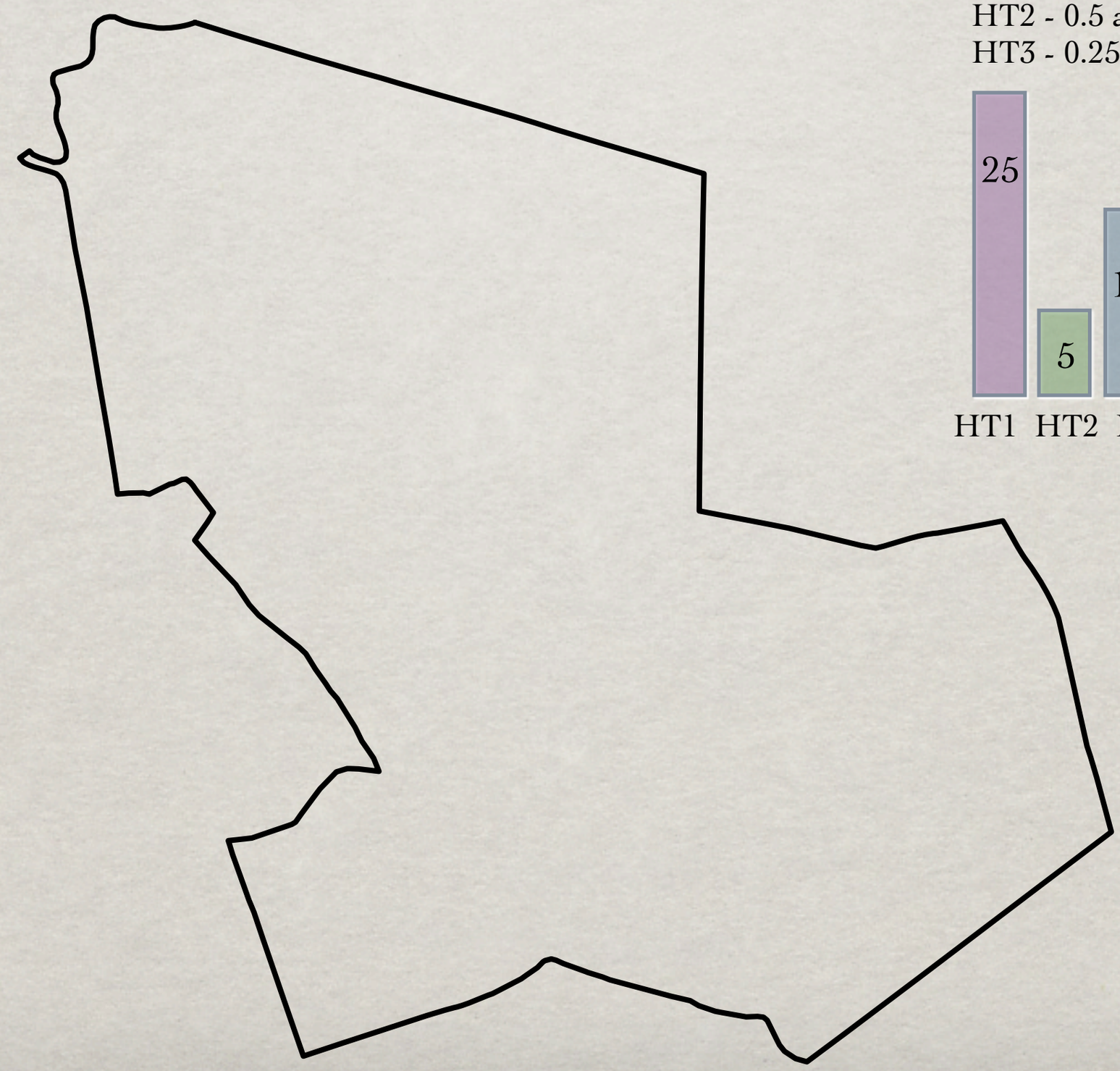
As just mentioned, we consider each TAZ individually when performing our allocation into grids. We will no look at one such TAZ.

TAZ 2301



This is TAZ 2301. It was allocated between 100-500 new units in the Winds of Change (WOC) scenario.

TAZ 2301

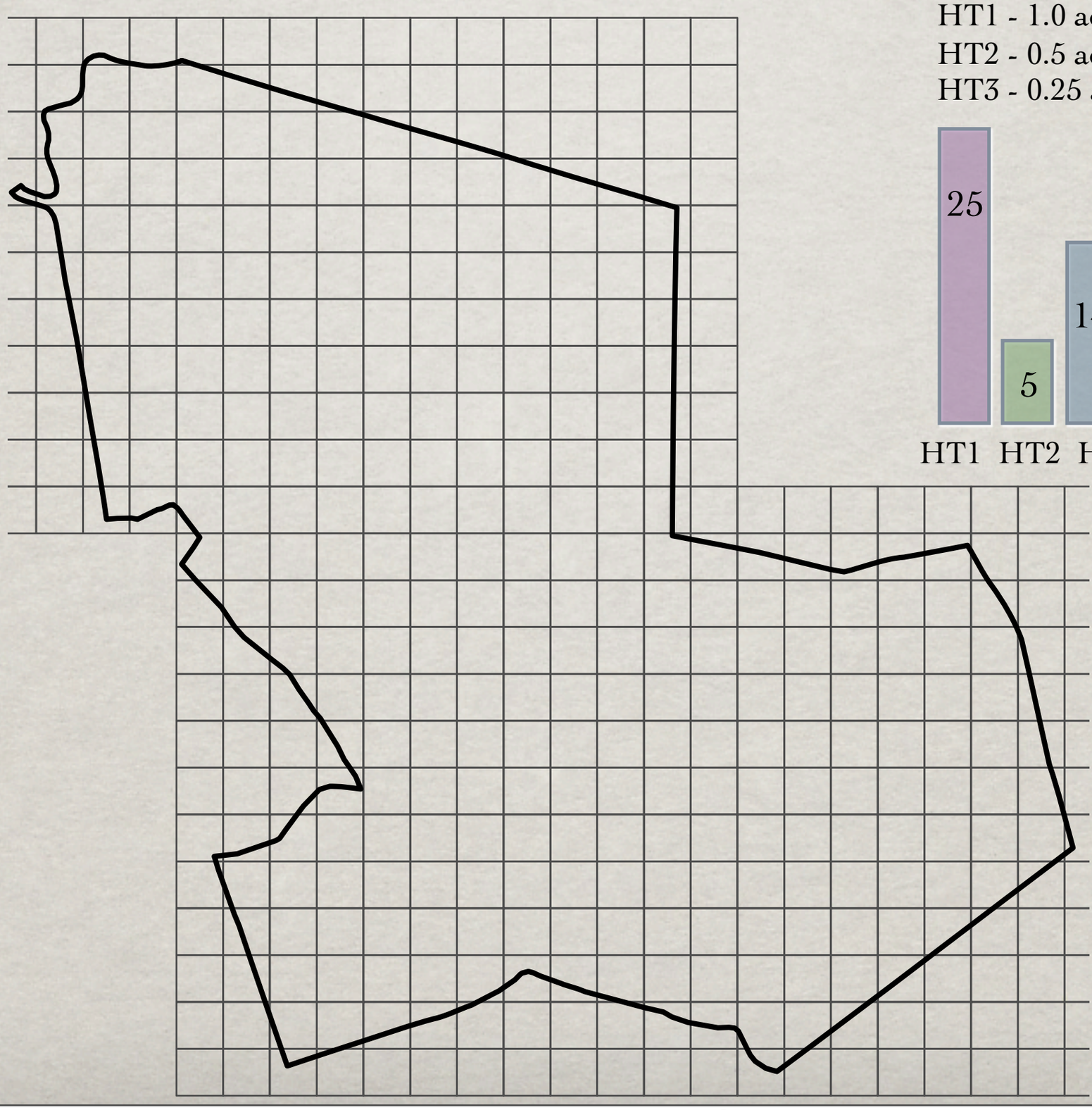


TAZ Demand

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

The total units requested are distributed over 16 different housing types. Each housing type represents a different class of housing in terms of the amount of land required per unit, and the type of land the unit can be built on (for example, single family unit built on 0.5 acres of buildable non wetland, or multi-unit residences using 0.1 acres per unit, on existing commercial land). Each of the 16 housing types can have different numbers of requested new units.

Grids

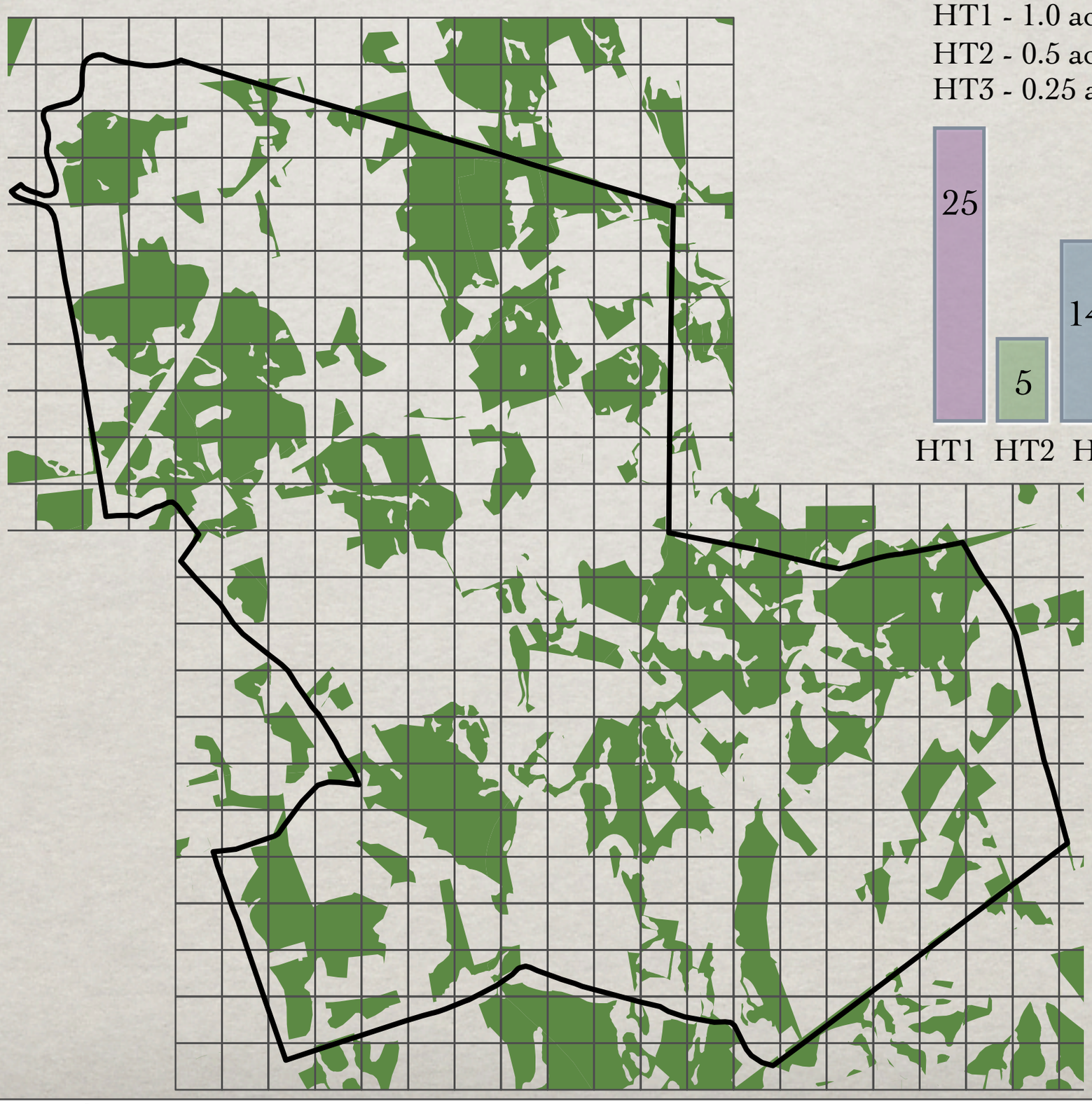


TAZ Demand

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

The goal of our allocation algorithm is to take the numbers of requested new units for the entire TAZ and distribute them to cells in a uniform 250 square meter grid.

Land

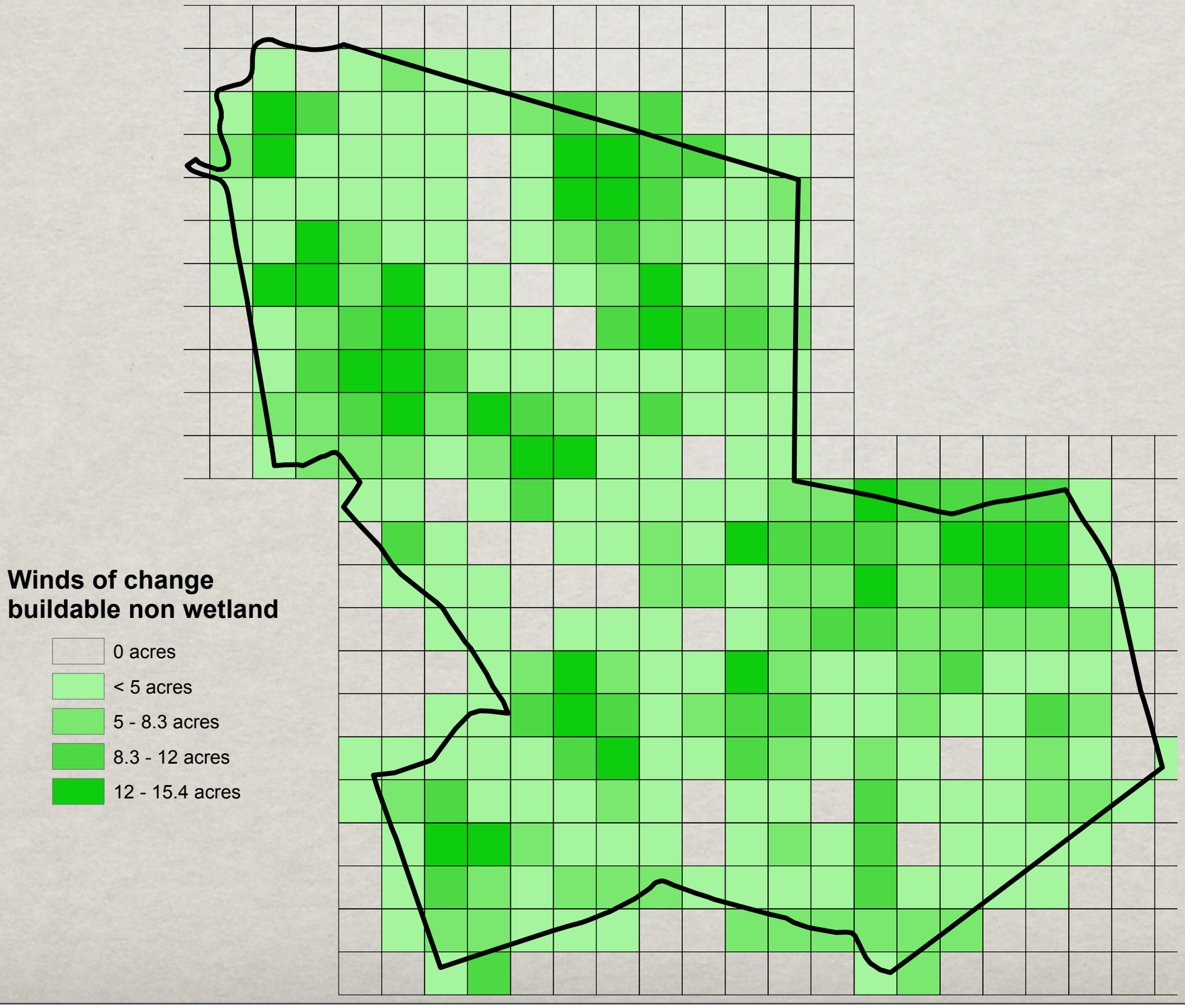


TAZ Demand

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

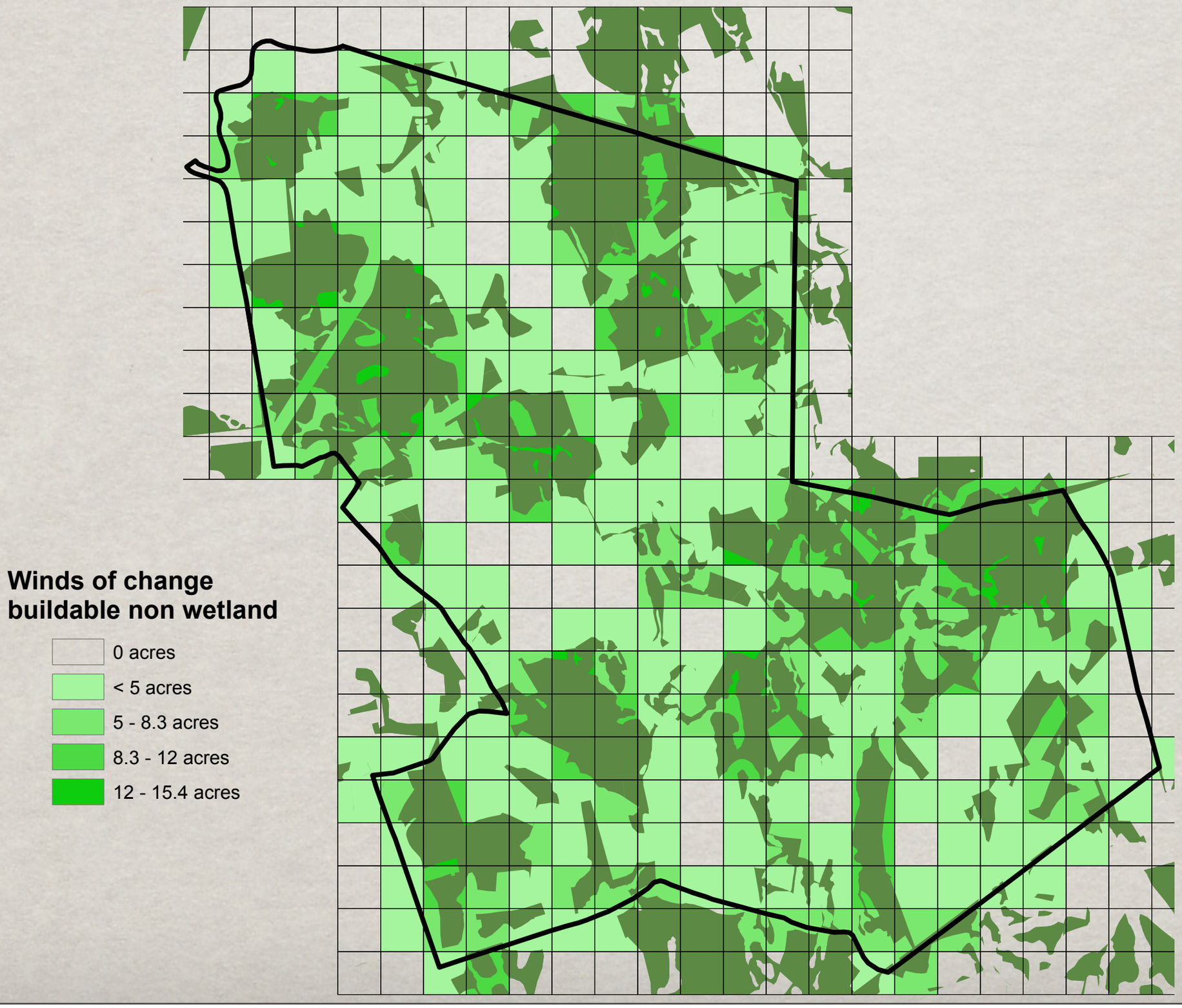
One of the constraints of our allocation algorithm is that the new units be placed into grid cells with enough available land to accommodate them.

Land in Grids



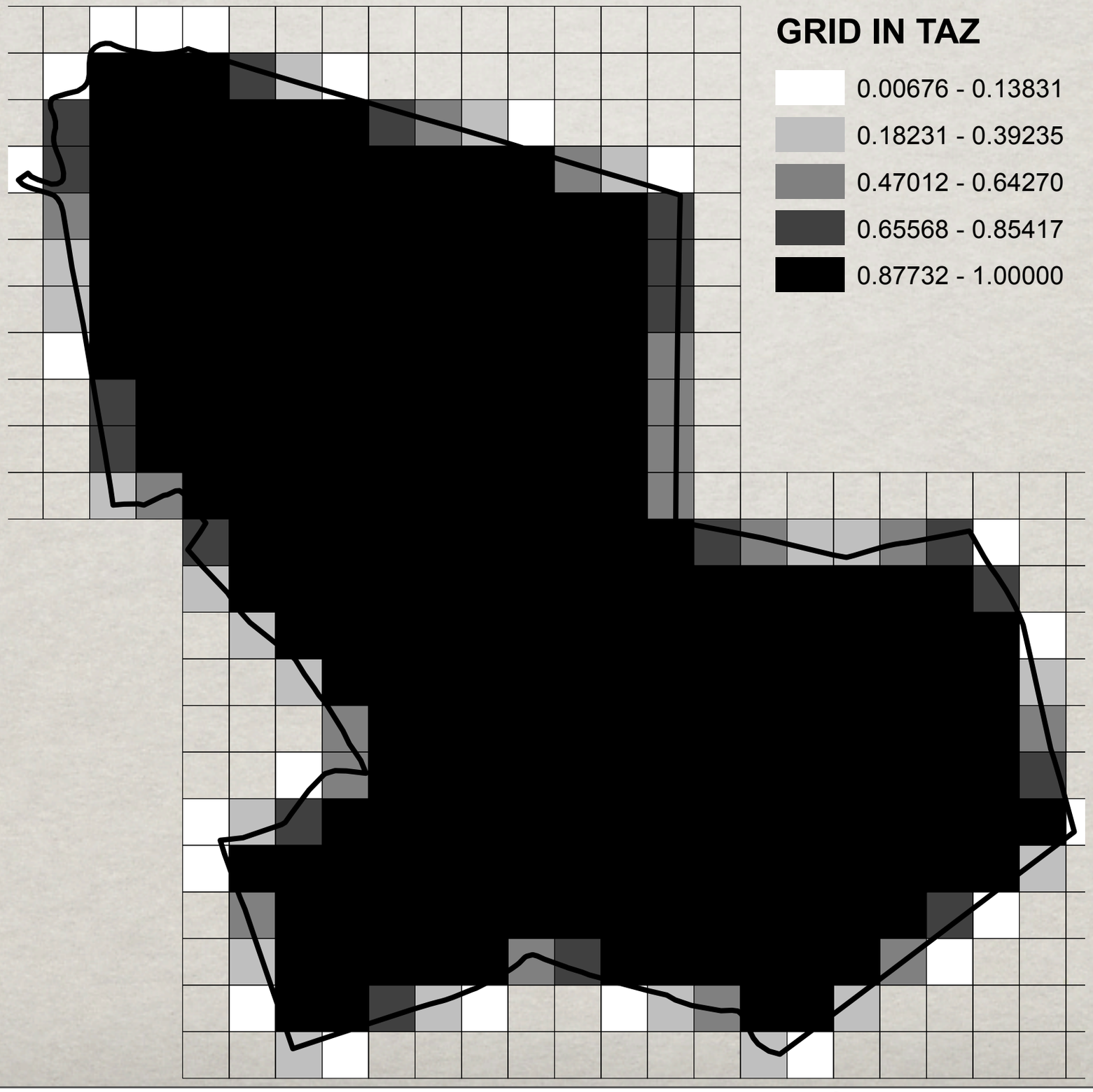
Instead of using the actual shape of the land, we rasterized the land shape files into the grid cells and ignore the exact shape of land plots.

Land in Grids



Here is an overlay of the original land layer and the rasterized grids.

Grid In TAZ



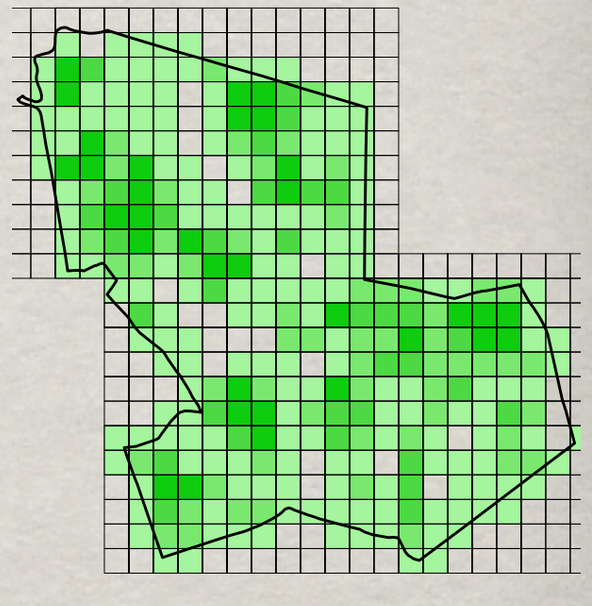
The boundary of a TAZ may intersect grid cells. We make the simplification that the amount of land in a grid cell available to a TAZ is proportional to the percentage of the the grid that is inside the TAZ. The map shows the percentage of each grid inside TAZ 2301.

Weighted Land in Grids

Here is a map of the original Land grid weighted by the percentage of the grid in the TAZ .

Allocation Goals and Requirements

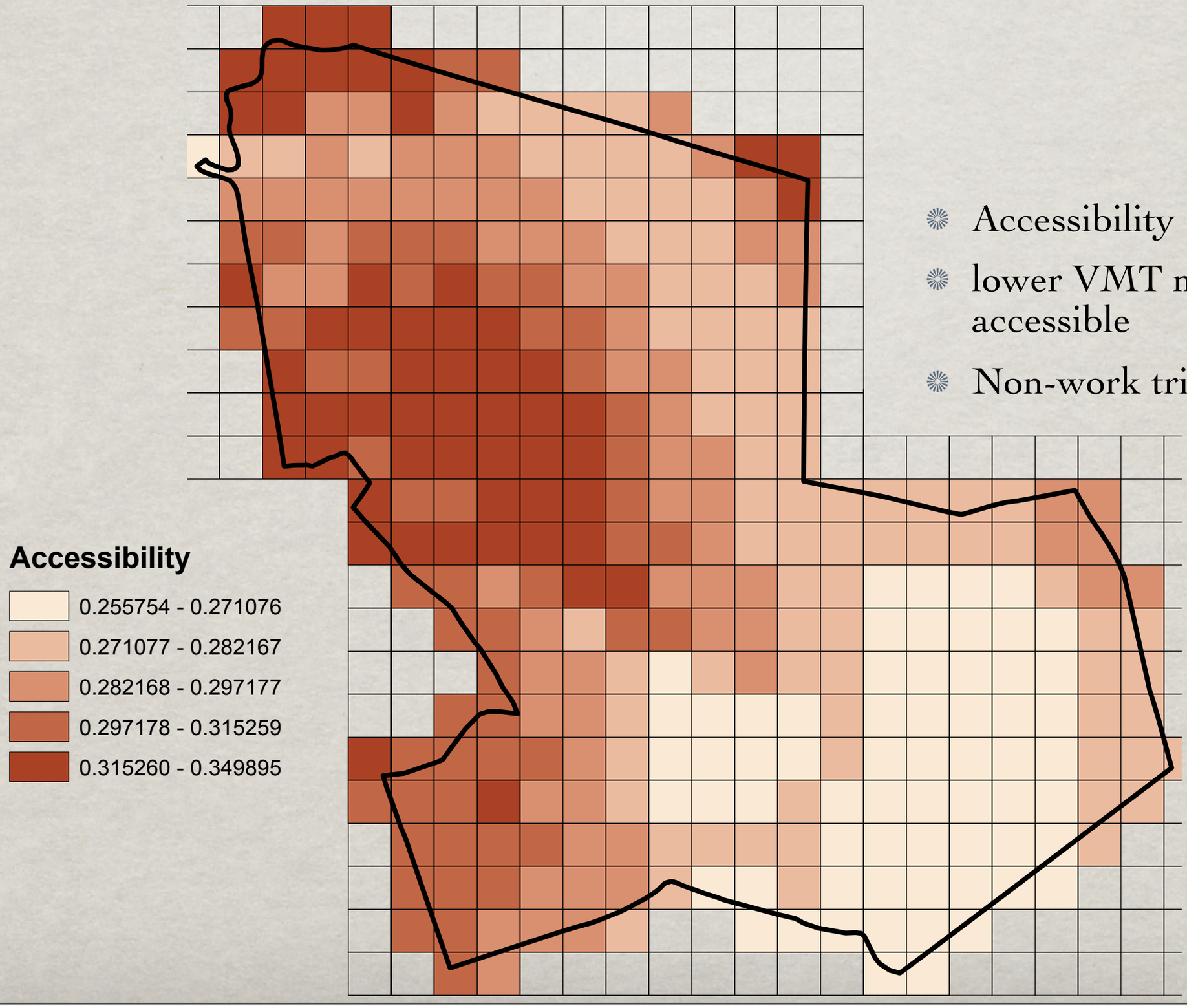
- Allocate all units demanded by MAPC model (or as many as possible)
- Several housing types compete for the same land
- Secondary Goal: Minimize Vehicle Miles Traveled (VMT) per household



The goal of the allocation algorithm is to allocate all of the units requested at the TAZ level to grid cells within the TAZ. A constraint is that multiple housing types compete for the same land. Unfortunately, some TAZs do not contain enough land to accommodate all of the requested new units, thus some decision about which units to allocate must be made.

In addition to producing an allocation that satisfies the land constraints, we have a secondary goal to minimize the average number of Vehicle Miles Traveled per household.

VMT and Accessibility



- Accessibility = 1/ VMT
- lower VMT means more accessible
- Non-work trips only

In order to incorporate VMT into the optimization, we create an “Accessibility” value for each grid that is inversely proportional to the average VMT in the grid (VMT data from MassGIS). Our optimization routine will attempt to place units into grids with high accessibility. We exclude commuting VMT in our accessibility measure, and instead use only non-work VMT.

Greedy Optimization

- Weighted Bin Packing
- NP-Hard problem - Global optimal hard to find
- Instead: Iterative Greedy solution
  - similar to Best Fit First Heuristic
- No constraints on minimum allocation size by housing type (ie. allocating only 1 unit of 50+ apartment is okay)

The optimization problem we have outlined to this point can be cast as the canonical discrete optimization problem known as bin packing. Bin packing is the problem of packing objects of different sizes into a finite number of bins. The bin packing problem is know to be NP-Hard, which in practical terms means that no efficient algorithm for finding a globally optimal solution is known. Therefore, we must give up any hope of finding a global optimum, and instead focus on approximate solutions (so called heuristics).
The Heuristic we adopt is to use an iterative greedy algorithm. It is greedy because at each step of the algorithm we make the choice that locally improves our allocation the most.
We also make a simplification of the problem, by removing any constraints on the minimum allocation sizes.

Algorithm

```
1 while (land still available) AND (more units desired)
2  for each Grid Cell g, and Housing Type h
3     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
4  end for
5  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )
6  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
7  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
8 end while
```

Here we present pseudocode of our greedy algorithm. The algorithm executes iteratively, until all the available land in the TAZ is exhausted or all off the requested units have been allocated (LINE 1)

Algorithm

maximum number of units of housing type h that can fit in grid g

```
1 while (land still available) AND (more units desired)
2  for each Grid Cell g, and Housing Type h
3     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
4  end for
5  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )
6  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
7  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
8 end while
```

At each iteration of the algorithm, a table called “max_units” is constructed (LINES 2-4) that stores for every housing type and grid cell combination, the maximum number of units of housing type h that can fit into grid cell g. max_units(g,h) is computed as the minimum of two terms.

Algorithm

maximum number of units of housing type h that can fit in grid g

# of units demanded for the entire TAZ, by housing type

```
1 while (land still available) AND (more units desired)
2  for each Grid Cell g, and Housing Type h
3     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
4  end for
5  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )
6  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
7  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
8 end while
```

The first term is the total number of units of housing type h requested for the entire TAZ. The maximum number of units of type h put in grid g cannot exceed the total demand for the entire TAZ.

Algorithm

maximum number of units of housing type h that can fit in grid g

# of units demanded for the entire TAZ, by housing type

# of units of type h that can fit in the available land in grid g

```
1 while (land still available) AND (more units desired)
2  for each Grid Cell g, and Housing Type h
3     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
4  end for
5  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )
6  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
7  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
8 end while
```

the second term measures the number of units of type h that can fit in the available land in grid g. For example, if there are 10 acres of land available and housing type h takes 0.5 acres per unit, then at most 20 units (20 = 10 / 0.5) can be placed in grid cell g. The actual maximum number of units is the minimum of this term and the previous term.

Algorithm

maximum number of units of housing type h that can fit in grid g

# of units demanded for the entire TAZ, by housing type

# of units of type h that can fit in the available land in grid g

```
1 while (land still available) AND (more units desired)
2  for each Grid Cell g, and Housing Type h
3     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
4  end for
5  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )
6  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
7  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
8 end while
```

the grid / housing type combination that increases total accessibility the most

Once the max_units table has been constructed, we can use it to find the best grid/housing type combination to allocate at the current step (LINE 5). We scale each entry of max_units by the accessibility for the grid cell g and choose the grid and housing type with the largest value. This choice represents the grid/housing combination that will increase the total accessibility the most in the current iteration.

LINES 6 and 7 update the number of desired units and the available land to reflect the current allocation step.

Allocation Example

```
while (land still available) AND (more units desired)
  for each Grid Cell g, and Housing Type h
     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
  end for

  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )

  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
end while
```

Iteration 1

Available Land

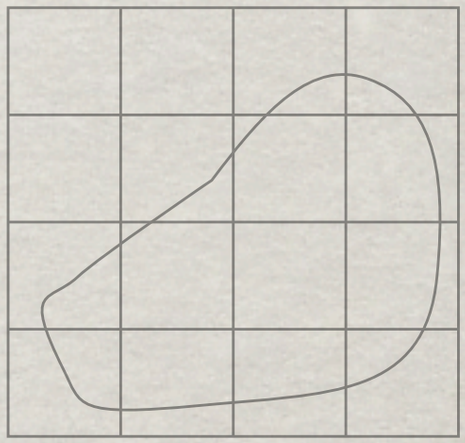
		0	1
	1	3	8
0	5	1	7
3	2	0	1

Accessibility

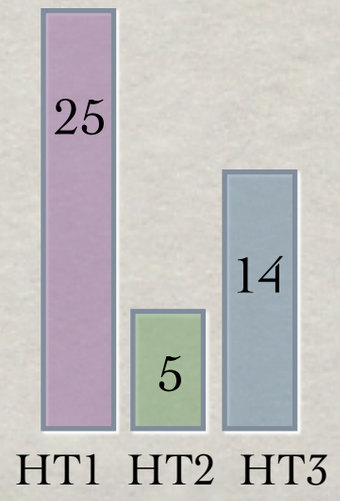
		3	5
	3	4	4
2	1	2	3
3	1	1	1

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

Allocated Units



TAZ Demand



Now, as an example, we will go through a few iterations of the algorithm. On the left we show a hypothetical TAZ superimposed over a 4x4 grid. The top grid shows the available land (in acres) in each cell. The bottom grid shows an example accessibility grid. On the right we show the number of units requested for three hypothetical housing types (w/ the acres per unit show in the bottom left).

Allocation Example

```
while (land still available) AND (more units desired)
  for each Grid Cell g, and Housing Type h
     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
  end for

  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )

  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
end while
```

Iteration 1

Available Land

		0	1
	1	3	8
0	5	1	7
3	2	0	1

max_units(g, HT1)

		0	1
	1	3	8
0	5	1	7
3	2	0	1

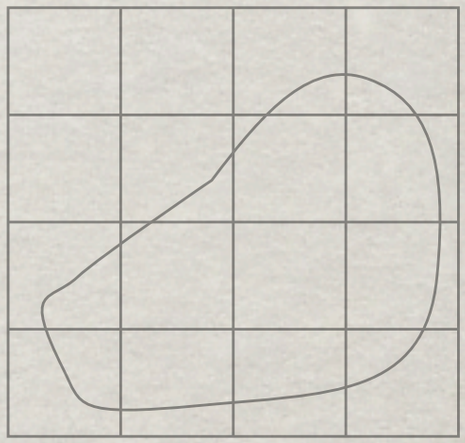
max_units(g, HT2)

		0	2
	2	5	5
0	5	2	5
5	4	0	2

max_units(g, HT3)

		0	4
	4	12	14
0	14	4	14
12	8	0	4

Allocated Units

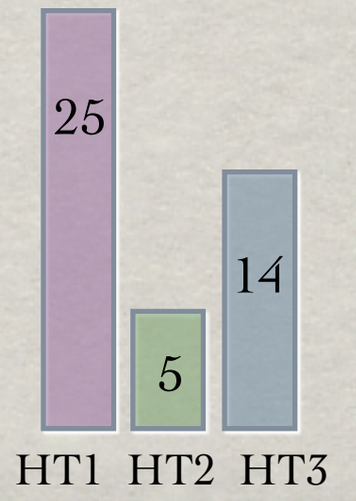


Accessibility

		3	5
	3	4	4
2	1	2	3
3	1	1	1

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

TAZ Demand



The first step is to calculate the max_units table. We show, for each housing type, the value of max_units arranged according to grid cell location. Each cell location shows the maximum number of units of the particular housing type that can fit in the grid cell.

It is instructive to notice that the maximum value in any cell of “max_units(g,HT3)” is 14 because the TAZ demand for HT3 is 14.
And notice that the maximum value in “max_units(g,HT1)” is 8, because the maximum amount of available land in any grid cell is 8 acres (and each unit of HT1 takes 1 acre per unit). This illustrates the two terms of max_units.

Allocation Example

```
while (land still available) AND (more units desired)
  for each Grid Cell g, and Housing Type h
     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
  end for

  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )

  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
end while
```

Iteration 1

Available Land

		0	1
	1	3	8
0	5	1	7
3	2	0	1

Accessibility

		3	5
	3	4	4
2	1	2	3
3	1	1	1

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

max_units(g, HT1)

		0	1
	1	3	8
0	5	1	7
3	2	0	1

max_units(g, HT2)

		0	2
	2	5	5
0	5	2	5
5	4	0	2

max_units(g, HT3)

		0	4
	4	12	14
0	14	4	14
12	8	0	4

max_units(g, HT1) x accessibility

		0	5
	3	12	32
0	5	2	21
9	2	0	1

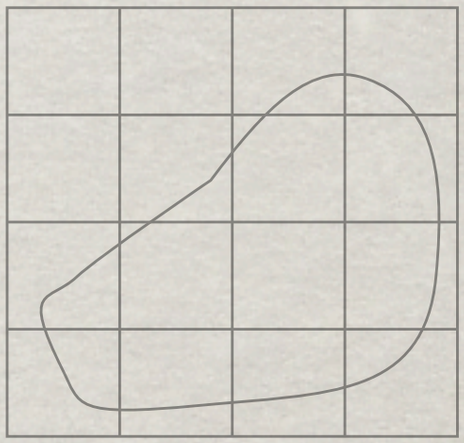
max_units(g, HT2) x accessibility

		0	10
	6	20	20
0	5	4	15
15	4	0	2

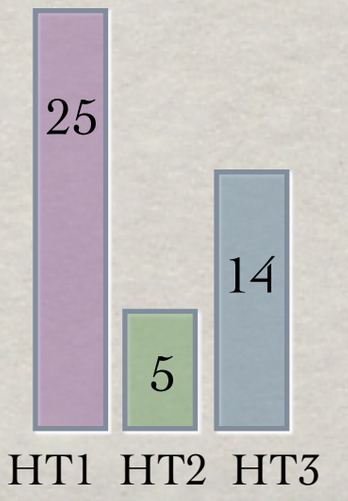
max_units(g, HT3) x accessibility

		0	20
	12	48	56
0	14	8	42
36	8	0	4

Allocated Units



TAZ Demand



Next we weight each entry of max_units by the corresponding grid accessibility.

Allocation Example

```
while (land still available) AND (more units desired)
  for each Grid Cell g, and Housing Type h
     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
  end for

  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )

  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
end while
```

Iteration 1

Available Land

		0	1
	1	3	8
0	5	1	7
3	2	0	1

Accessibility

		3	5
	3	4	4
2	1	2	3
3	1	1	1

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

max_units(g, HT1)

		0	1
	1	3	8
0	5	1	7
3	2	0	1

max_units(g, HT2)

		0	2
	2	5	5
0	5	2	5
5	4	0	2

max_units(g, HT3)

		0	4
	4	12	14
0	14	4	14
12	8	0	4

max_units(g, HT1) x accessibility

		0	5
	3	12	32
0	5	2	21
9	2	0	1

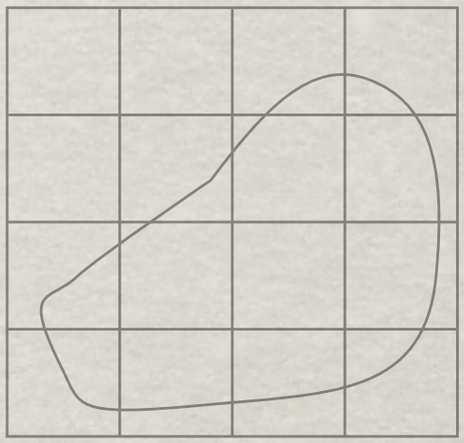
max_units(g, HT2) x accessibility

		0	10
	6	20	20
0	5	4	15
15	4	0	2

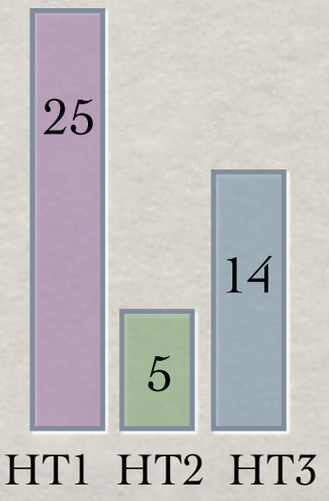
max_units(g, HT3) x accessibility

		0	20
	12	48	56
0	14	8	42
36	8	0	4

Allocated Units



TAZ Demand



Next we select the entry with the highest value (highlighted here in red).

This step is the essence of why this is a greedy algorithm. We choose the entry with the largest value at the current iteration. However, it may be that making this choice now, precludes some other later options that could produce an overall better allocation.

Allocation Example

```
while (land still available) AND (more units desired)
  for each Grid Cell g, and Housing Type h
     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
  end for

  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )

  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
end while
```

Iteration 1

Available Land

		0	1
	1	3	8
0	5	1	7
3	2	0	1

Accessibility

		3	5
	3	4	4
2	1	2	3
3	1	1	1

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

max_units(g, HT1)

		0	1
	1	3	8
0	5	1	7
3	2	0	1

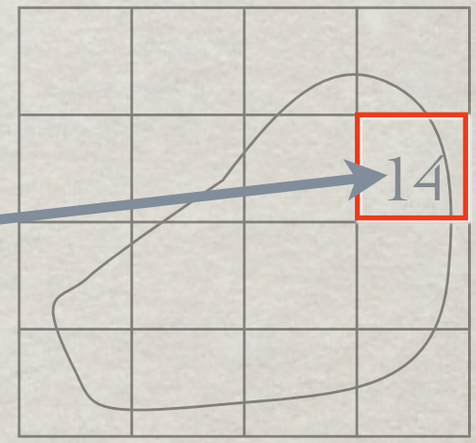
max_units(g, HT2)

		0	2
	2	5	5
0	5	2	5
5	4	0	2

max_units(g, HT3)

		0	4
	4	12	14
0	14	4	14
12	8	0	4

Allocated Units



max_units(g, HT1) x accessibility

		0	5
	3	12	32
0	5	2	21
9	2	0	1

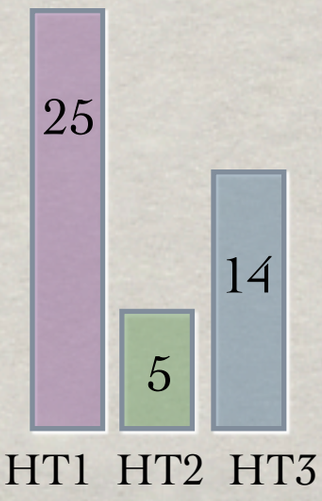
max_units(g, HT2) x accessibility

		0	10
	6	20	20
0	5	4	15
15	4	0	2

max_units(g, HT3) x accessibility

		0	20
	12	48	56
0	14	8	42
36	8	0	4

TAZ Demand



suppose g_best and h_best are the grid location and housing type that maximize our objective. We allocate “max_units(g_best,h_best)” number of units of housing type h_best into grid g_best. In the example above, g_best is highlighted in red, h_best is HT3, and max_units(g_best,h_best) is 14.

Allocation Example

```
while (land still available) AND (more units desired)
  for each Grid Cell g, and Housing Type h
     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
  end for

  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )

  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
end while
```

Iteration 1

Available Land

		0	1
	1	3	4.5
0	5	1	7
3	2	0	1

Accessibility

		3	5
	3	4	4
2	1	2	3
3	1	1	1

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

max_units(g, HT1)

		0	1
	1	3	8
0	5	1	7
3	2	0	1

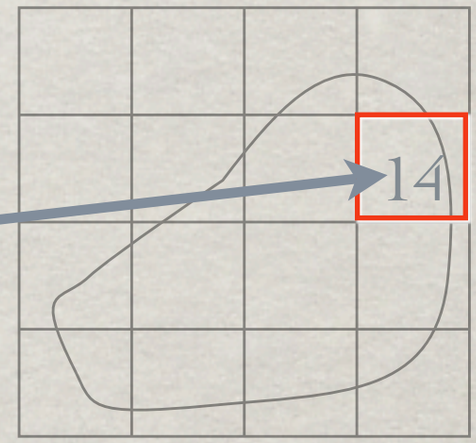
max_units(g, HT2)

		0	2
	2	5	5
0	5	2	5
5	4	0	2

max_units(g, HT3)

		0	4
	4	12	14
0	14	4	14
12	8	0	4

Allocated Units



max_units(g, HT1) x accessibility

		0	5
	3	12	32
0	5	2	21
9	2	0	1

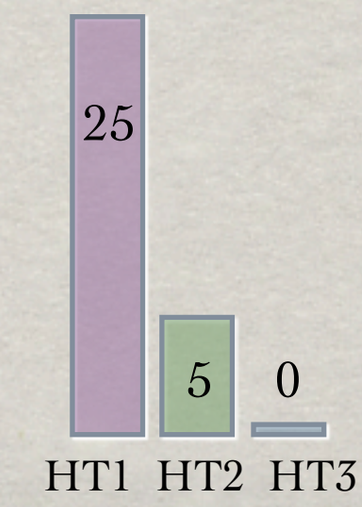
max_units(g, HT2) x accessibility

		0	10
	6	20	20
0	5	4	15
15	4	0	2

max_units(g, HT3) x accessibility

		0	20
	12	48	56
0	14	8	42
36	8	0	4

TAZ Demand



Finally, we update the TAZ demand (ie. subtracting 14 units of type HT3) and the available land for the affected grid cell. In this case 14 units of HT3 requires 3.5 acres of land, so 3.5 acres of land is subtracted from the original 8 acres of land available in the grid cell.

Allocation Example

```
while (land still available) AND (more units desired)
  for each Grid Cell g, and Housing Type h
     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
  end for

  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )

  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
end while
```

Iteration 2

Available Land

		0	1
	1	3	4.5
0	5	1	7
3	2	0	1

Accessibility

		3	5
	3	4	4
2	1	2	3
3	1	1	1

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

max_units(g, HT1)

		0	1
	1	3	4
0	5	1	7
3	2	0	1

max_units(g, HT2)

		0	2
	2	5	5
0	5	2	5
5	4	0	2

max_units(g, HT3)

		0	0
	0	0	0
0	0	0	0
0	0	0	0

Allocated Units

			14

max_units(g, HT1) x accessibility

		0	5
	3	12	16
0	5	2	21
9	2	0	1

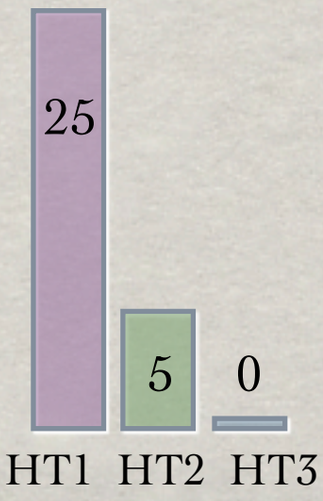
max_units(g, HT2) x accessibility

		0	10
	6	20	20
0	5	4	15
15	4	0	2

max_units(g, HT3) x accessibility

		0	0
	0	0	0
0	0	0	0
0	0	0	0

TAZ Demand



In the next iteration we recalculate the max_units table and the objective. We show the entries that have changed from the previous iterations in light brown.

Allocation Example

```
while (land still available) AND (more units desired)
  for each Grid Cell g, and Housing Type h
     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
  end for

  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )

  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
end while
```

Iteration 2

Available Land

		0	1
	1	3	4.5
0	5	1	7
3	2	0	1

Accessibility

		3	5
	3	4	4
2	1	2	3
3	1	1	1

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

max_units(g, HT1)

		0	1
	1	3	4
0	5	1	7
3	2	0	1

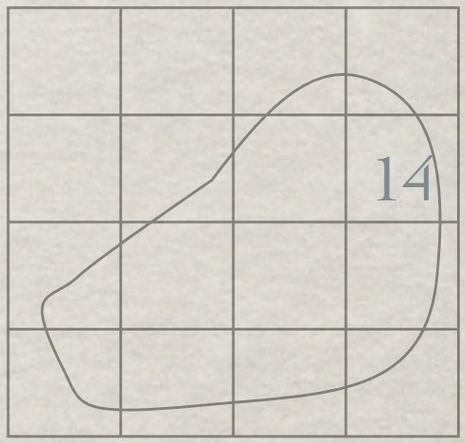
max_units(g, HT2)

		0	2
	2	5	5
0	5	2	5
5	4	0	2

max_units(g, HT3)

		0	0
	0	0	0
0	0	0	0
0	0	0	0

Allocated Units



max_units(g, HT1) x accessibility

		0	5
	3	12	16
0	5	2	21
9	2	0	1

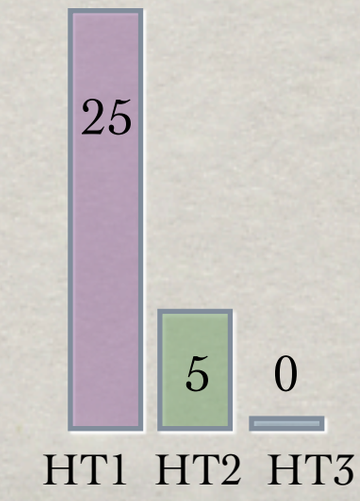
max_units(g, HT2) x accessibility

		0	10
	6	20	20
0	5	4	15
15	4	0	2

max_units(g, HT3) x accessibility

		0	0
	0	0	0
0	0	0	0
0	0	0	0

TAZ Demand



again we find the entry with the maximum value (highlighted in red).

Allocation Example

```
while (land still available) AND (more units desired)
  for each Grid Cell g, and Housing Type h
     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
  end for

  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )

  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
end while
```

Iteration 2

Available Land

		0	1
	1	3	4.5
0	5	1	7
3	2	0	1

Accessibility

		3	5
	3	4	4
2	1	2	3
3	1	1	1

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

max_units(g, HT1)

		0	1
	1	3	4
0	5	1	7
3	2	0	1

max_units(g, HT2)

		0	2
	2	5	5
0	5	2	5
5	4	0	2

max_units(g, HT3)

		0	0
	0	0	0
0	0	0	0
0	0	0	0

Allocated Units

			14
			7

max_units(g, HT1) x accessibility

		0	5
	3	12	16
0	5	2	21
9	2	0	1

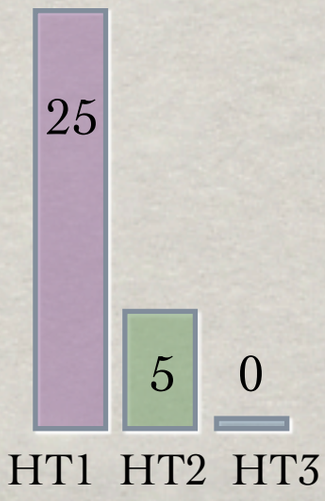
max_units(g, HT2) x accessibility

		0	10
	6	20	20
0	5	4	15
15	4	0	2

max_units(g, HT3) x accessibility

		0	0
	0	0	0
0	0	0	0
0	0	0	0

TAZ Demand



and update the allocated units table.

Allocation Example

```
while (land still available) AND (more units desired)
  for each Grid Cell g, and Housing Type h
     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
  end for

  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )

  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
end while
```

Iteration 2

Available Land

		0	1
	1	3	4.5
0	5	1	0
3	2	0	1

Accessibility

		3	5
	3	4	4
2	1	2	3
3	1	1	1

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

max_units(g, HT1)

		0	1
	1	3	4
0	5	1	7
3	2	0	1

max_units(g, HT2)

		0	2
	2	5	5
0	5	2	5
5	4	0	2

max_units(g, HT3)

		0	0
	0	0	0
0	0	0	0
0	0	0	0

Allocated Units

			14
			7

max_units(g, HT1) x accessibility

		0	5
	3	12	16
0	5	2	21
9	2	0	1

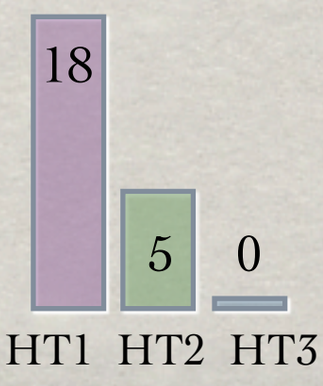
max_units(g, HT2) x accessibility

		0	10
	6	20	20
0	5	4	15
15	4	0	2

max_units(g, HT3) x accessibility

		0	0
	0	0	0
0	0	0	0
0	0	0	0

TAZ Demand



and the available land and TAZ demand.

Allocation Example

```
while (land still available) AND (more units desired)
  for each Grid Cell g, and Housing Type h
     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
  end for

  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )

  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
end while
```

Iteration 3

Available Land

		0	1
	1	3	4.5
0	5	1	0
3	2	0	1

Accessibility

		3	5
	3	4	4
2	1	2	3
3	1	1	1

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

max_units(g, HT1)

		0	1
	1	3	4
0	5	1	0
3	2	0	1

max_units(g, HT2)

		0	2
	2	5	5
0	5	2	0
5	4	0	2

max_units(g, HT3)

		0	0
	0	0	0
0	0	0	0
0	0	0	0

max_units(g, HT1) x accessibility

		0	5
	3	12	16
0	5	2	0
9	2	0	1

max_units(g, HT2) x accessibility

		0	10
	6	20	20
0	5	4	0
15	4	0	2

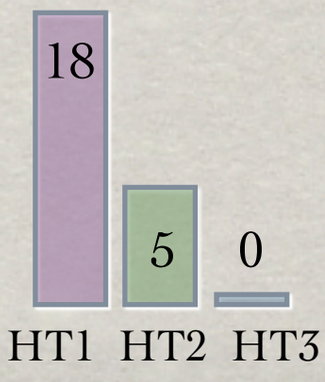
max_units(g, HT3) x accessibility

		0	0
	0	0	0
0	0	0	0
0	0	0	0

Allocated Units

			14
			7

TAZ Demand



In this iteration there are two entries with a maximal value. In this case, we choose one of the two entries at random.

Allocation Example

```
while (land still available) AND (more units desired)
  for each Grid Cell g, and Housing Type h
     max_units(g,h) = MIN( units_desired(h), floor( available_land(g) / acres_per_unit(h) ) )
  end for

  g_best, h_best = ARG MAX ( accessibility(g) x max_units (g,h) )

  units_desired(h_best) = units_desired(h_best) - Max_units(g_best,h_best)
  available_land(g_best) = avail_land(g_hat) - max_units(g_best,h_best) x acres_per_unit(h_best)
end while
```

Iteration 3

Available Land

		0	1
	1	0.5	4.5
0	5	1	0
3	2	0	1

Accessibility

		3	5
	3	4	4
2	1	2	3
3	1	1	1

HT1 - 1.0 acre / unit
HT2 - 0.5 acres / unit
HT3 - 0.25 acres /unit

max_units(g, HT1)

		0	1
	1	3	4
0	5	1	0
3	2	0	1

max_units(g, HT2)

		0	2
	2	5	5
0	5	2	0
5	4	0	2

max_units(g, HT3)

		0	0
	0	0	0
0	0	0	0
0	0	0	0

max_units(g, HT1) x accessibility

		0	5
	3	12	16
0	5	2	0
9	2	0	1

max_units(g, HT2) x accessibility

		0	10
	6	20	20
0	5	4	0
15	4	0	2

max_units(g, HT3) x accessibility

		0	0
	0	0	0
0	0	0	0
0	0	0	0

Allocated Units

		5	14
			7

TAZ Demand

And update all the tables accordingly.

The algorithm will continue for several more iterations while allocating the remaining HT1 units to the available land.

This example shows how the algorithm adapts the order and quantity that housing types are allocated based on the housing demand at the TAZ level, the amount of available land, and the accessibility of each grid.

Stats

	Winds of Change	Let It Be	Let It Be - Random
Total Units Requested	348,505	307,477	307,477
Total Units Allocated	338,873	297,156	297,160
Remainder	9,632 (<2.8%)	10,321 (<3.4%)	10,317 (<3.4%)
VMT per Household	0.9266	1.2559	1.4024

We applied our allocation algorithm to the Winds of Change scenario and two variants of the Let it be scenario – one version used inverse VMT as accessibility, the other used randomly generated accessibility values. We show the number of allocated units and remaining unallocated units for all 3 versions. We also show average VMT per household for the 3 scenarios. The data demonstrate a marked difference between winds of change and let it be scenarios. Additionally, it shows that using VMT as part of the objective function during optimization can further improve VMT numbers.

Appendix A: Code

do_full_alloc.m –

Run the allocation algorithm for all TAZs using the winds of change data.

```
%% run the full allocation
passwd = input('enter password: ','s');
username = 'plg';
conn = database('CRL',username,passwd,'oracle.jdbc.driver.OracleDriver',...
     'jdbc:oracle:thin:@bulfinch.mit.edu:1526:');

%%

Transactions = cell(3,1);
Remainders = zeros(2727,16);
for TAZ = 1:2727,
   %if mod(TAZ,100) == 1
       TAZ
   %end
   [Trans Rems] = greedy_allocate(TAZ, passwd, conn );
   for i=1:3, Transactions{i}(end+1:end+size(Trans{i},1),:) = Trans{i}; end
   Remainders(TAZ,:) = Rems(:);

end

close(conn);

clear TAZ i Trans Rems
%%
save(['new_allocations_' date()]);

%%
alloc_table = sparse(max(([Transactions{1}(:,1); Transactions{2}(:,1); Transactions{3}↙
(:,1)])),16);

for i=1:3
   lt = Transactions{i};
   for j=1:size(lt,1)
      alloc_table(lt(j,1), lt(j,2) )  = alloc_table( lt(j,1), lt(j,2)) + lt(j,3);
   end
   clear lt
end

u = unique([Transactions{1}(:,1); Transactions{2}(:,1); Transactions{3}(:,1)]);
Final_alloc = zeros(size(u,1), 17);
Final_alloc(:,1) = u;
Final_alloc(:,2:17) = full(alloc_table(u,:));

clear alloc_table i j u

%% save back to database

% open the database connection
conn = database('CRL','plg',passwd,'oracle.jdbc.driver.OracleDriver',...
'jdbc:oracle:thin:@bulfinch.mit.edu:1526:');
```

```
%%
curs = exec(conn, ...
        ['drop table my_alloc_full_woc_new CASCADE CONSTRAINTS']);

%% create the table
curs = exec(conn, ...
    ['create table my_alloc_full_woc_new ( g250m_id number(38,0) CONSTRAINT pk_alloc_new↙
PRIMARY KEY, HT_1A number(12,4), HT_2A number(12,4), '...
     'HT_3A number(12,4),HT_4A number(12,4), HT_4B number(12,4), HT_4C number(12,4),↙
HT_5A number(12,4), HT_5B number(12,4), HT_5C number(12,4), '...
     'HT_6A number(12,4), HT_6B number(12,4), HT_6C number(12,4), HT_7A number(12,4),↙
HT_7B number(12,4), HT_7C number(12,4), HT_8C number(12,4) )']);

%% insert the data
colnames = {'g250m_id','HT_1A', 'HT_2A', 'HT_3A','HT_4A', 'HT_4B', 'HT_4C', 'HT_5A',↙
'HT_5B', 'HT_5C', ...
      'HT_6A', 'HT_6B', 'HT_6C', 'HT_7A', 'HT_7B', 'HT_7C', 'HT_8C' };

fastinsert(conn, 'my_alloc_full_woc_new', colnames, Final_alloc);

%% commit
curs = exec(conn,'commit');

%%
curs = exec(conn, ...
        ['drop table my_alloc_remainder_woc_new CASCADE CONSTRAINTS']);

%% create the table
curs = exec(conn, ...
    ['create table my_alloc_remainder_woc_new ( taz number(38,0) CONSTRAINT↙
pk_alloc_rem_woc_new PRIMARY KEY, HT_1A number(12,6), HT_2A number(12,6), '...
     'HT_3A number(12,6),HT_4A number(12,6), HT_4B number(12,6), HT_4C number(12,6),↙
HT_5A number(12,6), HT_5B number(12,6), HT_5C number(12,6), '...
     'HT_6A number(12,6), HT_6B number(12,6), HT_6C number(12,6), HT_7A number(12,6),↙
HT_7B number(12,6), HT_7C number(12,6), HT_8C number(12,6) )']);

%% insert the data
colnames = {'taz','HT_1A', 'HT_2A', 'HT_3A','HT_4A', 'HT_4B', 'HT_4C', 'HT_5A', 'HT_5B',↙
'HT_5C', ...
      'HT_6A', 'HT_6B', 'HT_6C', 'HT_7A', 'HT_7B', 'HT_7C', 'HT_8C' };

fastinsert(conn, 'my_alloc_remainder_woc_new', colnames, [[1:2727]' Remainders]);

%% commit
curs = exec(conn,'commit');

%% close the connection
close(conn);
```

greedy_allocate.m –

Run the allocation algorithm for a single TAZ using the winds of change data.

```
function [Transactions Drem]= greedy_allocate ( TAZ , passwd, conn)
% greedy allocate TAZ

%%
% allocation constraints [LandType, MinUnitAlloc, AreaPerUnit]
% LandType 1:BNW, 2:C/I, 3:RES
Z = [
    1, 1, 1.35
    1, 1, .6
    1, 1, .33
    1, 2, .09
    2, 2, .09
    3, 2, .09
    1, 6, .045
    2, 6, .045
    3, 6, .045
    1,20, .025
    2,20, .025
    3,20, .025
    1,51, .011
    2,51, .011
    3,51, .011
    3, 1,  0];

%% open the database connection
if ~exist('conn', 'var')
  conn = database('CRL','plg',passwd,'oracle.jdbc.driver.OracleDriver',...
     'jdbc:oracle:thin:@bulfinch.mit.edu:1526:');
    close_conn = 1;
else
    close_conn = 0;
end

%%
% need D(1:16) = #units demanded for each of the 16 housing types
curs = exec(conn, ...
    ['select newunits from tazreslong where taz = ' num2str(TAZ) ' order by devtype']);
curs=fetch(curs);
D = cell2mat(curs.data);
close(curs);


%%
% need Y(1:N,1:4) = amount of available resources (BNW_LIB,BNW_WOC,C/I,RES) in each grid
% cell [G250m_id, BNW_LIB,BNW_WOC,C/I,RES]
curs = exec(conn, ...
['select t.g250m_id, g.bnw_lib, g.bnw_woc, g.comin, g.resi, max(t.grid_in_taz) ' ...
 'from tazgrid t, gridlanduse g where taz = ' num2str(TAZ) ' and t.g250m_id = g.g250m_id↙
'...
 'group by t.g250m_id, g.bnw_lib, g.bnw_woc, g.comin, g.resi order by t.g250m_id'] );
%    ['select g.*, t.grid_in_taz from gridlanduse g, tazgrid t where t.taz
```

```
%    = ' num2str(TAZ) ' and t.g250m_id = g.g250m_id'] ...
curs=fetch(curs);
Y = cell2mat(curs.data);
close(curs);

% convert sq meters to acres
Y(:,2:5) = Y(:,2:5) / 4046.85642;

%% get the accessibility measure for each grid cell (VMT)
% [G250m_id, access_dist]
curs = exec(conn, ...
    ['select g.g250m_id, g.eudist from gridvmt g, tazgrid t where t.taz = ' num2str(TAZ)↙
' and t.g250m_id = g.g250m_id order by t.g250m_id']);
curs=fetch(curs);
V = cell2mat(curs.data);
close(curs);


%% close the database connection
if close_conn
    close(conn);
end

%% rank the gridcells by vmt/household
%[R,I] = sort(V(:,3)./V(:,4));
%accessibility = @(x) exp(-(x.^2)./100^2);
accessibility = @(x) (1./x);

%% for each gridcell, allocate the housing type that increases vmt the most

% current desired amount of allocated houses
Dcurr = D;
Dcurr(Dcurr<0) = 0; % only allocate integer number of units
Dcurr = round(Dcurr);
Drem = D - Dcurr;
% weight by grid_in_taz
Ycurr = Y(:,1:5);
Ycurr(:,2:5) = Ycurr(:,2:5) .* repmat(Y(:,6),[1 4]);
Transactions = cell(3,1);

%Dcurr(Dcurr<Z(:,2)) = 0; % only allocate units over minimum
Access = accessibility(V(:,2));
%%

for HOUSING_TYPE = 1:3

    HT = find(Z(:,1)==HOUSING_TYPE);
    while 1
        % find the best grid cell / housing type to allocate
        max_housing = floor(Ycurr(:,2+HOUSING_TYPE) * (1./Z(HT,3))');
        max_housing(isnan(max_housing)) = inf;
```

```
        demand = repmat(Dcurr(HT)',[size(max_housing,1) 1]);
        max_housing(demand < max_housing) = demand(demand < max_housing);

        if sum(Dcurr(HT)) == 0 || sum(max_housing(:)) == 0
            break;
        end

        [best bestI] = max(reshape(max_housing.* repmat(Access,[1 length(HT)]),[],1));

        [i j] = ind2sub(size(max_housing),bestI(1));

        % keep track of the book keeping
        Transactions{HOUSING_TYPE}(end+1,:) = [Ycurr(i,1) HT(j) max_housing(i,j)];
        Dcurr(HT(j)) = Dcurr(HT(j)) - max_housing(i,j);
        Ycurr(i,2+HOUSING_TYPE) = Ycurr(i,2+HOUSING_TYPE) - max_housing(i,j)*Z(HT(j),3);

    end
end

%%
Drem = (Drem + Dcurr)';
```

do_full_alloc_lib.m –

Run the allocation algorithm for all TAZs using the let it be data.

```
%% run the full allocation
passwd = input('enter password: ','s');

conn = database('CRL','plg',passwd,'oracle.jdbc.driver.OracleDriver',...
     'jdbc:oracle:thin:@bulfinch.mit.edu:1526:');

%%
Transactions = cell(3,1);
Remainders = zeros(2727,16);
for TAZ = 1:2727,
   %if mod(TAZ,100) == 1
       TAZ
   %end
   [Trans Rems] = greedy_allocate_lib(TAZ, passwd, conn );
   for i=1:3, Transactions{i}(end+1:end+size(Trans{i},1),:) = Trans{i}; end
   Remainders(TAZ,:) = Rems(:);

end

close(conn);

clear TAZ i Trans Rems
%%
save(['allocations_lib_' date()]);

%%
alloc_table = sparse(max(([Transactions{1}(:,1); Transactions{2}(:,1); Transactions{3}↙
(:,1)])),16);

for i=1:3
   lt = Transactions{i};
   for j=1:size(lt,1)
      alloc_table(lt(j,1), lt(j,2) )  = alloc_table( lt(j,1), lt(j,2)) + lt(j,3);
   end
   clear lt
end

u = unique([Transactions{1}(:,1); Transactions{2}(:,1); Transactions{3}(:,1)]);
Final_alloc = zeros(size(u,1), 17);
Final_alloc(:,1) = u;
Final_alloc(:,2:17) = full(alloc_table(u,:));

clear alloc_table i j u

%% save back to database

% open the database connection
conn = database('CRL','plg',passwd,'oracle.jdbc.driver.OracleDriver',...
'jdbc:oracle:thin:@bulfinch.mit.edu:1526:');

%%
```

```
curs = exec(conn, ...
        ['drop table my_alloc_full_lib_new CASCADE CONSTRAINTS']);

%% create the table
curs = exec(conn, ...
    ['create table my_alloc_full_lib_new ( g250m_id number(38,0) CONSTRAINT↙
pk_alloc_lib_new PRIMARY KEY, HT_1A number(12,4), HT_2A number(12,4), '...
     'HT_3A number(12,4),HT_4A number(12,4), HT_4B number(12,4), HT_4C number(12,4),↙
HT_5A number(12,4), HT_5B number(12,4), HT_5C number(12,4), '...
     'HT_6A number(12,4), HT_6B number(12,4), HT_6C number(12,4), HT_7A number(12,4),↙
HT_7B number(12,4), HT_7C number(12,4), HT_8C number(12,4) )']);

%% insert the data
colnames = {'g250m_id','HT_1A', 'HT_2A', 'HT_3A','HT_4A', 'HT_4B', 'HT_4C', 'HT_5A',↙
'HT_5B', 'HT_5C', ...
      'HT_6A', 'HT_6B', 'HT_6C', 'HT_7A', 'HT_7B', 'HT_7C', 'HT_8C' };

fastinsert(conn, 'my_alloc_full_lib_new', colnames, Final_alloc);

%% commit
curs = exec(conn,'commit');


%%
curs = exec(conn, ...
        ['drop table my_alloc_remainder_lib_new CASCADE CONSTRAINTS']);

%% create the table
curs = exec(conn, ...
    ['create table my_alloc_remainder_lib_new ( taz number(38,0) CONSTRAINT↙
pk_alloc_rem_lib_new PRIMARY KEY, HT_1A number(12,6), HT_2A number(12,6), '...
     'HT_3A number(12,6),HT_4A number(12,6), HT_4B number(12,6), HT_4C number(12,6),↙
HT_5A number(12,6), HT_5B number(12,6), HT_5C number(12,6), '...
     'HT_6A number(12,6), HT_6B number(12,6), HT_6C number(12,6), HT_7A number(12,6),↙
HT_7B number(12,6), HT_7C number(12,6), HT_8C number(12,6) )']);

%% insert the data
colnames = {'taz','HT_1A', 'HT_2A', 'HT_3A','HT_4A', 'HT_4B', 'HT_4C', 'HT_5A', 'HT_5B',↙
'HT_5C', ...
      'HT_6A', 'HT_6B', 'HT_6C', 'HT_7A', 'HT_7B', 'HT_7C', 'HT_8C' };

fastinsert(conn, 'my_alloc_remainder_lib_new', colnames, [[1:2727]' Remainders]);

%% commit
curs = exec(conn,'commit');

%% close the connection
close(conn);
```

greedy_allocate_lib.m –

Run the allocation algorithm for a single TAZ using the let it be data.

```
function [Transactions Drem]= greedy_allocate_lib ( TAZ , passwd, conn)
% greedy allocate TAZ

%%
% allocation constraints [LandType, MinUnitAlloc, AreaPerUnit]
% LandType 1:BNW, 2:C/I, 3:RES
Z = [
    1, 1, 1.35
    1, 1, .6
    1, 1, .33
    1, 2, .09
    2, 2, .09
    3, 2, .09
    1, 6, .045
    2, 6, .045
    3, 6, .045
    1,20, .025
    2,20, .025
    3,20, .025
    1,51, .011
    2,51, .011
    3,51, .011
    3, 1,  0];

%% open the database connection
if ~exist('conn', 'var')
  conn = database('CRL','plg',passwd,'oracle.jdbc.driver.OracleDriver',...
     'jdbc:oracle:thin:@bulfinch.mit.edu:1526:');
    close_conn = 1;
else
    close_conn = 0;
end


%%
% need D(1:16) = #units demanded for each of the 16 housing types
curs = exec(conn, ...
    ['select newunits from tazresliblong where taz = ' num2str(TAZ) ' order by↙
devtype']);
curs=fetch(curs);
D = cell2mat(curs.data);
close(curs);


%%
% need Y(1:N,1:4) = amount of available resources (BNW_LIB,BNW_WOC,C/I,RES) in each grid
% cell
% Y = [G250m_id, BNW_LIB, BNW_WOC, C/I, RES, GRID_IN_TAZ]
curs = exec(conn, ...
['select t.g250m_id, g.bnw_lib, g.bnw_woc, g.comin, g.resi, max(t.grid_in_taz) ' ...
 'from tazgrid t, gridlanduse g where taz = ' num2str(TAZ) ' and t.g250m_id = g.g250m_id↙
```

```
'...
 'group by t.g250m_id, g.bnw_lib, g.bnw_woc, g.comin, g.resi order by t.g250m_id'] );
%    ['select g.*, t.grid_in_taz from gridlanduse g, tazgrid t where t.taz
%    = ' num2str(TAZ) ' and t.g250m_id = g.g250m_id'] ...
curs=fetch(curs);
Y = cell2mat(curs.data);
close(curs);

% convert sq meters to acres
Y(:,2:5) = Y(:,2:5) / 4046.85642;

%% get the accessibility measure for each grid cell (VMT)
% [G250m_id, access_dist]
curs = exec(conn, ...
    ['select g.g250m_id, g.eudist from gridvmt g, tazgrid t where t.taz = ' num2str(TAZ)↙
' and t.g250m_id = g.g250m_id order by t.g250m_id']);
curs=fetch(curs);
V = cell2mat(curs.data);
close(curs);


%% close the database connection
if close_conn
    close(conn);
end

%% rank the gridcells by vmt/household
%[R,I] = sort(V(:,3)./V(:,4));
%accessibility = @(x) exp(-(x.^2)./100^2);
accessibility = @(x) (1./x);

%% for each gridcell, allocate the housing type that increases vmt the most

% current desired amount of allocated houses
Dcurr = D;
Dcurr(Dcurr<0) = 0; % only allocate integer number of units
Dcurr = round(Dcurr);
Drem = D - Dcurr;
% weight by grid_in_taz
Ycurr = Y(:,1:5);
Ycurr(:,2:5) = Ycurr(:,2:5) .* repmat(Y(:,6),[1 4]);
Transactions = cell(3,1);

%Dcurr(Dcurr<Z(:,2)) = 0; % only allocate units over minimum
Access = accessibility(V(:,2));
%%
H_TABLE = [2 4 5]; %[BWN_LIB C/I RES]

for HOUSING_TYPE = 1:3

    HT = find(Z(:,1)==HOUSING_TYPE);
```

```
    while 1
        % find the best grid cell / housing type to allocate
        max_housing = floor(Ycurr(:,H_TABLE(HOUSING_TYPE)) * (1./Z(HT,3))');
        max_housing(isnan(max_housing)) = inf;
        demand = repmat(Dcurr(HT)',[size(max_housing,1) 1]);
        max_housing(demand < max_housing) = demand(demand < max_housing);

        if sum(Dcurr(HT)) == 0 || sum(max_housing(:)) == 0
            break;
        end

        [best bestI] = max(reshape(max_housing.* repmat(Access,[1 length(HT)]),[],1));

        [i j] = ind2sub(size(max_housing),bestI(1));

        % keep track of the book keeping
        Transactions{HOUSING_TYPE}(end+1,:) = [Ycurr(i,1) HT(j) max_housing(i,j)];
        Dcurr(HT(j)) = Dcurr(HT(j)) - max_housing(i,j);
        Ycurr(i, H_TABLE(HOUSING_TYPE)) = Ycurr(i,H_TABLE(HOUSING_TYPE)) - max_housing↙
(i,j)*Z(HT(j),3);

    end
end

%%
Drem = (Drem + Dcurr)';
```

do_full_alloc_lib_random.m –

Run the allocation algorithm for all TAZs using the let it be data and random accessiblity.

```
%% run the full allocation
passwd = input('enter password: ','s');

conn = database('CRL','plg',passwd,'oracle.jdbc.driver.OracleDriver',...
     'jdbc:oracle:thin:@bulfinch.mit.edu:1526:');

%%
Transactions = cell(3,1);
Remainders = zeros(2727,16);
for TAZ = 1:2727,
   %if mod(TAZ,100) == 1
       TAZ
   %end
   [Trans Rems] = greedy_allocate_lib_random(TAZ, passwd , conn);
   for i=1:3, Transactions{i}(end+1:end+size(Trans{i},1),:) = Trans{i}; end
   Remainders(TAZ,:) = Rems(:);

end

close(conn);


clear TAZ i Trans Rems
%%
save(['allocations_lib_rand_new_' date()]);

%%
alloc_table = sparse(max(([Transactions{1}(:,1); Transactions{2}(:,1); Transactions{3}↙
(:,1)])),16);

for i=1:3
   lt = Transactions{i};
   for j=1:size(lt,1)
      alloc_table(lt(j,1), lt(j,2) )  = alloc_table( lt(j,1), lt(j,2)) + lt(j,3);
   end
   clear lt
end

u = unique([Transactions{1}(:,1); Transactions{2}(:,1); Transactions{3}(:,1)]);
Final_alloc = zeros(size(u,1), 17);
Final_alloc(:,1) = u;
Final_alloc(:,2:17) = full(alloc_table(u,:));

clear alloc_table i j u

%% save back to database

% open the database connection
conn = database('CRL','plg',passwd,'oracle.jdbc.driver.OracleDriver',...
'jdbc:oracle:thin:@bulfinch.mit.edu:1526:');
```

```
%%
curs = exec(conn, ...
        ['drop table my_alloc_full_lib_rand_new CASCADE CONSTRAINTS']);

%% create the table
curs = exec(conn, ...
    ['create table my_alloc_full_lib_rand_new ( g250m_id number(38,0) CONSTRAINT↙
pk_alloc_lib_rand_new PRIMARY KEY, HT_1A number(12,4), HT_2A number(12,4), '...
     'HT_3A number(12,4),HT_4A number(12,4), HT_4B number(12,4), HT_4C number(12,4),↙
HT_5A number(12,4), HT_5B number(12,4), HT_5C number(12,4), '...
     'HT_6A number(12,4), HT_6B number(12,4), HT_6C number(12,4), HT_7A number(12,4),↙
HT_7B number(12,4), HT_7C number(12,4), HT_8C number(12,4) )']);

%% insert the data
colnames = {'g250m_id','HT_1A', 'HT_2A', 'HT_3A','HT_4A', 'HT_4B', 'HT_4C', 'HT_5A',↙
'HT_5B', 'HT_5C', ...
      'HT_6A', 'HT_6B', 'HT_6C', 'HT_7A', 'HT_7B', 'HT_7C', 'HT_8C' };

fastinsert(conn, 'my_alloc_full_lib_rand_new', colnames, Final_alloc);

%% commit
curs = exec(conn,'commit');


%%
curs = exec(conn, ...
        ['drop table my_remainder_lib_rand_new CASCADE CONSTRAINTS']);

%% create the table
curs = exec(conn, ...
    ['create table my_remainder_lib_rand_new ( taz number(38,0) CONSTRAINT↙
pk_alloc_rem_lib_rand_new PRIMARY KEY, HT_1A number(12,6), HT_2A number(12,6), '...
     'HT_3A number(12,6),HT_4A number(12,6), HT_4B number(12,6), HT_4C number(12,6),↙
HT_5A number(12,6), HT_5B number(12,6), HT_5C number(12,6), '...
     'HT_6A number(12,6), HT_6B number(12,6), HT_6C number(12,6), HT_7A number(12,6),↙
HT_7B number(12,6), HT_7C number(12,6), HT_8C number(12,6) )']);

%% insert the data
colnames = {'taz','HT_1A', 'HT_2A', 'HT_3A','HT_4A', 'HT_4B', 'HT_4C', 'HT_5A', 'HT_5B',↙
'HT_5C', ...
      'HT_6A', 'HT_6B', 'HT_6C', 'HT_7A', 'HT_7B', 'HT_7C', 'HT_8C' };

fastinsert(conn, 'my_remainder_lib_rand_new', colnames, [[1:2727]' Remainders]);

%% commit
curs = exec(conn,'commit');

%% close the connection
close(conn);
```

greedy_allocate_lib_random.m –

Run the allocation algorithm for a single TAZ using the let it be data with random accessibility.

```
function [Transactions Drem]= greedy_allocate_lib_random ( TAZ , passwd, conn)
% greedy allocate TAZ


%%
% allocation constraints [LandType, MinUnitAlloc, AreaPerUnit]
% LandType 1:BNW, 2:C/I, 3:RES
Z = [
    1, 1, 1.35
    1, 1, .6
    1, 1, .33
    1, 2, .09
    2, 2, .09
    3, 2, .09
    1, 6, .045
    2, 6, .045
    3, 6, .045
    1,20, .025
    2,20, .025
    3,20, .025
    1,51, .011
    2,51, .011
    3,51, .011
    3, 1,  0];

%% open the database connection
if ~exist('conn', 'var')
  conn = database('CRL','plg',passwd,'oracle.jdbc.driver.OracleDriver',...
     'jdbc:oracle:thin:@bulfinch.mit.edu:1526:');
    close_conn = 1;
else
    close_conn = 0;
end

%%
% need D(1:16) = #units demanded for each of the 16 housing types
curs = exec(conn, ...
    ['select newunits from tazresliblong where taz = ' num2str(TAZ) ' order by↙
devtype']);
curs=fetch(curs);
D = cell2mat(curs.data);
close(curs);

%%
% need Y(1:N,1:4) = amount of available resources (BNW_LIB,BNW_WOC,C/I,RES) in each grid
% cell
% Y = [G250m_id, BNW_LIB, BNW_WOC, C/I, RES, GRID_IN_TAZ]
curs = exec(conn, ...
['select t.g250m_id, g.bnw_lib, g.bnw_woc, g.comin, g.resi, max(t.grid_in_taz) ' ...
 'from tazgrid t, gridlanduse g where taz = ' num2str(TAZ) ' and t.g250m_id = g.g250m_id↙
'...
```

```
 'group by t.g250m_id, g.bnw_lib, g.bnw_woc, g.comin, g.resi order by t.g250m_id'] );
%    ['select g.*, t.grid_in_taz from gridlanduse g, tazgrid t where t.taz
%    = ' num2str(TAZ) ' and t.g250m_id = g.g250m_id'] ...
curs=fetch(curs);
Y = cell2mat(curs.data);
close(curs);

% convert sq meters to acres
Y(:,2:5) = Y(:,2:5) / 4046.85642;

%% get the accessibility measure for each grid cell (VMT)
% [G250m_id, access_dist]
curs = exec(conn, ...
    ['select g.g250m_id, g.eudist from my_random_vmt g, tazgrid t where t.taz = '↙
num2str(TAZ) ' and t.g250m_id = g.g250m_id order by t.g250m_id']);
curs=fetch(curs);
V = cell2mat(curs.data);
close(curs);


%% close the database connection
if close_conn
    close(conn);
end

%% rank the gridcells by vmt/household
%[R,I] = sort(V(:,3)./V(:,4));
%accessibility = @(x) exp(-(x.^2)./100^2);
%accessibility = @(x) (1./x);
accessibility = @(x) (x);

%% for each gridcell, allocate the housing type that increases vmt the most

% current desired amount of allocated houses
Dcurr = D;
Dcurr(Dcurr<0) = 0; % only allocate integer number of units
Dcurr = round(Dcurr);
Drem = D - Dcurr;
% weight by grid_in_taz
Ycurr = Y(:,1:5);
Ycurr(:,2:5) = Ycurr(:,2:5) .* repmat(Y(:,6),[1 4]);
Transactions = cell(3,1);

%Dcurr(Dcurr<Z(:,2)) = 0; % only allocate units over minimum
Access = accessibility(V(:,2));
%%
H_TABLE = [2 4 5]; %[BWN_LIB C/I RES]

for HOUSING_TYPE = 1:3

    HT = find(Z(:,1)==HOUSING_TYPE);
```

```
    while 1
        % find the best grid cell / housing type to allocate
        max_housing = floor(Ycurr(:,H_TABLE(HOUSING_TYPE)) * (1./Z(HT,3))');
        max_housing(isnan(max_housing)) = inf;
        demand = repmat(Dcurr(HT)',[size(max_housing,1) 1]);
        max_housing(demand < max_housing) = demand(demand < max_housing);

        if sum(Dcurr(HT)) == 0 || sum(max_housing(:)) == 0
            break;
        end

        [best bestI] = max(reshape(max_housing.* repmat(Access,[1 length(HT)]),[],1));

        [i j] = ind2sub(size(max_housing),bestI(1));

        % keep track of the book keeping
        Transactions{HOUSING_TYPE}(end+1,:) = [Ycurr(i,1) HT(j) max_housing(i,j)];
        Dcurr(HT(j)) = Dcurr(HT(j)) - max_housing(i,j);
        Ycurr(i, H_TABLE(HOUSING_TYPE)) = Ycurr(i,H_TABLE(HOUSING_TYPE)) - max_housing↙
(i,j)*Z(HT(j),3);

    end
end

%%
Drem = (Drem + Dcurr)';
```