Problem 1 Version 2:

Part 1:

Sets:

LOT_TYPE: This is a list of different types of lots to be built on demolished land (Houses, Duplex, and Parks)

Variables:

DEMO: An integer representing the number of demolished lots

COUNT: An array of integers that map to the different lot types and represent the number of each type that should be built

Parameters:

acres: An array of real numbers representing the amount of land needed to build the corresponding lot type

cost: An array of real numbers representing the cost of building a single unit of each lot type

profit: The expected tax value of a single unit of each lot type

min_percent: The minimum percentage of total new units each lot type should have

budget: The amount of money available for demolition and construction

Constraints:

lots_avail: A constraint that keeps DEMO (# of demolished lots) within the number of available lots. i.e.) We can't demolish 400 lots when we only have 350

DEMO <= 350

demo_acres: A constraint that makes sure we have enough land for each new unit. The available land is the mount of land provided from each demolished lot multiplied by the number of demolished lots. This must be greater than or equal to the amount of land required for our new lots.

.25*DEMO >= sum(#lots*acres of lot type) for all lot types

costs: A constraint that keeps our spending within our budget. We have 2 costs. The cost of demolition and the cost of construction. The total of these 2 costs must be less than our budget.

DEMO*3000+sum(cost*count)for each lot type <= budget

divers: A constraint that keeps us from picking only the most profitable lot. The number of units of each lot type must be at least of min_percentage for that respective lot type.

(Lot type units)/(total new units) >= min_percentage

Objective:

Maximize Tax: The goal of our code is to maximize the tax revenue for the city. This is simply the expected profit for each unit multiplied by the number of units for each lot type

Sum(# units * 1 unit profit) for each lot type

Part 2)

```
reset;
option solver cplex;
option cplex_options 'sensitivity';

#---------------------------Sets------------------------------------
set LOT_TYPE;   #New lot options

#------------------------Parameters---------------------------------

param acres {LOT_TYPE} >= 0;            #Acres required for lot type
param cost {LOT_TYPE} >= 0;         #Cost of building new lot type
param profit {LOT_TYPE} >= 0;       #Expected profit from new lot type
param min_percent {LOT_TYPE} >= 0;  #Percent of all lots for each type
param budget = 15000000;            #Federal Grant Budget


#-------------------------Decision Variables--------------------------

var DEMO integer >= 0;              #Number of lots to demolish
var COUNT {LOT_TYPE} integer >= 0;  #Number of lots to build for each type

#-------------------------Objective Function--------------------------

maximize Tax:  (sum{l in LOT_TYPE} profit[l]*COUNT[l]);

#--------------------------Constraints--------------------------------

subject to lots_avail: DEMO <= 350;                                             #Only 350 lots available to demolish
subject to demo_acres: .25*DEMO >= sum{l in LOT_TYPE} COUNT[l]*acres[l];        #Can only build on available land
subject to costs: DEMO*3000 + sum{l in LOT_TYPE} cost[l]*COUNT[l] <= budget;    #Can only use grant money
subject to divers {l in LOT_TYPE}: COUNT[l] >= min_percent[l]*(sum{t in LOT_TYPE} COUNT[t]);#Must meet min requirements for each lot type


#----------------------------Data--------------------------------------

data "C:\Users\Dylan\Documents\OU\DSA5113\FINAL\dsteimel_final_p1.dat";


solve;

display DEMO;
display COUNT;
display Tax;
```


```
#-------------------------Objective Function--------------------------

maximize Tax:  (sum{l in LOT_TYPE} profit[l]*COUNT[l]);
```


```
DEMO = 205

COUNT [*] :=
DUPLEX   76
 HOUSE   10
  PARK   14
;

Tax = 239000
```

Problem 2 Version 7

Problem 3 Version 1

Part i)

Below are the results of each iteration of the hill climbing function with the given parameters.

The steps are as follows:

1. Get Neighborhood.
2. For each neighbor check if new best is found.
3. If no new best then end.
4. Else move to new best and repeat (and track best position and evaluation).

```
Total number of solutions checked:  4
Solutions checked this iteration:  [[1, -3], [-1, -3], [0, -2], [0, -4]]
Best value found so far:  -2.0
Best solution fo far:  [0, -4]
Better solution found:  True

Total number of solutions checked:  8
Solutions checked this iteration:  [[1, -4], [-1, -4], [0, -3], [0, -5]]
Best value found so far:  -2.5
Best solution fo far:  [0, -5]
Better solution found:  True

Total number of solutions checked:  12
Solutions checked this iteration:  [[1, -5], [-1, -5], [0, -4], [0, -6]]
Best value found so far:  -3.018108996753427
Best solution fo far:  [-1, -5]
Better solution found:  True

Total number of solutions checked:  16
Solutions checked this iteration:  [[0, -5], [-2, -5], [-1, -4], [-1, -6]]
Best value found so far:  -3.1509688379721745
Best solution fo far:  [-1, -6]
Better solution found:  True

Total number of solutions checked:  20
Solutions checked this iteration:  [[0, -6], [-2, -6], [-1, -5], [-1, -7]]
Best value found so far:  -3.1509688379721745
Best solution fo far:  [-1, -6]
Better solution found:  False

Final number of solutions checked:  20
Best value found:  -3.1509688379721745
Best solution:  [-1, -6]
[Finished in 0.241s]
```

Part ii)

Note: We will let our guiding function be the neighbor closest to B by Euclidean distance.

Now instead of moving to the best evaluated position, we move to the point closest to point B.

The resulting steps are:

1. Get Neighborhood.
2. For each neighbor check if new best is found (store pos and value if true) and calculate distance to B.
3. Move to neighbor with the minimum distance to B.
4. If neighbor is B, end. Else repeat

```
Total number of solutions checked:  4
Solutions checked this iteration:  [[0, -3], [-2, -3], [-1, -2], [-1, -4]]
Next position:  [-2, -3]
Best value found so far:  -2.408902133301636
Best solution fo far:  [-1, -4]
Reached B:  False

Total number of solutions checked:  8
Solutions checked this iteration:  [[-1, -3], [-3, -3], [-2, -2], [-2, -4]]
Next position:  [-2, -2]
Best value found so far:  -2.408902133301636
Best solution fo far:  [-1, -4]
Reached B:  False

Total number of solutions checked:  12
Solutions checked this iteration:  [[-1, -2], [-3, -2], [-2, -1], [-2, -3]]
Next position:  [-3, -2]
Best value found so far:  -3.7005928892065527
Best solution fo far:  [-3, -2]
Reached B:  False

Total number of solutions checked:  16
Solutions checked this iteration:  [[-2, -2], [-4, -2], [-3, -1], [-3, -3]]
Next position:  [-3, -1]
Best value found so far:  -3.7005928892065527
Best solution fo far:  [-3, -2]
Reached B:  True

Final number of solutions checked:  16
Best value found:  -3.7005928892065527
Best solution:  [-3, -2]
[Finished in 0.215s]
```

Part iii)

P = e^-((f(s1)-f(s2))/t)

f(s) = evaluation of solution

t = temperature

f(0,-4) = -2,   f(1,-4) = -1.591,   f(-1,-4) = 1.591

P((0,-4)->(1,-4)) = .873

P((0,-4)->(-1,-4)) = .302   Note: Our goal is to minimize f(s) so this makes sense

Problem 4 Version 4

Part i)

f(10001) = 5 + 0 + 0 + 0 + 1 = 6

f(00101) = 0 + 0 + 3 + 0 + 1 = 4

f(01011) = 0 + 4 + 0 + 2 + 1 = 7

f(11000) = 5 + 4 + 0 + 0 + 0 = 9

Total Fitness = 6 + 4 + 7 + 9 = 26

Roulette Selection Probabilities = individual fitness / total fitness

p(10001) = 6 / 26 = .231

p(00101) = 4 / 26 = .154

p(01011) = 7 / 26 = .269

p(11000) = 9 / 26 = .346

Part ii)

Parent 1 = 11000

Parent 2 = 00101

Parent 1 split: 11 000

Parent 2 split: 00 101

Child 1: 11101

Child 2: 00000

Part iii) f(11101) = 5 + 4 + 3 + 0 + 1 = 13

Problem 5 Version 2

Part i)

$V(t+1) = 1*V(t) + 1*.5*(P(i)-X(i,t)) + 1*.15*(P(g) – X(i,t))$

$V(t)$ = current velocity = (1,0,1)

$P(i)$ = personal best position = (10,13,8)

$X(i,t)$ = current position = (14,5,2)

$P(g)$ = global best = (8,2,0)

$V(t+1) = (1,0,1) + .5*((10,13,8)-(14,5,2)) + .15*((8,2,0)-(14,5,2))$

$= (1,0,1) + .5*(-4,8,6) + .15*(-6, -3, -2)$

$= (1,0,1) + (-2,4,3) + (-.9, -.45, -.3)$

$V(t+1) = (-1.9, 3.55, 3.7)$

New Position = $X(i,t) + V(t) = (14, 5, 2) + (1, 0, 1) = (15, 5, 3)$

Part ii)

$V(t+1) = 1*V(t) + 1*.5*(P(i)-X(i,t)) + 1*.15*(P(g) – X(i,t))$

$V(t)$ = current velocity = (1,0,1)

$P(i)$ = personal best position = (10,13,8)

$X(i,t)$ = current position = (14,5,2)

$P(g)$ = neighborhood best = (18,7,5)

$V(t+1) = (1,0,1) + .5*((10,13,8)-(14,5,2)) + .15*((18,7,5)-(14,5,2))$

$= (1,0,1) + .5*(-4,8,6) + .15*(4, 2, 3)$

$= (1,0,1) + (-2,4,3) + (.6, .3, .45)$

$V(t+1) = (-.4, 4.3, 4.45)$

New Position = $X(i,t) + V(t) = (14, 5, 2) + (1, 0, 1) = (15, 5, 3)$